17

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES
CUAUTITLAN

"LOGICA DIFUSA APLICADA EN EL COP8"

T E S I S

QUE PARA OBTENER EL TITULO DE :

INGENIERO MECANICO ELECTRICISTA

P R E S E N T A :

OSCAR CERVANTES MARTINEZ

ASESOR : ING. JORGE BUENDIA GOMEZ

CUAUTITLAN IZCALLI, EDO. DE MEX. 2000

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN
UNIDAD DE LA ADMINISTRACION ESCOLAR
DEPARTAMENTO DE EXAMENES PROFESIONALES

U N A M.
FACULTAD DE ESTUDIOS
SUPERIORES CUAUTITLAN

ASUNTO: VOTOS APROBATORIOS

DEPARTAMENTO DE
EXAMENES PROFESIONALES

DR. JUAN ANTONIO MONTARAZ CRESPO
DIRECTOR DE LA FES CUAUTITLAN
P R E S E N T E

ATN. Q. Ma. del Carmen Garcia Mijares
Jefe del Departamento de Exámenes
Profesionales de la FES Cuautitlán

Con base en el art. 28 del Reglamento General de Exámenes, nos permitimos comunicar a usted que revisamos la TESIS:

"Lógica Difusa aplicada en el COP8"

que presenta el pasante: Oscar Cervantes Martínez
con número de cuenta: 8637292-1 para obtener el título de :
Ingeniero Mecánico Electricista

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el EXAMEN PROFESIONAL correspondiente, otorgamos nuestro VOTO APROBATORIO.

A T E N T A M E N T E
"POR MI RAZA HABLARA EL ESPIRITU"
Cuautitlán Izcalli, Méx. a 16 de Agosto del 2000

PRESIDENTE	Ing. Juan González Vega	
VOCAL	Ing. Jorge Buendia Gómez	
SECRETARIO	Ing. Margarita López López	
PRIMER SUPLENTE	Ing. Jorge Ramírez Rodríguez	
SEGUNDO SUPLENTE	Ing. Adriana Corina Sandoval García	

AGRADECIMIENTOS

A MI FAMILIA:

A Tere (mi madre)

Por todo el amor que me ha dado.

Por haberme enseñado la nobleza del trabajo.

Por darme la oportunidad de ser un hombre libre y de bien.

A Manuel (mi hermano)

Por ser más que un hermano, más que un amigo.

Por enseñarme que la sabiduría nos hace más

Libres y por ser parte de mi inspiración en el estudio.

A Rebeca (mi hermana)

Por ser un buen ejemplo, por la confianza que me da y

por compartirme parte de su gran carácter.

A Rocío, Esmeralda y Miguel (mi hermana y mis sobrinos)

Por todo el amor que me han entregado.

Agradezco a todos ellos el gran amor e invaluable apoyo

que me brindaron en los momentos que más lo necesite.

Nunca olvidaré su solidaridad.

A MIS COMPAÑEROS:

A todos aquellos que fueron parte de mi vida cuando realice mis estudios,

aunque pocos, les agradezco su amistad.

Especialmente a Yareli por haber compartido gran parte

de mi formación académica y personal.

A MIS PROFESORES:

A todos mis profesores, los buenos y los malos.

A los buenos por enseñarme algunas de las mejores cosas

que aprendí fuera de mi hogar.

A los malos por enseñarme como NO debo actuar.

INDICE

INTRODUCCION

La lógica difusa es una nueva tecnología, que ha surgido como una poderosa herramienta, para el control de sistemas complejos e industriales, así como para máquinas domesticas, de entretenimiento, diagnostico de sistemas y otros sistemas expertos. La lógica difusa expresa leyes operacionales de un sistema en términos lingüísticos en lugar de ecuaciones matemáticas. La mayoría de los sistemas son difíciles en la precisión de su modelo, a la vez que sus ecuaciones matemáticas son complejas, sin embargo, los términos lingüísticos dan un método útil para definir las características operacionales de un sistema. Los sistemas basados en reglas de lógica difusa aplican este método para resolver muchos problemas típicos del mundo real, especialmente donde un sistema tiene problemas con su modelo matemático, donde el control es realizado por un operador o experto, o donde la ambiguedad y falta de claridad son comunes.

La lógica difusa fue propuesta en un documento llamado "Conjuntos difusos" en 1965 por un profesor de la Universidad de Berkley, llamado Lotfi A. Zadeh. Sin embargo su gran crecimiento se dio en Japón y de ahí se a extendido a Estados Unidos y Europa. En la actualidad una de las áreas más activas de la aplicación de lógica difusa es en sistemas de control.

Debido a que en aplicaciones con procesadores el aprendizaje constante no es lo normal, a diferencia de un modelo de lógica difusa que se adapta bien a este caso, el modelo de lógica difusa con ayuda de algunas técnicas de aproximación, puede ser fácilmente codificado en un procesador. Aunque el resultado final se aproxima al resultado deseado éste es suficientemente aceptable para su desempeño en el mundo real.

El propósito de este trabajo es el de servir como una introducción a la lógica difusa y al microcontrolador COP8 Para ello se proporcionan las bases de la lógica difusa así como aspectos del control difuso. La parte referente al microcontrolador pretende servir como información de consulta para aquellas personas que se estén introduciendo al diseño con este dispositivo. Para completar el trabajo se desarrolla un ejemplo de aplicación con ayuda de la herramienta de desarrollo de lógica difusa NeuFuz y se implanta en el microcontrolador de ocho bits OTP

COP8SAx. Los siguientes párrafos describen la estructura usada en el desarrollo de este documento.

El primer capítulo esta dividido en dos partes: la primera trata de los conjuntos clásicos y la segunda trata las bases de la lógica difusa. La primer parte empieza con un repaso breve de la teoría de conjuntos tradicional definiendo el concepto de universo y subconjuntos así como sus relaciones. Enseguida se describen las operaciones y propiedades que son utilizadas en los conjuntos ordinarios, y finalmente el mapeo de conjuntos ordinarios en funciones. La segunda parte trata del concepto del conjunto difuso, las operaciones y propiedades usadas en ellos. El siguiente tema menciona las características de las funciones de membresía y la definición de producto y suma de subconjunto difusos. Termina el capítulo con los temas de conjunto difuso condicionado y de proposiciones de lógica difusa. La segunda parte del capítulo sirve como un comparativo de la lógica ordinaria o clásica y la lógica difusa.

El capítulo dos también cuenta con dos partes: en la primera se habla sobre el método clásico de control, se comenta sobre los sistemas de control y la compensación de estos. Esta parte termina con la descripción de un problema de control en donde se marcan las diferencias entre la solución por medio del control tradicional y de la técnica de control difuso. La segunda parte trata sobre los sistemas de control difuso, así como de la teoría de la lógica difusa aplicada a estos sistemas. A continuación se termina el capítulo explicando las partes integrantes del control difuso: difusificación, reglas de inferencia y desdifusificación; haciendo uso del ejemplo usado en la primera parte de este capítulo.

En el capítulo tercero se describen algunas herramientas de desarrollo de lógica difusa, continuando con la descripción detallada del programa usado para el desarrollo de la aplicación en este trabajo, NeuFuz. Primero se discuten las bases de NeuFuz, la lógica difusa y el concepto de red neuronal, que es utilizado en la búsqueda de la solución difusa. Enseguida se describen los fundamentos de NeuFuz, la obtención del conjunto de datos de entrenamiento necesarios para la red neuronal, el proceso de entrenamiento y el reconocimiento de la convergencia de la red neuronal. El tema siguiente discute la evaluación y prueba de la solución, y por último el capítulo

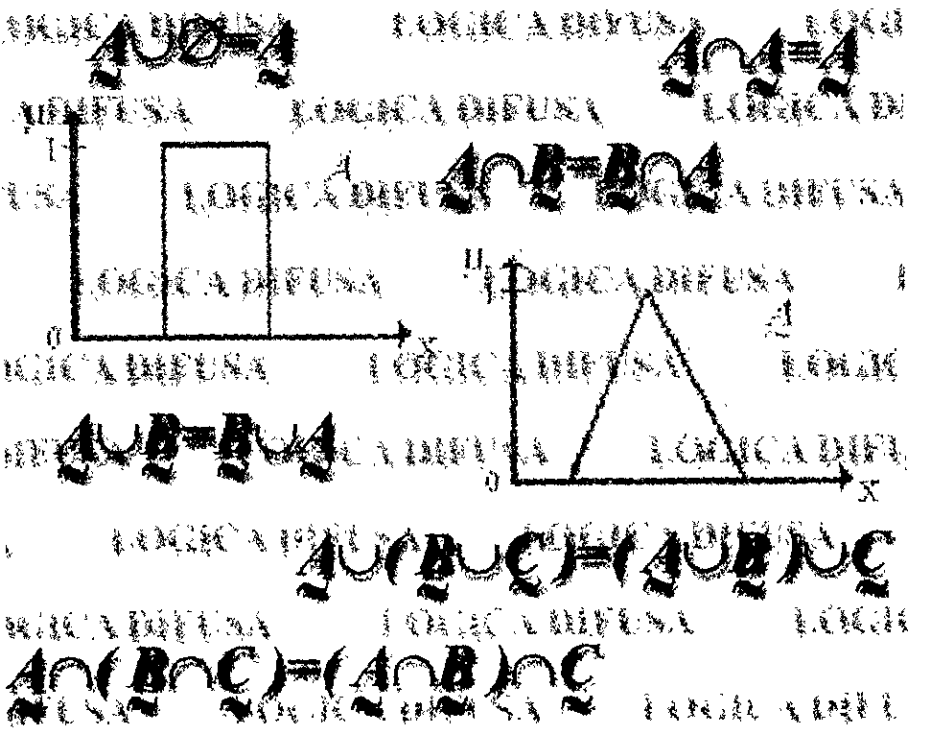
termina con la traducción de las reglas de lógica difusa, en código de lenguaje ensamblador pata el COP8 (Procesadores Orientados de Control), obtenidas en el paso anterior.

El capítulo cuatro empieza introduciendo las características de los microcontroladores en forma general, su arquitectura y su operación interna. Se describen por completo las características del microcontrolador COP8SAx7. Algunas de estas explican que el microcontrolador no requiere de componentes externos, esto quiere decir que el R/C, el power on reset (POR), las resistencias pull-up, schmitt-trigers y los diodos de protección se encuentran incluidos en el diseño interno del microcontrolador. También se describen otras cualidades como la operación del timer (temporizadores T0 y T1), las propiedades de bajo consumo(modos HALT e IDLE), las Interrupciones (enmascarada y no enmascarada, VIS y del puerto L), los monitores de operación WATCHDOG y de reloj, la interfase de entrada MICROWIRE/PLUS. El capítulo termina con el mapa de memoria y un resumen del conjunto de instrucciones, la descripción detallada del conjunto de instrucciones se encuentra en el apéndice A.

En el capítulo número cinco se desarrolla la aplicación de lógica difusa que es implantada en el microcontrolador COP8. La aplicación trata sobre el control de la velocidad en un motor de corriente continua. Lo primero que se hace es el adiestramiento de la red neuronal con el conjunto de datos de entrenamiento, el cual es obtenido de las características propias del motor. Una vez que la red aprende el conjunto de datos de entrenamiento se editan las funciones de membresía y se prueba y evalúa la solución. Una vez que se ha obtenido una solución que satisface los requerimientos de la aplicación se procede con la generación del código ensamblador y de la anexión del código principal (programa principal), el cual hará el llamado a todas las rutinas de la aplicación de lógica difusa. En los apartados siguientes se da una descripción del hardware usado y la interconexión con el microcontrolador. Por último se realiza la prueba y depuración del software y hardware registrando los resultados obtenidos. Un apartado final de conclusiones termina con el capítulo y con el desarrollo de la aplicación.

CAPITULO 1

BASES DE LOGICA DIFUSA



CAPITULO 1 BASES DE LÓGICA DIFUSA

1.1. Repaso de teoría de conjuntos ordinarios.

1.1.1. Definiciones.

Primero definamos a U como el conjunto de todos los objetos con las mismas características, llamado el universo, y por x a cada uno de sus elementos; estos pueden ser discretos y finitos o continuos e infinitos. Un conjunto A consiste en algunos elementos de U, llamado también subconjunto de U. Llamamos n_x al total de los elementos de U. El conjunto que no tiene ningún elemento es llamado el conjunto vació $\emptyset$. Ahora tomemos la siguiente notación:

$x \in U \rightarrow x$ pertenece a U

$x \in A \rightarrow x$ pertenece a A

$x \notin A \rightarrow x$ no pertenece a A

También si A y B pertenecen a U entonces:

$A \subset B \rightarrow A$ es subconjunto de $B \rightarrow \forall x \in A$, entonces $x \in B$

$A \subseteq B \rightarrow A$ esta incluido en B

$A = B \rightarrow A \subseteq B$ y $B \subseteq A$

1.1.2. Operaciones y propiedades con conjuntos ordinarios.

Sean A y B dos conjuntos contenidos en U y x sus elementos individuales, entonces:

Unión.- $A \cup B = \{x | x \in A \text{ o } x \in B\}$

La unión entre dos conjuntos representa todos aquellos elementos del universo U, que pertenecen al conjunto A, al conjunto B o a ambos.

Intersección - $A \cap B = \{x | x \in A \text{ y } x \in B\}$

Esta operación representa a aquellos elementos del universo U, que estén en el conjunto A y al mismo tiempo en el conjunto B.

Complemento - $\overline{A} = \{x | x \notin A, x \in U\}$

El complemento de un conjunto A, esta definido como la colección de los elementos del universo U los cuales no están contenidos en A.

Diferencia.- $A \setminus B = \{x | x \in A, x \notin B\}$

Es la operación en la cual los elementos del universo U que están en el conjunto A no lo están simultáneamente en el conjunto B.

En la figura 1.1 se muestran diagramas de Veen que muestran la unión, la intersección y el complemento de dos conjuntos ordinarios.

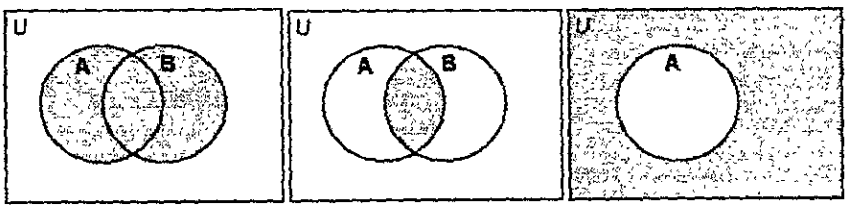


Figura 1. 1. Unión, Intersección de A y B, y Complemento de A.

Propiedades de conjuntos ordinarios:

Conmutativa	$A \cup B = B \cup A$
	$A \cap B = B \cap A$
Asociativa	$A \cup (B \cup C) = (A \cup B) \cup C$
	$A \cap (B \cap C) = (A \cap B) \cap C$
Idempotencia	$A \cup A = A$
	$A \cap A = A$
Identidad	$A \cup \varnothing = A$
	$A \cap U = A$
	$A \cap \varnothing = \varnothing$
	$A \cup U = U$
Transitividad	Si $A \subseteq B \subseteq C$, entonces $A \subseteq C$
Involución	$\overline{\overline{A}} = A$

Propiedades especiales:

Ley de Exclusión	$A \cup \overline{A} = U$
Ley de Contradicción	$A \cap \overline{A} = \varnothing$
Leyes de DeMorgan	$\overline{(A \cap B)} = \overline{A} \cup \overline{B}$
	$\overline{(A \cup B)} = \overline{A} \cap \overline{B}$

1.1.3. Mapeo de conjuntos ordinarios en funciones.

En forma específica el mapeo es usado para transportar elementos o conjuntos de un universo a elementos o conjuntos de otro universo. Este es un concepto importante con relación a la teoría de conjuntos y a la teoría de funciones. Definimos a X e Y como dos universos diferentes de información. Si un elemento x esta contenido en X y este corresponde a un elemento y contenido en Y, designamos a esto como un mapeo de X a Y o $f: X \rightarrow Y$. Por lo tanto al mapear, la función característica μ_A queda:

$$\mu_A(x) = \begin{cases} 1, x \in A \\ 0, x \notin A \end{cases}$$

donde μ_A expresa la membresía en el conjunto A para todos los elementos x en el universo Esta idea de membresía, se refiere al mapeo de un elemento x en el universo X a uno de los dos elementos en el universo Y, por ejemplo los elementos 0 ó 1.

Para cualquier conjunto A definido sobre el universo X existe un conjunto de valores $V(A)$ bajo el mapeo de la función característica μ_A. Por convención el conjunto vacío es designado al valor de membresía 0 y el conjunto completo X es asignado al valor de membresía 1.

De tal forma que si definimos un conjunto universo $U = \{x_1, x_2, x_3, x_4, x_5, ..., x_n\}$, un subconjunto $A = \{x_2, x_3, x_5\}$ y haciendo uso de la función característica, entonces, podemos escribir:

$$\mu_A(x_1) = 0,\ \mu_A(x_2) = 1,\ \mu_A(x_3) = 1,\ \mu_A(x_4) = 0,\ \mu_A(x_5) = 1, ...,\ \mu_A(x_n) = 0.$$

con lo que podemos representar al conjunto A haciendo acompañar a los elementos de U con su valor de membresía

$$A = \{(x_1, 0), (x_2, 1), (x_3, 1), (x_4, 0), (x_5, 1), ..., (x_n, 0)\}.$$

Definamos ahora dos conjuntos en el universo U, conjuntos A y B En términos de la función característica las operaciones de conjuntos toman la siguiente forma.

Unión.- $A \cup B \rightarrow \mu_{A \cup B}(x) = \mu_A(x) + \mu_B(x) = MAX(\mu_A(x), \mu_B(x))$

donde la operación (+) es la operación "suma booleana" definida por la tabla:

(+)	0	1
0	0	1
1	1	1

Intersección.- $A \cap B \rightarrow \mu_{A \cap B}(x) = \mu_A(x) \bullet \mu_B(x) = MIN(\mu_A(x), \mu_B(x))$

la operación (•) es llamada "producto booleano" y es definida por la tabla.

(•)	0	1
0	0	0
1	0	1

1.2. Conjuntos difusos.

En conjuntos ordinarios, o conjuntos precisos, la transición entre membresía y no membresía, en un conjunto dado, para un elemento del universo es abrupta y bien definida, dicho de otra forma es precisa. Esta transición entre varios grados de membresía puede cambiar la idea de que los límites de un conjunto difuso son vagos y ambiguos

1.2.1. El concepto de conjunto difuso.

Consideremos al conjunto $U = \{x_1, x_2, x_3, x_4, x_5, \ldots, x_n\}$ que llamaremos el conjunto universo. Ahora definamos un subconjunto $A = \{x_2, x_3, x_5\}$ de U. Los elementos del conjunto universo pertenecen o no al subconjunto A, lo uno o lo otro. Por lo tanto la función característica solo puede tomar los valores 0 o 1. Este concepto es suficiente para muchas áreas de aplicación. Pero se pueden encontrar fácilmente situaciones que requieran mayor flexibilidad

Supongamos ahora que la función característica toma un valor cualquiera en el segmento [0,1] por lo que un elemento x_i de U se encontraría en cualquiera de las siguientes posibilidades:

No pertenece a A $(\mu_A = 0)$

Pertenece un poco a A (μ_A cercano a 0)

Pertenece bastante a A (μ_A no muy cercas a 0, ni muy cercas a 1)

Pertenece fuertemente a A (μ_A muy cercano a 1)

Pertenece a A $(\mu_A = 1)$

Consideremos entonces la expresión siguiente:

$$A = \{(x_1|0.2), (x_2|0), (x_3|0.3), (x_4|1), (x_5|0.8)\}$$

donde x_i en un elemento del universo U y donde el número colocado enseguida de la barra es el valor de la función característica de este elemento, a esta expresión se le conoce como "subconjunto difuso" de U, y se representa como $\underset{\sim}{A} \subset U$. Por consiguiente el subconjunto difuso $\underset{\sim}{A}$ contiene:

Un poco de x_1 $(\mu_{\underset{\sim}{A}} = 0.2)$ o $x_1 \underset{0.2}{\in} \mu_{\underset{\sim}{A}}$

Nada de x_2 $(\mu_{\underset{\sim}{A}} = 0)$ o $x_2 \underset{0}{\in} \mu_{\underset{\sim}{A}}$

Un poco mas de x_3 $(\mu_{\underset{\sim}{A}} = 0.3)$ o $x_3 \underset{0.3}{\in} \mu_{\underset{\sim}{A}}$

Por completo a x_4 $(\mu_{\underset{\sim}{A}} = 1)$ o $x_4 \underset{1}{\in} \mu_{\underset{\sim}{A}}$

Una gran parte de x_5 $(\mu_{\underset{\sim}{A}} = 0.8)$ o $x_5 \underset{0.8}{\in} \mu_{\underset{\sim}{A}}$

Por comodidad se toman como equivalentes $\underset{1}{\in} \Leftrightarrow \in$ y $\underset{0}{\in} \Leftrightarrow \notin$ para indicar la membresía y no membresía respectivamente. Entonces y de acuerdo al desarrollo mostrado, la membresía, o pertenencia, toma una amplitud que conduce a desarrollos de mucha utilidad

Con lo visto hasta ahora se puede dar una definición del concepto de conjunto difuso.

Sea U un conjunto finito o no, y x un elemento de U; entonces un "subconjunto difuso" $\underset{\sim}{A}$ de U es un conjunto de pares ordenados

$$\{(x, \mu_{\underset{\sim}{A}}(x))\}, \forall x \in U$$

donde $\underset{\sim}{A}$ es una "función característica de membresía" que toma sus valores en un conjunto totalmente ordenado M que indica el "grado" o "nivel" de membresía, al conjunto M se le llama "conjunto de membresía".

Si, $M = \{0,1\}$, el subconjunto difuso $\underset{\sim}{A}$ se convierte en un subconjunto no difuso o conjunto ordinario. Esto se muestra en la figura 1.2 siguiente para conjuntos difusos y conjuntos ordinarios respectivamente.

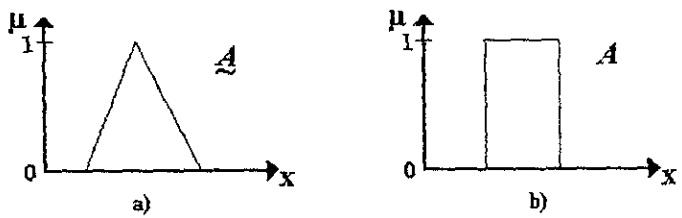


Figura 1. 2. Funciones de membresía para conjuntos : a)Difusos , b)Ordinarios.

1.2.2. Operaciones con conjuntos difusos.

Ahora que se a dado la idea de lo que son los conjuntos difusos, se introducen las operaciones sobre estos De forma similar que las operaciones de los conjuntos ordinarios se tienen. unión, intersección y complemento.

Tomaremos en cuenta los que en su primer trabajo acerca de los conjuntos difusos L. A. Zadeh propuso. el operados mínimo (MIN) para la intersección y el operador máximo (MÁX) para la unión de los conjuntos difusos. Es fácil observar que estos operadores coinciden con las operaciones de unión e intersección de los conjuntos ordinarios si se consideran solamente los grados de membresía como 0 y 1

Para las siguientes definiciones sean:

U un conjunto y $M = \{0,1\}$ su conjunto de membresía asociado, $\underset{\sim}{A}$ y $\underset{\sim}{B}$ dos subconjuntos difusos de U.

Unión.- la unión se define:

$$\underset{\sim}{A} \cup \underset{\sim}{B}$$

por el subconjunto difuso más pequeño que contiene a $\underset{\sim}{A}$ como a $\underset{\sim}{B}$. Es decir:

$$\forall x \in U \quad : \quad \mu_{\underset{\sim}{A} \cup \underset{\sim}{B}}(x) = \text{MAX}\,(\mu_{\underset{\sim}{A}}(x), \mu_{\underset{\sim}{B}}(x))$$

En la figura 1.3 se muestra un diagrama de Veen extendido para conjuntos difusos bajo esta operación:

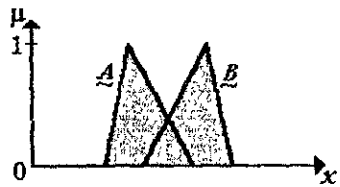


Figura 1. 3. Unión de los subconjuntos difusos $\underset{\sim}{A}$ y $\underset{\sim}{B}$.

Intersección - Se define la intersección:

$$\underset{\sim}{A} \cap \underset{\sim}{B}$$

por el subconjunto difuso más grande contenido, a la vez, en $\underset{\sim}{A}$ y en $\underset{\sim}{B}$; es decir:

$$\forall x \in U \quad : \quad \mu_{\underset{\sim}{A} \cap \underset{\sim}{B}}(x) = \text{MIN}(\mu_{\underset{\sim}{A}}(x), \mu_{\underset{\sim}{B}}(x))$$

El diagrama de Veen extendido para esta operación se muestra en la figura 1 4 a continuación.

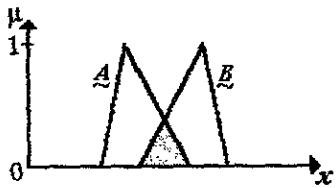


Figura 1. 4. Intersección de los subconjuntos difusos $\underset{\sim}{A}$ y $\underset{\sim}{B}$

Complemento.- Sean $\underset{\sim}{A}$ y $\overline{\underset{\sim}{A}}$ dos conjuntos difusos de U; se dice que $\underset{\sim}{A}$ y $\overline{\underset{\sim}{A}}$ son complementarios sí:

$$\boxed{\forall x \in U \quad : \quad \mu_{\overline{\underset{\sim}{A}}}(x) = 1 - \mu_{\underset{\sim}{A}}(x)}$$

evidentemente se tiene:

$$\underset{\sim}{A} = (\overline{\underset{\sim}{A}})$$

El diagrama correspondiente de Veen extendido se muestra en la figura 1.5.

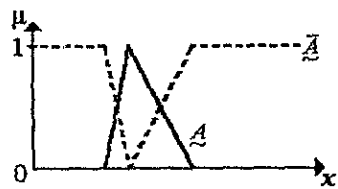


Figura 1. 5. Complemento del subconjunto difuso $\underset{\sim}{A}$

1.2.3. Propiedades de los conjuntos difusos.

Los conjuntos difusos siguen las mismas propiedades que los conjuntos ordinarios. Es por ello y debido a que los valores de membresía de un conjunto ordinario son un subconjunto del intervalo [0,1], por lo que los conjuntos ordinarios pueden tratarse como un caso especial de los conjuntos difusos.

Las propiedades más usadas de los conjuntos difusos son.

Conmutativa $\underset{\sim}{A} \cup \underset{\sim}{B} = \underset{\sim}{B} \cup \underset{\sim}{A}$

$$\underset{\sim}{A} \cap \underset{\sim}{B} = \underset{\sim}{B} \cap \underset{\sim}{A}$$

Asociativa $\underset{\sim}{A} \cup (\underset{\sim}{B} \cup \underset{\sim}{C}) = (\underset{\sim}{A} \cup \underset{\sim}{B}) \cup \underset{\sim}{C}$

$$\underset{\sim}{A} \cap (\underset{\sim}{B} \cap \underset{\sim}{C}) = (\underset{\sim}{A} \cap \underset{\sim}{B}) \cap \underset{\sim}{C}$$

Distributiva $\underset{\sim}{A} \cup (\underset{\sim}{B} \cap \underset{\sim}{C}) = (\underset{\sim}{A} \cup \underset{\sim}{B}) \cap (\underset{\sim}{A} \cup \underset{\sim}{C})$

$$\underset{\sim}{A} \cap (\underset{\sim}{B} \cup \underset{\sim}{C}) = (\underset{\sim}{A} \cap \underset{\sim}{B}) \cup (\underset{\sim}{A} \cap \underset{\sim}{C})$$

Idempotencia $\underset{\sim}{A} \cup \underset{\sim}{A} = \underset{\sim}{A}$

$$\underset{\sim}{A} \cap \underset{\sim}{A} = \underset{\sim}{A}$$

Identidad: $\underset{\sim}{A} \cup \varnothing = \underset{\sim}{A}$

$$\underset{\sim}{A} \cap U = \underset{\sim}{A}$$

$$\underset{\sim}{A} \cap \varnothing = \varnothing$$

$$\underset{\sim}{A} \cup U = U$$

Transitividad Si $\underset{\sim}{A} \subseteq \underset{\sim}{B} \subseteq \underset{\sim}{C}$,

Entonces $\underset{\sim}{A} \subseteq \underset{\sim}{C}$

Involución $\underset{\sim}{A} = \overline{(\overline{\underset{\sim}{A}})}$

1.2.4. Características de las funciones de membresía.

Ya que toda la información contenida en un conjunto difuso es descrita por la función de membresía, es de utilidad desarrollar un léxico de términos para describir varias características especiales de esta función. En la figura 1.6 se muestra un esquema ilustrativo que nos servirá de guía.

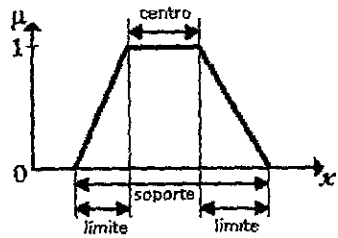


Figura 1. 6. Centro, soporte y límites de un conjunto difuso.

Centro.- Es aquella región del universo que se caracteriza por tener por completo la membresia del conjunto dado. Es decir, el centro comprende aquellos elementos del universo donde: $\mu_{\underset{\sim}{A}}(x)=1$.

Soporte.- Se define como la región del universo que se caracteriza por no tener ningún elemento con membresía igual a cero Es decir, el soporte abarca aquellos elementos del universo, donde: $\mu_{\underset{\sim}{A}}(x)\neq 0$.

Limites.- Comprende aquella región del universo que contiene elementos que no tienen membresía cero, pero que tampoco presentan una membresía completa. Esto es, aquellos elementos del universo donde $0<\mu_{\underset{\sim}{A}}(x)<1$. Estos elementos presentan cierto grado difuso de membresía en el conjunto dado.

Conjunto difuso Normal.- Es aquel que tiene una función de membresia en la que al menos uno de sus elementos del universo posee un valor de membresía completo, es decir $\mu_{\underset{\sim}{A}}(x)=1$. A este elemento se le llama "prototipo" del conjunto. La figura 1.7 ilustra lo anterior.

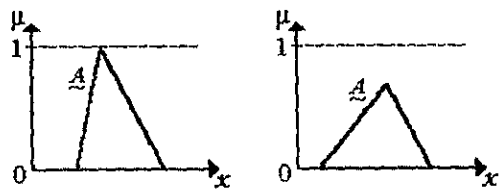


Figura 1. 7. Conjuntos normal (izquierda) y no-normal (derecha).

1.2.5. Producto y suma de dos subconjuntos difusos.

Sean :

U un conjunto y $M = [0,1]$ su conjunto de membresía

$\underset{\sim}{A}$ y $\underset{\sim}{B}$ dos conjuntos difusos de U.

se define el "producto algebraico" de $\underset{\sim}{A}$ y $\underset{\sim}{B}$, como:

$$\forall x \in U: \qquad \mu_{\underset{\sim}{A}\bullet\underset{\sim}{B}}(x) = \mu_{\underset{\sim}{A}}(x) \bullet \mu_{\underset{\sim}{B}}(x)$$

y se representa por: $\underset{\sim}{A} \bullet \underset{\sim}{B}$

también se define la "suma algebraica" de $\underset{\sim}{A}$ y $\underset{\sim}{B}$, como:

$$\forall x \in U: \qquad \mu_{\underset{\sim}{A}+\underset{\sim}{B}}(x) = \mu_{\underset{\sim}{A}}(x) + \mu_{\underset{\sim}{B}}(x)$$

y se representa por: $\underset{\sim}{A} + \underset{\sim}{B}$

Ejemplo 1.1. - Considérense los siguientes conjuntos difusos.

$$\underset{\sim}{A} = \{(x1, 0.2), (x2, 0.7), (x3, 1), (x4, 0), (x5, 0.5)\}$$

$$\underset{\sim}{B} = \{(x1, 0.5), (x2, 0.3), (x3, 1), (x4, 0.1), (x5, 0.5)\}$$

Entonces el producto y suma algebraicos, son respectivamente.

$$\underset{\sim}{A} \bullet \underset{\sim}{B} = \{(x1, 0.10), (x2, 0.21), (x3, 1), (x4, 0), (x5, 0.25)\}$$

$$\underset{\sim}{A} + \underset{\sim}{B} = \{(x1, 0.7), (x2, 1), (x3, 1), (x4, 0 1), (x5, 1)\}$$

1.2.6. Relaciones difusas.

En todas las aplicaciones de las matemáticas juegan un papel importante las nociones de gráfico, correspondencia y relación. A partir de los subconjuntos difusos se puede construir una teoría nueva que permita describir mejor muchos de los fenómenos complejos, los cuales comúnmente se trataban con relaciones de todo o nada

Gráfico difuso.

Sean dos conjuntos ***E1*** y ***E2***, llamemos x a un elemento de ***E1***, y por y a un elemento de ***E2***. El conjunto de pares ordenados (x, y) define al conjunto producto $E1 \times E2$.

El subconjunto difuso $\underset{\sim}{G}$ tal que $\forall (x,y) \in E1 \times E2 : \mu_{\underset{\sim}{G}}(x) \in M$, donde M es el conjunto de membresía de $E_1 \times E_2$ se denomina "grafo difuso".

Sean: $E_1 = (x_1, x_2, x_3, ..., x_i)$

$E_2 = (x_1, x_2, x_3, ..., x_i)$

entonces

$$E_1 \times E_2 = \{(x_1, y_1), (x_1, y_2), ..., (x_1, y_j), (x_2, y_1), (x_2, y_2), ..., (x_2, y_j), ..., (x_i, y_j)\}$$

ahora para simplificar:

$$\mu(x_i, y_j) = \mu_{\underset{\sim}{G}}(x_i, y_j), \qquad i = 1,2,3... \qquad j = 1,2,3...$$

El cual se llamara "valor del par ordenado" $(x_i, x_j) = z_{i,j}$. Que junto con los pares ordenados definen al subconjunto difuso $\underset{\sim}{G} \subset E_1 \times E_2$, o también:

$$\underset{\sim}{G} = \{((x_1, y_1) | z_{1,1}), ((x_1, y_2) | z_{1,2}), ..., ((x_2, y_1) | z_{2,1}), ((x_2, y_2) | z_{2,2}), ..., ((x_2, y_1) | z_{2,1}), ((x_2, y_2) | z_{2,2}), ..., ((x_i, y_j) | z_{i,j})\}$$

que puede ser representado por una matriz, de la siguiente manera:

	y_1	y_2	$\cdots$	y_j
x_1	$z_{1,1}$	$z_{1,2}$	$\cdots$	$z_{1,j}$
x_2	$z_{2,1}$	$z_{2,2}$	$\cdots$	$z_{2,j}$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$
x_i	$z_{i,1}$	$z_{i,2}$	$\cdots$	$z_{i,2}$

Ejemplo 1.2. - Sean

$$\mu(x1,y1)=0.3 \quad \mu(x2,y1)=0.7 \quad \mu(x3,y1)=1$$

$$\mu(x1,y2)=0 \quad \mu(x2,y2)=0.5 \quad \mu(x3,y2)=0.2$$

Que definen al conjunto borroso:

$$\underset{\sim}{G} = \{((x1,y1)|0.3), ((x1,y2)|0), ((x2,y1)|0.7), ((x2,y2)|0.5), ((x3,y1)|1), ((x3,y2)|0.2))\}$$

El cual puede ser representado por una matriz como la siguiente.

	$y1$	$y2$
$x1$	0.3	0
$x2$	0.7	0.5
$x3$	1	0.2

Esto que se acaba de definir para un conjunto producto binario $E_1 \times E_2$ puede generalizarse para un conjunto producto más amplio:

$$E_1 \times E_2 \times E_3 \times \ldots E_n$$

Relación difusa.

La noción de gráfico difuso puede explicarse con la ayuda del concepto de "relación difusa".

Digamos que:

Si P es un conjunto producto de n y M su conjunto de membresía **entonces** una relación n-aria difusa es un subconjunto de P que toma sus valores en M.

Ejemplo 1.3. -

Sean

$$E_1 = \{x1, x2, x3\}$$

$$E_2 = \{y1, y2, y3, y4, y5\}$$

y $$M = [0,1]$$

Entonces la matriz siguiente expresa una relación binaria difusa.

	$y1$	$y2$	$y3$	$y4$	$y5$
$x1$	0	0	0.1	0.3	1
$x2$	0	0.8	0	0	1
$x3$	0.4	0.4	0.5	0	0.2

Una relación difusa en $E_1 \times E_2$ se escribe como:

$$x \in E_1 \ , \ y \in E_2 \qquad x \Re y$$

Máximo y Mínimo.

Se usan:

$\vee_x$ para representar el **máximo** respecto a un elemento o variable x

$\wedge_x$ para representar el **mínimo** respecto a un elemento o variable x

por lo que podemos decir que:

$$\mu(x) = \vee \mu(x, y) = \text{MAX}\mu(x, y)$$

y

$$\mu(x) = \wedge \mu(x, y) = \text{MIN}\mu(x, y)$$

Operaciones con relaciones.

Unión con dos relaciones.- La unión de dos relaciones $\underset{\sim}{\mathfrak{R}}_1$ y $\underset{\sim}{\mathfrak{R}}_2$, representada por $\underset{\sim}{\mathfrak{R}}_1 \cup \underset{\sim}{\mathfrak{R}}_2$ o $\underset{\sim}{\mathfrak{R}}_1 + \underset{\sim}{\mathfrak{R}}_2$ se define como:

$$\mu_{\underset{\sim}{\mathfrak{R}}_1 \cup \underset{\sim}{\mathfrak{R}}_2}(x,y) = \mu_{\underset{\sim}{\mathfrak{R}}_1}(x,y) \vee \mu_{\underset{\sim}{\mathfrak{R}}_2}(x,y) = \text{MAX}\left[\mu_{\underset{\sim}{\mathfrak{R}}_1}(x,y), \mu_{\underset{\sim}{\mathfrak{R}}_2}(x,y)\right]$$

Ejemplo 1.4. – Sean las relaciones $\underset{\sim}{\mathfrak{R}}_1$ y $\underset{\sim}{\mathfrak{R}}_2$ como se muestra:

$\underset{\sim}{\mathfrak{R}}_1$	$y1$	$y2$	$y3$	$y4$
$x1$	0.3	0.2	1	0
$x2$	0.8	1	0	0.2
$x3$	0.5	0	0.4	0

$\underset{\sim}{\mathfrak{R}}_2$	$y1$	$y2$	$y3$	$y4$
$x1$	0.3	0	0.7	0
$x2$	0.1	0.8	1	1
$x3$	0.6	0.9	0.3	0.2

Entonces la operación $\underset{\sim}{\mathfrak{R}}_1 \cup \underset{\sim}{\mathfrak{R}}_2$ da como resultado:

$\underset{\sim}{\mathfrak{R}}_1 \cup \underset{\sim}{\mathfrak{R}}_2$	$y1$	$y2$	$y3$	$y4$
$x1$	0.3	0.2	1	0
$x2$	0.8	1	1	1
$x3$	0.6	0.9	0.4	0.2

Intersección de dos relaciones.- Se define la intersección de dos relaciones $\underset{\sim}{\mathfrak{R}}_1$ y $\underset{\sim}{\mathfrak{R}}_2$, representada como $\underset{\sim}{\mathfrak{R}}_1 \cap \underset{\sim}{\mathfrak{R}}_2$, por la expresión:

$$\mu_{\underset{\sim}{\mathfrak{R}}_1 \cap \underset{\sim}{\mathfrak{R}}_2}(x,y) = \mu_{\underset{\sim}{\mathfrak{R}}_1}(x,y) \wedge \mu_{\underset{\sim}{\mathfrak{R}}_2}(x,y) = \text{MIN}\left[\mu_{\underset{\sim}{\mathfrak{R}}_1}(x,y), \mu_{\underset{\sim}{\mathfrak{R}}_2}(x,y)\right]$$

Ejemplo 1.5. – Sean las relaciones $\underset{\sim}{\mathfrak{R}}_1$ y $\underset{\sim}{\mathfrak{R}}_2$ como se muestra:

$\underset{\sim}{\mathfrak{R}}_1$	$y1$	$y2$	$y3$	$y4$
$x1$	0.3	0.2	1	0
$x2$	0.8	1	0	0.2
$x3$	0.5	0	0.4	0

$\underset{\sim}{\mathfrak{R}}_2$	$y1$	$y2$	$y3$	$y4$
$x1$	0.3	0	0.7	0
$x2$	0.1	0.8	1	1
$x3$	0.6	0.9	0.3	0.2

Entonces la operación $\underset{\sim}{\mathfrak{R}}_1 \cap \underset{\sim}{\mathfrak{R}}_2$ da como resultado:

$\underset{\sim}{\mathfrak{R}}_1 \cap \underset{\sim}{\mathfrak{R}}_2$	$y1$	$y2$	$y3$	$y4$
$x1$	0.3	0	0.7	0
$x2$	0.1	0.8	0	0.2
$x3$	0.6	0	0.3	0

Producto algebraico de dos relaciones.- Se define el producto algebraico de dos relaciones $\underset{\sim}{\mathfrak{R}}_1$ y $\underset{\sim}{\mathfrak{R}}_2$, mediante la expresión:

$$\mu_{\underset{\sim}{\mathfrak{R}}_1 \bullet \underset{\sim}{\mathfrak{R}}_2}(x, y) = \mu_{\underset{\sim}{\mathfrak{R}}_1}(x, y) \bullet \mu_{\underset{\sim}{\mathfrak{R}}_2}(x, y)$$

El símbolo (•) del miembro de la derecha, indica un producto numérico (multiplicación ordinaria).

Ejemplo 1.6. – Sean las relaciones $\underset{\sim}{\mathfrak{R}}_1$ y $\underset{\sim}{\mathfrak{R}}_2$ como se muestra:

$\underset{\sim}{\mathfrak{R}}_1$	$y1$	$y2$	$y3$	$y4$
$x1$	0.3	0.2	1	0
$x2$	0.8	1	0	0.2
$x3$	0.5	0	0.4	0

$\underset{\sim}{\mathfrak{R}}_2$	$y1$	$y2$	$y3$	$y4$
$x1$	0.3	0	0.7	0
$x2$	0.1	0.8	1	1
$x3$	0.6	0.9	0.3	0.2

Entonces la operación $\underset{\sim}{\mathfrak{R}}_1 \bullet \underset{\sim}{\mathfrak{R}}_2$ da como resultado:

$\underset{\sim}{\mathfrak{R}}_1 \bullet \underset{\sim}{\mathfrak{R}}_2$	$y1$	$y2$	$y3$	$y4$
$x1$	0.09	0	0.7	0
$x2$	0.08	0.8	0	0.2
$x3$	0.3	0	0.12	0

Suma algebraica de dos relaciones.- La suma algebraica de dos relaciones $\underset{\sim}{\mathfrak{R}}_1$ y $\underset{\sim}{\mathfrak{R}}_2$, se representa por $\underset{\sim}{\mathfrak{R}}_1 \hat{+} \underset{\sim}{\mathfrak{R}}_2$ mediante la ecuación:

$$\mu_{\underset{\sim}{\mathfrak{R}}_1 \hat{+} \underset{\sim}{\mathfrak{R}}_2}(x,y) = \mu_{\underset{\sim}{\mathfrak{R}}_1}(x,y) \div \mu_{\underset{\sim}{\mathfrak{R}}_2}(x,y) - \mu_{\underset{\sim}{\mathfrak{R}}_1}(x,y) \bullet \mu_{\underset{\sim}{\mathfrak{R}}_2}(x,y)$$

Ejemplo 1.7. – Sean las relaciones $\underset{\sim}{\mathfrak{R}}_1$ y $\underset{\sim}{\mathfrak{R}}_2$ como se muestra:

$\underset{\sim}{\mathfrak{R}}_1$	$y1$	$y2$	$y3$	$y4$
$x1$	0.3	0.2	1	0
$x2$	0.8	1	0	0.2
$x3$	0.5	0	0.4	0

$\underset{\sim}{\mathfrak{R}}_2$	$y1$	$y2$	$y3$	$y4$
$x1$	0.3	0	0.7	0
$x2$	0.1	0.8	1	1
$x3$	0.6	0.9	0.3	0.2

Entonces la operación $\underset{\sim}{\mathfrak{R}}_1 \hat{+} \underset{\sim}{\mathfrak{R}}_2$ da como resultado:

$\underset{\sim}{\mathfrak{R}}_1 \hat{+} \underset{\sim}{\mathfrak{R}}_2$	$y1$	$y2$	$y3$	$y4$
$x1$	0.51	0.2	1	0
$x2$	0.82	1	1	1
$x3$	0.8	0.9	0.58	0.2

Complemento de una relación.- El complemento de $\underset{\sim}{\mathfrak{R}}$, representado por $\overline{\underset{\sim}{\mathfrak{R}}}$, es tal que:

$$\forall (x,y) \in E_1 \times E_2 : \quad \mu_{\overline{\underset{\sim}{\mathfrak{R}}}}(x,y) = 1 - \mu_{\underset{\sim}{\mathfrak{R}}}(x,y)$$

Ejemplo 1.8. – Para la relación $\underset{\sim}{\mathfrak{R}}$ se tiene que $\overline{\underset{\sim}{\mathfrak{R}}}$ es:

$\underset{\sim}{\mathfrak{R}}$	$y1$	$y2$	$y3$	$y4$
$x1$	0.3	0.4	0.2	0
$x2$	0.5	0	1	0.9
$x3$	0.4	0	0.1	0.8

$\overline{\underset{\sim}{\mathfrak{R}}}$	$y1$	$y2$	$y3$	$y4$
$x1$	0.7	0.6	0.8	1
$x2$	0.5	1	0	0.1
$x3$	0.6	1	0.9	0.2

Producto Cartesiano.- El producto Cartesiano entre dos conjuntos difusos $\underset{\sim}{A}$ y $\underset{\sim}{B}$ produce una relación $\underset{\sim}{\mathfrak{R}}$. Debido a que generalmente las relaciones difusas son conjuntos difusos se puede definir el producto Cartesiano como.

$$\underset{\sim}{A} \times \underset{\sim}{B} = \underset{\sim}{\mathfrak{R}} \subset X \times Y$$

con función de membresía.

$$\mu_{\underset{\sim}{R}}(x,y) = \mu_{\underset{\sim}{A}\times\underset{\sim}{B}}(x,y) = \text{MIN}\left[\mu_{\underset{\sim}{A}}(x), \mu_{\underset{\sim}{B}}(y)\right]$$

Ejemplo 1.9. – Supónganse dos conjuntos universo, $\underset{\sim}{A}$ y $\underset{\sim}{B}$

$$\underset{\sim}{A} = \{(x_1 \mid 0.2), (x_2 \mid 0.5), (x_3 \mid 1)\} \text{ y } \underset{\sim}{B} = \{(y_1 \mid 0.3), (y_2 \mid 0.9)\}$$

entonces el producto Cartesiano es,

$$\underset{\sim}{A} \times \underset{\sim}{B} = \underset{\sim}{\mathfrak{R}} =$$

$\underset{\sim}{\mathfrak{R}}$	$y1$	$y2$
$x1$	0.2	0.2
$x2$	0.3	0.5
$x3$	0.3	0.9

1.2.7. Conjunto difuso condicionado.

Un subconjunto difuso condicionado $\underset{\sim}{B}(x) \subset E_2$ se llama "condicionado en E_1" si su función de membresía depende de la relación $x \in E_1$ como parámetro.

La función condicional de membresía se escribirá entonces:

$$\mu_{\underset{\sim}{B}}(x|y), \text{ donde } x \in E_1 \text{ y } y \in E_2$$

esta función define una aplicación de E_1 en el conjunto de los subconjuntos difusos definidos en E_2.

De esta forma, un subconjunto difuso $\underset{\sim}{A} \subset E_1$, inducirá un subconjunto difuso $\underset{\sim}{B} \subset E_2$, cuya función de membresía es:

$$\mu_{\underset{\sim}{B}}(y) = \underset{x \in E_1}{\text{MAX}}(\text{MIN}[\mu_{\underset{\sim}{B}}(x|y), \mu_{\underset{\sim}{A}}(x)])$$

La noción de función para los elementos de los conjuntos formales puede expresarse de la siguiente forma: si $x = a$, entonces $y = b$ por lo tanto la función f se representa por:

$$y = f(x)$$

La noción de conjunto difuso condicionado toma el mismo papel, pero a diferencia de la noción para conjuntos formales en vez de considerar los elementos $x \in E_1$, $x \in E_2$ y la relación f, se define de la siguiente forma:

Sean $\underset{\sim}{X} \in E_1$, $\underset{\sim}{Y} \in E_2$ y la relación difusa $\underset{\sim}{\mathfrak{R}}$ que existe entre E_1 y E_2. Se dirá que : si $\underset{\sim}{X} = \underset{\sim}{A}$, entonces $\underset{\sim}{Y} = \underset{\sim}{B}$ por la relación $\underset{\sim}{\mathfrak{R}}$; y puede representarse por

$$\underset{\sim}{A} \underset{\underset{\sim}{\mathfrak{R}}}{\mapsto} \underset{\sim}{B}$$

Ahora si $\mu_{\underset{\sim}{\mathfrak{R}}}(x, y)$ es la función de membresía de la relación difusa $\underset{\sim}{\mathfrak{R}}$, $\mu_{\underset{\sim}{A}}(x)$ la de $\underset{\sim}{A}$ y $\mu_{\underset{\sim}{B}}(y)$ la de $\underset{\sim}{B}$, se tiene entonces

$$\mu_{\underset{\sim}{B}}(y) = \underset{x \in E_1}{\text{MAX}}\,\text{MIN}\left[\mu_{\underset{\sim}{A}}(x), \mu_{\underset{\sim}{\mathfrak{R}}}(x, y)\right] = \bigvee_x \left[\mu_{\underset{\sim}{A}}(x) \vee \mu_{\underset{\sim}{\mathfrak{R}}}(x, y)\right]$$

Ejemplo 1.10. – Sean:

$$E_1 = (x_1, x_2, x_3)$$

$$\underset{\sim}{A} = \{(x_1 | 0.3), (x_2 | 0.7), (x_3 | 1)\}$$

$$E_2 = (y_1, y_2, y_3, y_4, y_5)$$

$\underset{\sim}{\mathfrak{R}}$

	y_1	y_2	y_3	y_4	y_5
x_1	0.8	1	0	0.3	0.7
x_2	0.8	0.3	0.8	0.4	0.7
x_3	0.2	0.3	0	0.2	1

1.2.8. Proposiciones de Lógica difusa.

Una proposición lógica difusa $\underset{\sim}{P}$, es una sentencia que involucra algunos conceptos sin límites claramente definidos. Son sentencias lingüísticas que expresan ideas subjetivas y las cuales pueden ser interpretadas con poca diferencia por varios individuos. La mayoría del lenguaje natural es difuso, en el cual se involucran vagos e imprecisos términos. Las sentencias describen la altura o peso de una persona, o porcentajes de preferencias de la gente acerca de colores o niveles de temperatura, pueden ser usados como ejemplos de proposiciones difusas. El valor de verdad asignado a $\underset{\sim}{P}$ puede ser cualquier valor comprendido en el intervalo $[0,1]$. El valor de verdad asignado a una proposición es un mapeo del intervalo $[0,1]$ al universo U de valores de verdad T. Es decir,

$$T : U \rightarrow [0,1]$$

Como en lógica binaria clásica, se asigna una proposición lógica a un conjunto en el universo. Es decir, proposiciones difusas son asignadas a conjuntos difusos

Sea la proposición $\underset{\sim}{P}$ asignada al conjunto difuso $\underset{\sim}{A}$ entonces el valor de una proposición, denotado por $T(\underset{\sim}{P})$, esta dado por

$$T(\underset{\sim}{P}) = \mu_{\underset{\sim}{A}}(x) \qquad \text{donde } 0 \leq \mu_{\underset{\sim}{A}} \leq 1$$

El grado de verdad para la proposición $\underset{\sim}{P}: x \in \underset{\sim}{A}$ es igual al grado de membresía de x en $\underset{\sim}{A}$.

Las conectivas lógicas de negación, disyunción, conjunción, e implicación son definidas de igual forma para la lógica difusa.

Para dos preposiciones simples las conectivas lógicas son:

Sean $\underset{\sim}{P}$ definida sobre el conjunto difuso $\underset{\sim}{A}$ y $\underset{\sim}{Q}$ definida sobre el conjunto difuso $\underset{\sim}{B}$.

Negación: $T(\overline{\underset{\sim}{P}}) = 1 - T(\underset{\sim}{P})$

Disyunción: $\underset{\sim}{P} \vee \underset{\sim}{Q} \Rightarrow x \in \underset{\sim}{A}$ o $x \in \underset{\sim}{B}$

$$T(\underset{\sim}{P} \vee \underset{\sim}{Q}) = \text{MAX}(T(\underset{\sim}{P}), T(\underset{\sim}{Q}))$$

Conjunción: $\underset{\sim}{P} \wedge \underset{\sim}{Q} \Rightarrow x \in \underset{\sim}{A}$ y $x \in \underset{\sim}{B}$

$$T(\underset{\sim}{P} \wedge \underset{\sim}{Q}) = \text{MIN}(T(\underset{\sim}{P}), T(\underset{\sim}{Q}))$$

Implicación: $\underset{\sim}{P} \rightarrow \underset{\sim}{Q} \Rightarrow x \in \underset{\sim}{A}$, entonces $x \in \underset{\sim}{B}$

$$T(\underset{\sim}{P} \rightarrow \underset{\sim}{Q}) = T(\overline{\underset{\sim}{P}} \vee \underset{\sim}{Q}) = \text{MAX}(T(\overline{\underset{\sim}{P}}), T(\underset{\sim}{Q}))$$

la implicación puede ser modelada en forma de una regla:

$$\underset{\sim}{P} \rightarrow \underset{\sim}{Q} \qquad \text{Si } x \in \underset{\sim}{A}, \text{ entonces } y \in \underset{\sim}{B}$$

lo que es equivalente a la siguiente relación difusa, $\underset{\sim}{\Re}$:

$$\underset{\sim}{\Re} = (\underset{\sim}{A} \times \underset{\sim}{B}) \cup (\overline{\underset{\sim}{A}} \times Y)$$

cuya función de membresía es expresada por la siguiente fórmula:

$$\mu_{\underset{\sim}{R}}(x, y) = \text{MAX}[(\mu_{\underset{\sim}{A}}(x) \wedge \mu_{\underset{\sim}{B}}(y), (1 - \mu_{\underset{\sim}{A}}(x))]$$

Por último, existe un razonamiento equivalente al predicado lógico de preposiciones ordinarias llamado razonamiento aproximado, el cual es una extensión del cálculo preposicional que trata con verdades parciales.

Para expresar este razonamiento utilizaremos reglas que representan información difusa. Por convención las reglas tienen la siguiente forma:

Si x es ***(antecedente)***, entonces y es ***(consecuente)***

Donde ***antecedente*** y ***consecuente*** representan proposiciones difusas (conjuntos).

Ahora supongamos la siguiente regla :

Regla uno: Si x es $\underset{\sim}{A}$, entonces y es $\underset{\sim}{B}$

También supongamos una segunda regla :

Regla dos: Si x es $\underset{\sim}{A}'$, entonces y es $\underset{\sim}{B}'$

Es posible obtener el consecuente de la segunda regla, $\underset{\sim}{B}'$, a partir de la información de la primera regla con la operación de composición, es decir :

$$\underset{\sim}{B}' = \underset{\sim}{A}' \circ \underset{\sim}{\Re}$$

cuya función de membresía se define como:

$$\mu_{\underset{\sim}{B}'}(y) = \text{MAX}\left[\text{MIN}(\mu_{\underset{\sim}{B}'}(y|x), \mu_{\underset{\sim}{A}'}(x))\right]$$

CAPITULO 2

CONTROL DIFUSO

Si *temperatura* es CALIENTE,
Entonces *velocidad* es RAPIDO

Si *temperatura* es TIBIO,
Entonces *velocidad* es MEDIO

Si *temperatura* es FRIO,
Entonces *velocidad* es LENTO

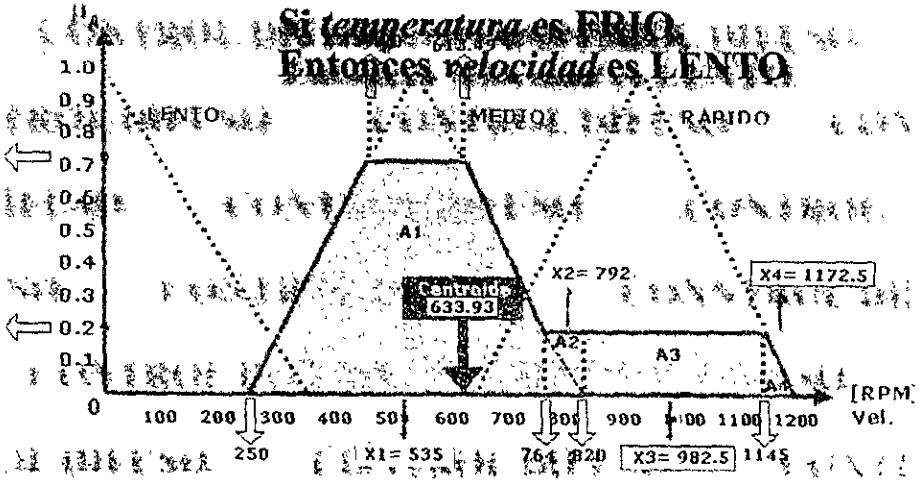


Figura 2. 1 Sistema de control de lazo abierto.

Un sistema de lazo cerrado es diferente de un sistema de lazo cerrado por que su salida es dependiente tanto de la entrada como de la salida. La salida es fabricada en función de la entrada y la salida por medio de realimentación, la cual en muchos sistemas forma una entrada extra al sistema. Ampliando el ejemplo anterior, supongamos que una persona regula la velocidad del automóvil presionando el pedal de la gasolina. La señal de realimentación es la persona, quien regula el flujo de gasolina al motor, basado en el velocímetro. De tal forma que si la persona desea aumentar la velocidad a cierto valor, presiona el pedal del acelerador hasta que el velocímetro alcanza la velocidad deseada, entonces deja de presionar el pedal del acelerador, como se muestra en la figura 2.2.

Figura 2. 2 Sistema de control de lazo cerrado.

2.1.2. Compensación.

Lo primero que se hace para adaptar un sistema, a fin de lograr el objetivo de control, es el ajuste de la ganancia Pero, en muchos casos esta sola operación no proporciona un comportamiento del sistema que alcance el propósito deseado. A veces al aumentar el valor de la ganancia se mejora el comportamiento estacionario, pero provoca una estabilidad débil o incluso inestabilidad. De tal modo que se hace necesario incluir componentes adicionales que cambien el comportamiento de tal forma que el sistema trabaje de la forma deseada. El componente adicional para estos casos se llama "compensador".

Existen dos tipos de compensación, serie y de realimentación (paralelo) El primero coloca un elemento en serie con la función de transferencia como se ve en la figura 2 3(a). El de

realimentación introduce un compensador en un bucle interno formado por una señal proveniente de un elemento, como se muestra en la figura 2.3(b).

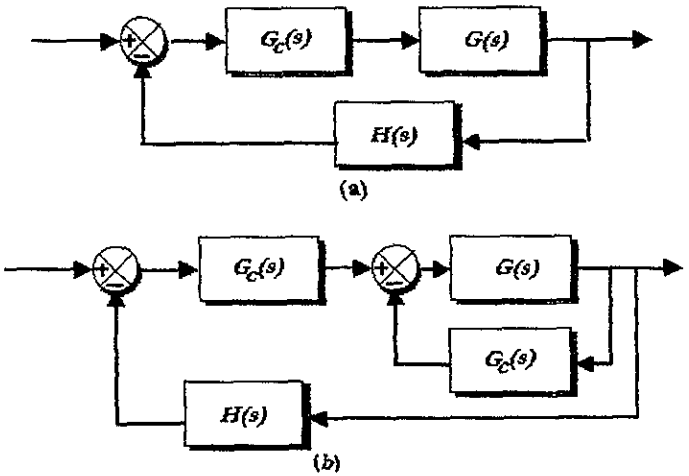


Figura 2. 3 (a) Compensación serie; (b) compensación paralelo.

Actualmente una gran cantidad de sistemas de control utilizan el método proporcional-integral-derivativo (PID). Este método lee la señal de entrada, a la que aplica una ecuación matemática, para producir una salida. Este tipo de compensador PID tiene una ecuación de ganancia de la siguiente forma:

$$G_C(s) = A_p + \frac{A_i}{s} + A_d s = \frac{K(s^2 + bs + c)}{s}$$

La ecuación matemática en la cual se basa este controlador PID es generalmente compleja, además que consume tiempo su construcción y prueba. Comúnmente la salida de este tipo de sistema es usada para controlar un actuador que regula el flujo de algún elemento del sistema. La figura 2.4 muestra la compensación en realimentación.

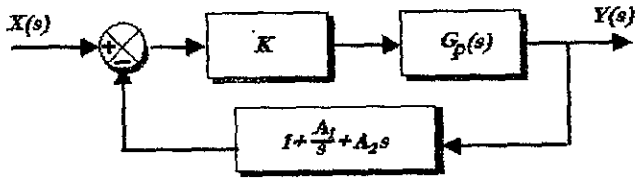


Figura 2. 4 Compensación en realimentación.

2.1.3. Un problema de control.

Para mostrar la diferencia entre lógica difusa y el método tradicional, plantearemos un problema de control. Supongamos que nuestro sistema de control consta de un ventilador eléctrico, de una habitación, que trabaja cuando un sensor de calor proporciona una señal al alcanzar un determinado valor. Deseamos que el ventilador trabaje a 1000 r.p.m. cuando la temperatura de la habitación es mayor o igual a 28 °C, y a 100 r.p.m. cuando la temperatura sea menor o igual a ésta. Este problema de control lo podemos trasladar a dos simples reglas de condición como se muestra a continuación:

Regla 1 Si *temperatura* es ≥28 °C, **entonces** *velocidad* = 1000 r.p.m.

Regla 2 Si *temperatura* es <28 °C, **entonces** *velocidad* = 100 r.p.m.

Un controlador ordinario (no difuso o preciso) depende de un punto de decisión de valor discreto. Para este tipo de sistemas de control, la entrada debe alcanzar un valor exacto antes que el sistema de control reaccione de alguna forma. Sin embargo, pequeñas variaciones en la variable de entrada provocan una reacción drástica y diferente, en la salida.

De las reglas podemos observar que si *temperatura* es de 28 °C o más, la primer regla fija a *velocidad* a 1000 r.p.m. Si *temperatura* esta debajo de 28 °C, la segunda regla fija a *velocidad* a 100 r.p.m. La figura 2.5 muestra un diagrama del valor de control, en términos del control ordinario

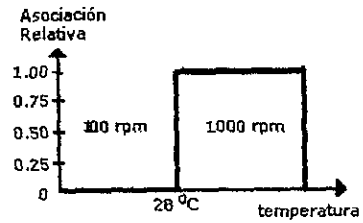


Figura 2. 5 Controlador preciso

Del diagrama podemos ver que no se puede decir nada seguro si la *temperatura* fuera 27.9 °C, es más no sabemos que le pasaría al sistema de control si la temperatura estuviera cambiando de abajo a hacia arriba de 28 °C(o en sentido contrario), por ejemplo de 27.0 °C a 29.0 °C. Es probable que el sistema de control cambie de forma desordenada la velocidad del ventilador, por cambios insignificantes en la variable de entrada. Estos puntos de transición son difíciles de manejar por un sistema de control preciso. Sin embargo en estos casos es precisamente donde la Lógica difusa se distingue.

2.2. La técnica del Control difuso

Una de las aplicaciones más importantes de la teoría difusa son los controladores difusos Estos trabajan un tanto diferente que los controladores convencionales, el reconocimiento experto es usado en lugar de ecuaciones diferenciales para describir un sistema. Este reconocimiento puede ser expresado de una forma muy natural usando variables lingüísticas, las cuales están descritas por conjuntos difusos.

2.2.1. Sistema de control difuso.

Los sistemas de control difuso son sistemas basados en reglas difusas, las cuales representan un mecanismo de decisión de control, para ajustar los efectos de ciertas causas que presente el sistema. El objetivo de los sistemas de control difuso es normalmente sustituir o reemplazar un operador humano experimentado con un sistema basado en reglas difusas. La figura 2.6 muestra un diagrama comparativo en este sentido

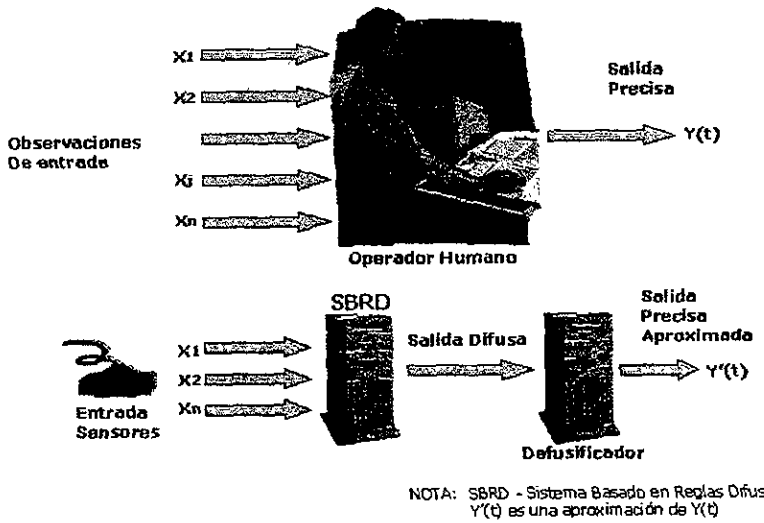


Figura 2. 6 Definición de un Sistema de Control Difuso.

Como se muestra, el operador humano observa una cantidad de datos a partir de un medidor o evaluando una tabla, es decir alguna variable precisa, y después ejecuta una acción definida, como oprimir un botón o girar un volante, obteniéndose una acción o salida precisa. En una forma similar el controlador difuso, usa datos precisos directamente de algún número de sensores; mediante el proceso de ***difusificación*** estos son cambiados a funciones lingüísticas o funciones de membresía. Entonces son transformados por medio de un conjunto de **reglas difusas** " SI – ENTONCES " en una relación de decisión, tal como lo haría un sistema experto, y después obteniéndose alguna salida difusa. La salida difusa es entonces transformada de regreso a valores precisos por medio de un proceso llamado ***desdifusificación***. Este proceso es realizado por medio de algún método de porcentaje promedio, tal como el método del " centroide", dando como resultado un valor preciso. De esta forma se obtiene un valor de salida aproximado al obtenido por el operador humano experto.

Un controlador difuso toma la forma de un conjunto de reglas " SI – ENTONCES ", donde sus antecedentes y consecuentes son así mismo funciones de membresía. Los consecuentes de varias reglas son combinadas por medio de la unión (MAX.), y entonces se obtiene un número real

tomado del centroide, a partir de esta combinación. En el esquema de un sistema experto difuso, tal como en un sistema clásico, las reglas pueden ser el resultado del conocimiento de un operador humano. Supongamos la siguiente regla:

SI la *Temperatura* es *Caliente*, **ENTONCES** disminuye la *Corriente* a *Nivel Medio*.

se observa que "Caliente" y "Nivel Medio" son variables difusas. Tales reglas de lenguaje natural tienen un equivalente en lenguaje de computación, por ejemplo:

SI(A es A1 y B es B1 y C es C1), **ENTONCES** (E es E1 y F es F1)

Es decir, al usar un conjunto de estas reglas se pueden obtener observaciones como si un operador humano estuviera realizando la tarea de control.

2.2.2. Teoría de control de lógica difusa.

Anteriormente se dijo que un sistema de control difuso se compone de tres fases:

1. Difusificación.- Esta sección es responsable de tomar datos de entrada reales (o también llamados datos precisos), y convertirlos en datos que tengan significado en el sistema difuso.
2. Inferencia.- Aquí se genera el conjunto de reglas. Los sistemas de control difusos son por lo general de realimentación, y es en esta fase donde ocurre. El resultado de esta sección son datos de salida difusos.
3. Desdifusificación.- En esta fase los datos de la sección anterior son regresados a datos reales (o precisos).

Un controlador difuso esta echo de variables difusas, reglas, conjuntos, y mecanismos de translación que transforman datos de entrada en datos difusos, y viceversa. La figura 2.7 muestra un típico sistema de control Las variables difusas son similares a las variables normales, excepto que aquellas son usadas para describir cosas que tienen múltiples valores.

Figura 2. 7 Controlador típico de lógica difusa.

Cada una de las fases que intervienen en el diseño difuso se desarrollan en los siguientes apartados.

2.2.3. Difusificación.

En esta primera fase de la lógica difusa, los datos reales de los valores de entrada son mapeados dentro de funciones de membresía difusas. Para describir el proceso completo utilizaremos el mismo ejemplo que se uso para describir el método de control clásico, un sistema de control de clima.

Para crear este sistema de control de clima, primero desarrollaremos funciones de membresía para la variable de entrada ***temperatura***. Estas funciones de membresía son definidas para el rango de valores y para el grado de membresía. En la lógica difusa esto es de suma importancia para distinguir no solamente a cual función de membresía pertenece una variable, sino también el grado relativo que posee. El grado relativo proporciona un "peso"(porcentaje) asociado en la función de membresía dada. Una variable puede tener un peso asociado en varias funciones de membresía al mismo tiempo. En la figura 2.8 se muestran las funciones de membresía para ***temperatura***.

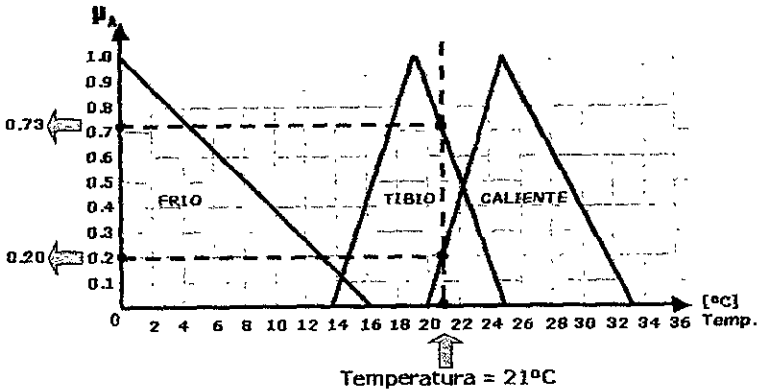


Figura 2. 8 Funciones de membresía para la temperatura.

Para la variable *temperatura* se definen tres conjuntos de membresía *FRIO*, *TIBIO* y *CALIENTE*. Tal y como se muestra en la figura anterior las funciones de membresía tienen un rango de valores, que en cierta parte de su intervalo pueden ser los mismos para dos o más funciones. Cada uno de estos conjuntos comprenden un rango de temperatura, es decir, *FRIO*=[0-16], *TIBIO*=[14-25], *CALIENTE*=[20-30].

Tomemos el valor de 21°C para el problema de control propuesto (ver flecha), trazamos una línea vertical sobre el eje horizontal ***temperatura*** intersecando con las líneas que definen una o más funciones de membresía, en este caso *TIBIO* y *CALIENTE*. A cada punto de intersección (ver flecha) se le asigna un valor correspondiente sobre el eje vertical, para definir su asociación relativa en cada conjunto, asociando de esta manera un valor de entrada preciso con el grado de membresía en determinada función. Nótese que para algún valor en particular de la variable ***temperatura***, este podría pertenecer a uno o más conjuntos difusos. Obsérvese que si el valor de la variable ***temperatura*** es de 21°C es miembro de la función *TIBIO* con una asociación relativa de 0.20. También es miembro de la función *CALIENTE* con una asociación relativa de 0.73.

A diferencia del control clásico, el controlador difuso tomara en cuenta que el valor 21°C de la variable ***temperatura*** es más caliente (0.73) que tibio (0.20), para definir que acción de salida tomara.

Definamos ahora las funciones de membresía para la variable de salida ***velocidad*** como: *LENTO*, *MODERADO* y *RAPIDO*. Sus rangos de velocidad son: *LENTO* =[0-370], *MEDIO* =[250-820] y *RAPIDO* =[650-1200]. Tal y como se muestra en la figura 2.9.

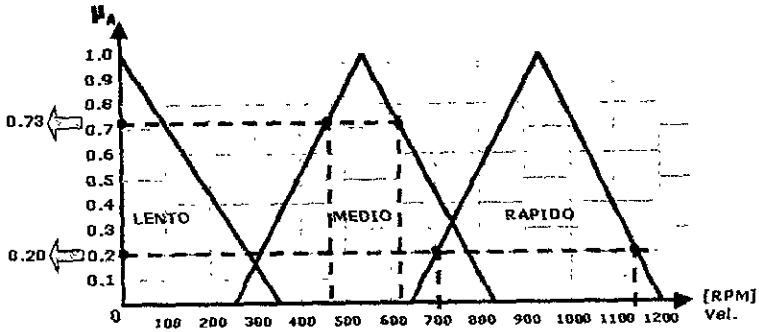


Figura 2. 9 Funciones de membresia para la velocidad.

2.2.4. Reglas de Inferencia

Una vez que las variables y sus conjuntos difusos son definidos se esta listo para describir las reglas encargadas de procesar la entrada. Cualquier número de reglas puede ser creado para definir las acciones de un controlador difuso.

Por ejemplo algunas reglas se muestran a continuación.

SI ***temperatura*** es FRIO, ENTONCES ***velocidad*** es LENTO
SI ***temperatura*** es CALIENTE, ENTONCES ***velocidad*** es RAPIDO
SI ***temperatura*** es TIBIO, ENTONCES ***velocidad*** es MEDIO

Que se puede resumir en una tabla:

temperatura	*velocidad*
FRIO	RAPIDO
TIBIO	LENTO
CALIENTE	MODERADO

Es importante definir suficientes reglas para precisar adecuadamente el medio del sistema sobre su región de operación o plano de control.

Aunque en este caso solo se relaciona una entrada con una salida, se pueden relacionar múltiples variables de entrada/salida. Como las reglas se basan en descripciones en vez de definiciones matemáticas, cualquier relación que puede ser descrita en términos lingüísticos, puede generalmente determinarse para un controlador de lógica difusa. Por lo que, además, sistemas no lineales encuentran una solución a su problema de control con este tipo de controladores. Y como las variables tienen un peso asociado (grado de membresía), las reglas que son compuestas de estas variables lo tienen también

Para un sistema que presenta múltiples entradas y salidas, se puede, con una gran cantidad de reglas definidas, moderar una fluctuación en una entrada en particular, con ayuda del peso asociado. Esto significa que diferentes reglas tienen diferentes impactos sobre el controlador, de acuerdo al valor de la variable de entrada. Esta es la razón por la cual un sistema basado en lógica difusa es en cierto grado robusto y que a menudo permita que algunas reglas sean removidas o alteradas sin que el impacto sea significativo en el controlador.

Ahora que se tienen definidas las reglas el controlador de lógica difusa evalúa las entradas en éstas, generando una salida útil para el control del sistema. Es decir, fijar un voltaje, una corriente, una velocidad, etc. La forma en que se realiza este proceso se describe en el siguiente apartado.

2.2.5. Desdifusificación

Los valores de las variables de salida deberán de ser convertidos en valores precisos que puedan ser realmente usados por el sistema de control. Cada valor de la variable de entrada determina la asociación relativa de esta variable sobre una función de membresía de entrada. Para obtener el mapeo de variables de salida a sus correspondientes funciones de membresía de salida, el peso de asociación de las funciones de entrada y la correspondiente evaluación de las reglas determinaran la asociación relativa en las funciones de salida. En otras palabras cualquier asociación relativa dada a la variable de entrada dará también la variable de salida, señalada por su regla correspondiente

Para observar claramente lo anterior, demos un valor a la variable de entrada, sea ***temperatura*** = 21° C. Para el cual se obtiene una intersección en el conjunto *TIBIO* con una asociación relativa de 0.20. Aplicamos la primera regla, para la cual se obtiene dos intersecciones en el conjunto MEDIO con una asociación relativa de 0.73. Entonces:

temperatura = 21° C – Función de membresía: ***TIBIO***

Grado de Asociación de la Variable de Entrada	0.73
Definición de Regla	SI ***temperatura*** es TIBIO, ENTONCES ***velocidad*** es MEDIO
Función de Membresía de Salida	MEDIO
Grado de Asociación de la Variable de Salida	***velocidad*** (MEDIO) =0.73

La figura 2.10 muestra el resultado producido por la regla:

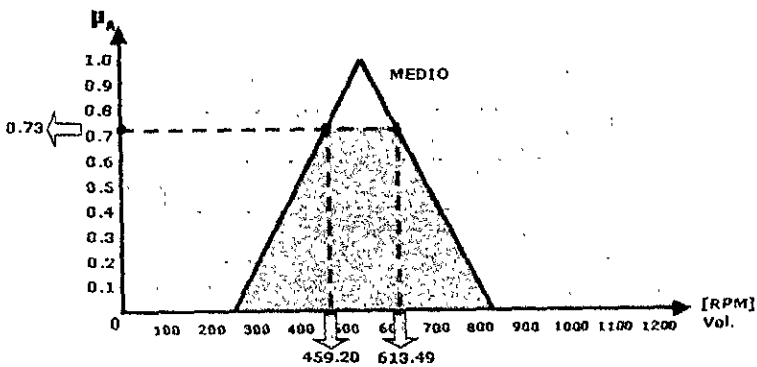


Figura 2. 10 Gráfico producido por la condición temperatura = 21° C – Función de membresía: MEDIO.

Ahora para esta misma ***temperatura*** = 21° C aplicamos la segunda regla, para la cual se obtienen dos intersecciones en el conjunto RAPIDO con una asociación relativa de 0.20. Entonces:

temperatura = 21° C – Función de membresía: *CALIENTE*

Grado de Asociación de la Variable de Entrada	0.20
Definición de Regla	SI *temperatura* es CALIENTE, ENTONCES *velocidad* es RAPIDO
Función de Membresía de Salida	RAPIDO
Grado de Asociación de la Variable de Salida	*velocidad* (RAPIDO) =0.20

Aplicando el valor del grado de asociación obtenido en el grafo correspondiente se tienen dos intersecciones tal y como se muestra en la figura 2.11:

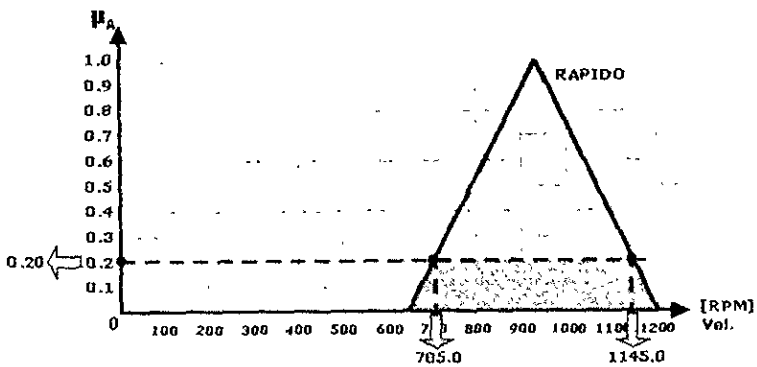


Figura 2. 11 Gráfico producido por la condición temperatura = 21° C – Función de membresía: RAPIDO.

La variable de salida *velocidad* tiene el mismo mapeo relativo como la variable de entrada *temperatura* Tal y como se puede apreciar en las gráficas, las funciones asociadas de entrada son usadas con la misma variable de salida, aplicando dos grados de asociación (peso), es decir:

- Grado de membresía *velocidad* 0 73 asignado a la función de salida MEDIO
- Grado de membresía *velocidad* 0.20 asignado a la función de salida RAPIDO

Obsérvese en las graficas que el valor de salida real se mueve horizontalmente hasta un punto de intersección alcanzando la línea que define el "peso" a la función asociada. Este punto de intersección es entonces trasportado al eje horizontal para determinar el valor de salida "preciso".

Por último a continuación se describirán dos métodos para obtener el valor optimo para la solución del problema.

Método de desdifusificación máximo.

El primer método que se describe es el llamado "método máximo". En este método, si más de una regla se encuentra activa hay que determinar cual sería la el valor de membresía que deberá ocuparse para la obtención del resultado. Siguiendo con nuestro ejemplo, recordamos que hay dos valores posibles de peso para la variable ***velocidad*** 0.73 y 0.20. Este método proporciona un criterio muy sencillo para determinar cual de los dos valores se debe usar, este es, tomar el que tenga el valor más grande, es decir, la siguiente regla que fue escogida por tener el valor más grande (0.73):

SI la ***temperatura*** es TIBIO, ENTONCES fija la ***velocidad*** a nivel MEDIO

En la figura 2.12 se muestra un detalle de la intersección del grado de asociación y de la función de salida afectada.

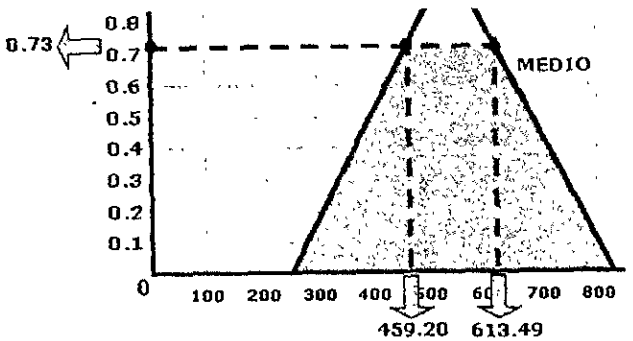


Figura 2. 12. Detalle de la función de salida la cual tiene asignada un mayor grado de membresía.

El valor 0.73 sobre el eje vertical intersecta a la función asociada en dos puntos, uno sobre la inclinación positiva en 459.20 rpm y el otro en 613 49 rpm. Los dos puntos representan dos posibles soluciones que pueden usarse para la solución del problema en cuestión

Método de desdifusificación calculando el centroide.

Otro método para evaluar el valor de salida es el conocido como "calculo del centroide", en este método lo que se determina es un peso promedio de todas las reglas activas, por la suma de todas las aportaciones de las variables de salida, sobre los valores de asociación relativa. Este método puede aprovechar la posibilidad de calcular el valor real de salida a través de algoritmos computacionales, ahorrando una gran parte del trabajo. Además se elimina el problema de soluciones múltiples, presentado en el método máximo, y que pudieran presentarse mientras se obtiene la solución deseada. Este método puede ser aplicado en un procesador que cuente con un hardware avanzado, como muchos de los que hay actualmente en el mercado.

Siguiendo con nuestro ejemplo, calcularemos el centroide de la forma tradicional (cálculo de áreas) con la ayuda de la figura 2.13 que se muestra a continuación.

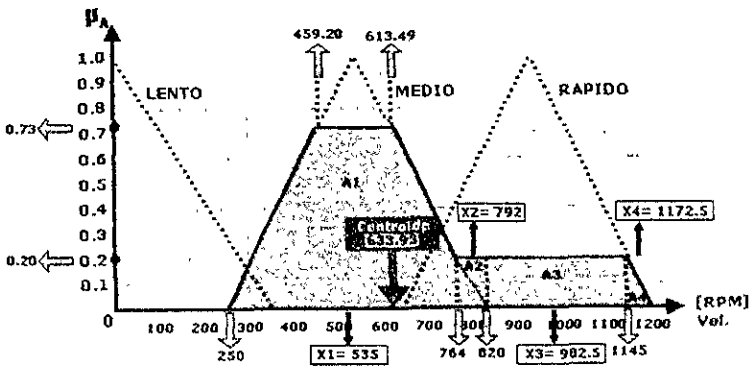


Figura 2. 13 Cálculo del centroide.

Las flechas grises muestran los diferentes puntos sobre el eje horizontal que definen por completo el área que a sido seleccionada a través de los grados de membresía y del conjunto de reglas difusas. Se a dividido el área en cuestión en cuatro para hacer el trabajo más cómodo, éstas son un trapecio A1, dos triángulos A2 y A4 y un cuadrado A3. Las flechas negras señalan el punto

medio requerido sobre el eje horizontal para cada una de las áreas. Entonces tenemos que las áreas y sus correspondientes puntos medios son:

Figura	Area	Xmedia
Trapecio	A1	535
Triangulo	A2	792
Cuadrado	A3	982.5
Triangulo	A4	1172.5

Para obtener el centroide para el eje horizontal se usa la siguiente formula:

$$X_C = \frac{\sum_i^n x_i A_i}{A_{TOT}}$$

donde : x_i = punto medio sobre el eje horizontal de cada una de las áreas.

A_i = área de cada una de las figuras.

A_{TOT} = área total, suma de todas las áreas.

X_C = centroide sobre el eje horizontal.

Al realizar el cálculo se obtiene $X_C = 633.93$ rpm.

Es evidente de acuerdo al ejercicio realizado que este método posee ventaja sobre el método anterior, ya que permite asegurar que se tomarán en cuenta todas las reglas difusas, que se vean afectadas por el grado de membresía relativa. Además que la obtención del resultado a partir del cálculo descrito no implica mayor esfuerzo.

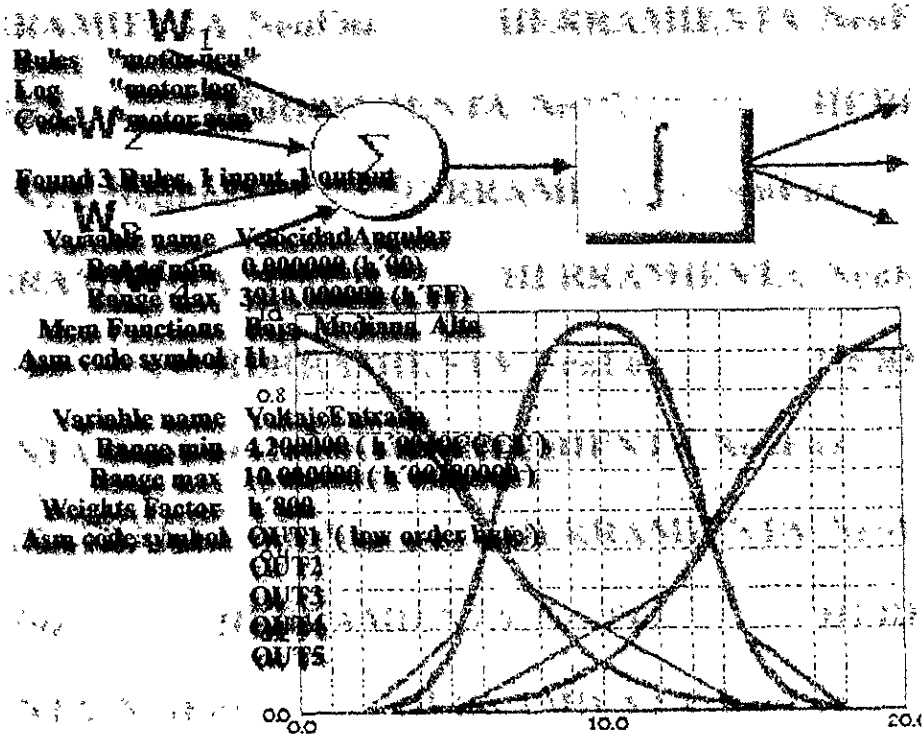
Resumiendo, haciendo uso de la lógica difusa se pueden simplificar problemas complejos de control, resolver otros que no se pueden resolver satisfactoriamente o por completo, o aquellos que su implementación resulta demasiado cara.

El uso de esta técnica aunado al aporte que dan los microprocesadores para realizar el trabajo de cálculo y a los sistemas de programación, permiten desarrollar aplicaciones de gran calidad y funcionamiento Algunos sistemas de programación que pueden ayudar al desarrollo de estas aplicaciones, además de herramientas de trabajo para la implementación, se discuten en el siguiente capítulo

CAPITULO 3

HERRAMIENTA DE DESARROLLO

NeuFuz



CAPÍTULO 3 HERRAMIENTA DE DESARROLLO NeuFuz

3.1. Herramientas de desarrollo.

No hace mucho tiempo empezaron aparecer en el mercado algunos productos ya sea como programas o como equipo, usados en diversas áreas del diseño de proyectos, auxiliando en la implementación y desarrollo de la lógica difusa. Estas herramientas deben ser capaces de mejorar el desempeño y el tiempo utilizado en el diseño en cuestión. Hasta no hace mucho las soluciones basándose en estas herramientas no estaban al alcance de cualquier persona, solo las grandes empresa podían disponer de ellas. Afortunadamente y gracias a la velocidad en la que se mueve la electrónica, estas herramientas han podio ver disminuido su costo (en algunos casos no) y se pueden conseguir desde demos, kit de aprendizaje, hasta versiones profesionales.

Algunas de estas herramientas se listan a continuación con una breve descripción:

Fuzzy C.	Togai InfraLogic Irvine, C	Lenguaje difuso TIL

Este programa permite escribir, probar, depurar, y emplear un sistema experto. No permite realizar una simulación completa por el mismo. Le permite al usuario escribir el conjunto de reglas con un lenguaje de alto nivel llamado *Lenguaje de Programación Difuso TIL*; generando códigos fuentes ANSI o Kernigham and Richie C para ser utilizadas en la implementación de sistemas expertos.

La estructura básica de un archivo escrito en este lenguaje es la siguiente:

- Identificación de las entradas/salidas
- Tipos de variables de datos
- Definición de las funciones de membresía
- Definición de las reglas difusas
- Conexión de las reglas difusas y las entradas/salidas

FULDEK	Bell Helicopter Textron, Inc. Forth Worth, TX	Ambiente completo de simulación para Windows

Este programa es una aplicación para el sistema Windows, esta escrito en lenguaje Microsoft Visual Basic. FULDEK tiene dos opciones principales, el EDITOR OPTION y el RUN OPTION. La opción EDITOR permite manejar las variables presentes en el problema de control, editar las funciones de membresía, creación y manejo de reglas difusas. La opción RUN permite crear y manejar un grafico, permite conectar el conjunto de reglas difusas a un modelo linear dinámico y después realizar la simulación.

FLCG	Univ. New México Albuquerque, NM	Generador de código lógico

Esta aplicación fue escrita para ser usada en una Macintosh Este programa puede generar un código en lenguaje C para la implementación de aplicaciones específicas de lógica difusa El código generado puede ser ligado a un modelo para simular procesos de control de lógica difusa. Este código también crea un archivo por default donde coloca resultados intermedios del programa de control, valores de membresía de entradas y salidas, las reglas usadas y la intervención que tienen éstas. El paquete FLCG consiste de dos módulos: un programa generador(FLCG), y una librería de funciones (FLO).

Fuzzy Microcontroller	Neural Logix Sanford, FL	Usa una tarjeta para el control en tiempo real

Conocido también como el NLX-230, viene con una tarjeta y con programación escrita en lenguaje C. En el programa las entradas son clasificadas de acuerdo a como el usuario haya determinado el conjunto de funciones de membresía El controlador usa una función de membresía simétrica y un sencillo esquema de deducción difuso para implementaciones digitales simples Las reglas son usadas para determinar que conjunto de condiciones esta presente en la entrada El NLX-230permite almacenar hasta 64 reglas en el chip de memoria de 24 bit.

FIDE	Aptronix, Inc, Palo Alto, CA	Programa, simulación y generador de código

Uno de los programas más recientes de simulación de lógica difusa es llamado FIDE que le permite al usuario realizar las siguientes tareas:

- Composición de funciones básicas difusas, inferencia, composición de funciones de membresía, creación y evaluación del conjunto de reglas.
- Simulación de control difuso.
- Depuración de programas.
- Generación de código para aplicaciones en tiempo real.

El lenguaje básico usado en FIDE se llama FIL(Fuzzy Inference Lenguage por sus siglas en ingles) el cual describe partes de un sistema de inferencia difuso, como cualquier otro lenguaje cuenta con su propia sintaxis y semántica. Los métodos de desdifusificación usados en FIDE son los del centroide, el máxumo, Maximización izquierda y el de Maximización derecha.

NeuFuz	National Semiconductor Corporation Santa Clara, CA	Programa hecho para trabajar con sus microprocesadores COP.

Es una sofisticada herramienta usada para diseñar lógica difusa, con el objeto de ser utilizada en microcontroladores de National Semiconductor NeuFuz resuelve el problema de diseño que se presenta al usar la lógica difusa en varias formas·

- Primero y más importante, crea las complejas reglas de lógica difusa y las funciones de membresía automáticamente.
- Segundo, una red neuronal es usada para aprender las relaciones de entrada-salida, las cuales son entonces convertidas en un modelo que funciona de manera equivalente a la lógica difusa.
- Por último, y no menos importante, completa el proceso convirtiendo el modelo de lógica difusa en un módulo de código máquina difuso, listo para ser implementado en la familia de microcontroladores de National Semiconductor.

La descripción completa de esta herramienta se desarrolla en las páginas siguientes, ya que es la que se va a utilizar para el diseño difuso del presente trabajo.

3.2. Bases de NeuFuz

NeuFuz es una herramienta de desarrollo de programación para diseñar lógica difusa con el objeto de ser utilizada en microcontroladores. La determinación de un conjunto correcto de reglas y de funciones de membresía para un sistema complejo es una tarea difícil, aún para un experto diseñador de sistemas de lógica difusa. NeuFuz genera automáticamente las reglas y las funciones asociadas usando su habilidad de aprendizaje.

3.2.1. Lógica difusa en NeuFuz.

Tanto las redes neuronales como la lógica difusa son dos técnicas diferentes para resolver problemas que no han tenido una solución suficientemente aceptable al usar los medios más convencionales.

Como ya se ha dicho la lógica difusa es una técnica que trata con relaciones imprecisas de entrada/salida. La lógica difusa obtiene la información de entrada, y basándose en deducciones que no están descritas exactamente toma decisiones basadas en aproximaciones.

La utilidad de una solución de lógica difusa viene del hecho de que se describen las relaciones de entrada/salida usando un lenguaje parecido al humano. Todo lo que se necesita es entender como trabaja el sistema, y la habilidad para poder describir en términos formales de lógica difusa estas relaciones.

La lógica difusa ***"no es imprecisa"*** a pesar que su término "difusa" hiciera pensar que así es. El termino "difusa" se refiere a la técnica de solución usada, donde los rangos de valor de las relaciones son descritos por términos aproximados al lenguaje humano. La lógica difusa es más simple que la lógica convencional y más barata, y evita complejas descripciones matemáticas del sistema, es decir la lógica difusa no usa un modelo matemático Otra posibilidad con la lógica difusa es poder expresar incertidumbre, imprecisiones y ruido.

Algunos términos clave de lógica difusa usados en NeuFuz son los siguientes:

Conjuntos difusos y funciones asociadas.

- En lógica difusa cada rango de entrada lingüística (frío, normal, caliente, etc.) es un conjunto difuso.
- El conjunto difuso usa funciones de membresía.
- Las funciones de membresía definen el grado de asociación de elemento al conjunto.
- Las funciones asociadas definen el grado de asociación en el rango [0%-100%]. De esta manera, un elemento puede ser miembro, no ser miembro o ser parcialmente miembro.

Reglas de lógica difusa.

- De la misma forma que los valores de entrada son definidos lingüísticamente, las reglas que relacionan la entrada con la salida son definidas de la misma forma. Por ejemplo considérese un sistema de control de temperatura con ventilador, de una entrada una salida.

  SI temperatura es caliente, ENTONCES velocidad ventilador es alta

  SI temperatura es normal, ENTONCES velocidad ventilador es media

  SI temperatura es frío, ENTONCES velocidad ventilador es baja
- Cada regla asocia el conjunto difuso de entrada con el conjunto difuso de salida.
- Las expresiones que aparecen primero que ENTONCES son llamadas el ***antecedente*** y las expresiones mostradas después son llamadas ***consecuentes***.
- Si el sistema tuviera dos entradas y una salida, la regla difusa incluirá dos antecedentes y un consecuente:

  SI entrada1 es bajo Y entrada2 es medio, ENTONCES la salida es largo

Pasos de diseño con lógica difusa.

- Difusificación: Como en el mundo real trata con números, definiendo y analizando la entrada y salida en términos lingüísticos (p.e. temperatura = frío, tibio o caliente)

- Contribución de reglas: En este paso se calcula la contribución de cada uno de los conjuntos de valores. El grado de la función de membresía se encuentra al encontrar la intersección de los valores de entrada con las funciones de membresía de entrada.

  El proceso para calcular el grado de membresía es el siguiente:

  1. Seleccionar un punto sobre el eje horizontal y proyectar una línea paralela al eje vertical y que cruce sobre los diferentes conjuntos de membresía.
  2. Calcular el grado de asociación para todos los conjuntos difusos involucrados con el punto anterior (p.e. Función de membresía frío = .08)
  3. La intersección de la función de membresía de salida con el mínimo grado de asociación de todas las entradas es utilizada para determinar el nivel final de la salida.

- Evaluación de reglas: Cuando múltiples reglas son seleccionadas, se necesita integrar en forma conveniente la contribución de todas ellas para determinar la salida final difusa.
- Desdifusificar: El valor difuso de la salida final necesita regresarse a un valor numérico ya que la aplicación necesita de un número preciso Para convertir la salida difusa final se emplea un algoritmo, el método más conocido es del centroide, de acuerdo a este método el centro del área es calculado mediante la formula siguiente:

$$Salida = \frac{\sum_{n} U(x) \times \mu(x)}{\sum_{n} \mu(x)}$$

donde:

U = Universo

μ = grado de membresía de la función

Número de funciones de membresía.

- En general, entre más funciones de membresía son usadas más robusta y exacta es la solución, sin embargo, a mayor cantidad de funciones de membresía mayor cantidad de reglas de memoria se necesitan.

Número de reglas.

- Para un sistema difuso de una sola entrada el número de reglas es igual al número funciones de membresía. Para sistemas con dos o más entradas, el número de reglas es igual al producto del número de funciones de membresía por cada entrada, como se muestra en la siguiente formula:

$$R = M_1 \bullet M_2 \bullet M_3 \bullet \ldots M_n$$

donde:

R es el número de reglas.

n es el número de entradas.

M es el número de funciones de membresía para la entrada.

Nótese que dependiendo el número de funciones de entrada, se incrementa el número de reglas significativamente.

Estabilidad del sistema.

- Distinto de los sistemas basados en modelos matemáticos convencionales, en una solución difusa no hay una verdadera prueba para probar su estabilidad. En términos de ingeniería generalmente se traduce como el probar la estabilidad del sistema, es decir probar que no haya un conjunto de entradas que pudieran causar fluctuaciones indeseadas. Por lo tanto, la estabilidad del sistema difuso es probada a través de extensivos ensayos y simulaciones.

3.2.2. Red Neuronal en NeuFuz.

Una red neuronal es una estructura artificial usada para imitar al cerebro humano. Usa varios elementos de conexión simple interconectados en alternancia con elementos de conexión compleja. Una red neuronal resuelve problemas de una forma similar en la que lo hace el cerebro humano. Puede aprender y generalizar habilidades, el aprendizaje es llevado a cabo por variaciones apropiadas de las conexiones complejas permitiendo aprender una conducta entrada/salida de un sistema. Una vez que la red ha sido entrenada, puede ser usada para desarrollar soluciones para controlar un sistema. Y si es necesario puede continuamente obtener experiencia adicional, y considerarla para el cambio de condiciones

Las redes neuronales son buenas en la creación generalizada de soluciones, una red neuronal podría ser más que suficiente, es decir no se requeriría usar una solución de lógica difusa. Pero en realidad es sencillamente impráctico, debido a los requerimientos de memoria y de capacidad de computo, los cuales deben ser muy grandes, y que hacen imposible que una solución de red neuronal sea implementada en un microcontrolador.

La lógica difusa puede ser usada para crear una equivalente red neuronal que expresa virtualmente las mismas relaciones de entrada/salida como si se tratase de una red neuronal. La gran diferencia es que mientras la red neuronal puede aprender continuamente, el modelo de lógica difusa representa un instante "congelado" de una red neuronal(aunque alguna adaptación difusa puede dar posibilidad de aprendizaje limitado). Además como el proceso de aprendizaje continuo no es lo normal en el diseño con microcontroladores, el modelo difuso resulta más adecuado para la solución de estas aplicaciones. El modelo de lógica difusa provee una solución que gracias a algunas técnicas de aproximación adicionales puede ser fácilmente codificado e implantado en un microcontrolador. El resultado final a pesar de ser una aproximación, es una aproximación suficientemente buena para aplicaciones en el mundo real.

Algunos términos de redes neuronales usados en NeuFuz son los siguientes:

Conjunto de entrenamiento.

- La entrada de una red neuronal es llamada conjunto de entrenamiento. Este conjunto es una serie de patrones; cada patrón define el valor(o valores) de entrada y el valor (o valores) de salida correspondientes.

Arquitectura.

- Una red neuronal consiste de varios elementos de procesamiento simples, cada elemento es llamado "neurona". Véase la figura 3.1

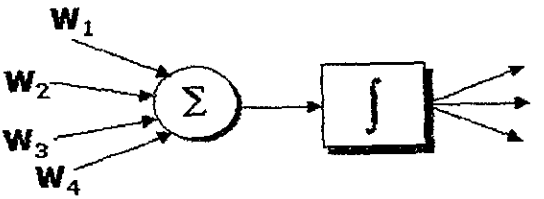


Figura 3. 1 Neurona.

Una neurona toma su entrada a partir de varias otras neuronas y da su salida a otras neuronas. Cada entrada a la neurona es asociada con un peso representado por Wn. Cada señal de entrada termina multiplicada por el peso correspondiente. La neurona suma todos los productos peso/entrada. Un umbral de operación se realiza en esta suma. La salida final de la neurona depende de su umbral de operación.

- Generalmente una red neuronal esta formada de capas de neuronas, lo que da mejor capacidad para aprender y tratar con datos no lineales. La figura 3.2 muestra un diagrama de las capas de una red.

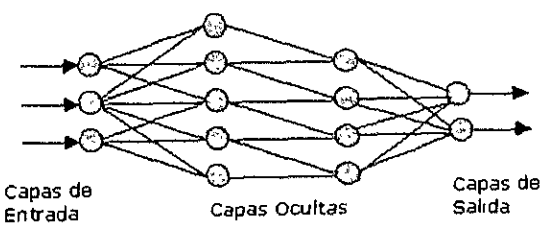


Figura 3. 2 Red neuronal y sus capas.

Cada neurona interna tiene asociado un peso, durante el aprendizaje la red neuronal ajusta todos sus pesos hasta que para un valor arbitrario de entrada proporcione un valor de salida correspondiente.

Aprendizaje.

- El proceso en el cual la red neuronal ajusta sus conexiones es llamado aprendizaje, la forma en que aprende es leyendo secuencialmente todos los patrones en el conjunto de entrenamiento.

Para cada patrón, la red ajusta sus pesos así que los valores de entrada producen, tan preciso como sea posible, el valor de salida deseado. El conjunto de entrenamiento es leído muchas veces, cada secuencia, a través del conjunto completo de entrenamiento es llamado "ciclo".

Convergencia.

- Cuando todas las entradas producen una correspondiente salida con un margen de error predefinido, el peso interno a sido ajustado apropiadamente, entonces la red neuronal alcanza la convergencia.
- El margen de error definido es llamado épsilon. Este parámetro le dice al algoritmo iterativo de la red cuando detenerse, esto ocurre cuando se alcanza una solución suficientemente aceptable.

Monitoreando el progreso de aprendizaje.

- El programa de aprendizaje monitorea sobre un ciclo por base. Conforme el número de ciclos se va efectuando, el progreso de aprendizaje es indicado por un consistente decremento en la cantidad de error de los valores de salida calculados.
- Una medida común del error es el porcentaje de error, calculado del margen de error(épsilon):

$$\frac{Err}{Eps} = \frac{\text{Valor de salida deseado} - \text{Valor de salida calculado}}{\text{Epsilon}}$$

  El porcentaje Err/Eps es usado para monitorear el proceso de aprendizaje, durante un ciclo de aprendizaje el Err/Eps es calculado para cada patrón. Mientras un patrón muestre solamente 1, el patrón se considera como no aprendido.
- Mientras el número de ciclos incrementa, el valor absoluto más grande de Err/Eps (llamado también Err/Eps Máx.) decrece. Este valor representa una función del número de ciclos y sirve para planear los caminos de convergencia de la red neuronal.

Caminos de convergencia de una red neuronal.

- Existen dos caminos básicos de convergencia del valor Err/Eps Máx. En el primero, el valor lentamente se va aproximando a 1, llamado convergencia directa como se ve en la figura 3.3.

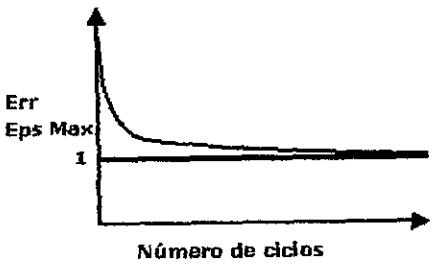


Figura 3. 3 Convergencia Directa.

En el segundo camino de convergencia, el valor inicia de Err/Eps Máx. decrece lentamente pero este empieza a oscilar. Véase la figura 3.4.

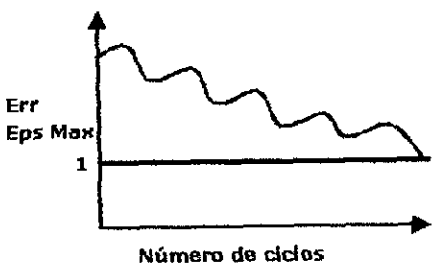


Figura 3. 4 Convergencia Oscilante.

Las oscilaciones ocurren cuando la diferencia entre los pesos es muy grande y esto se presenta hasta el momento en que se alcanza un valor de épsilon antes especificado.

Generalmente el aprendizaje de una red neuronal es controlado por varios parámetros de entrenamiento y al ajustar estos valores las oscilaciones pueden ser eliminadas o disminuidas.

- Aunque el camino de convergencia directa es posible cuando una red neuronal esta aprendiendo de un sistema no lineal, puede presentar algún grado de oscilación En el peor de los casos la red podría oscilar por siempre buscando convergencia. Ver la figura 3.5.

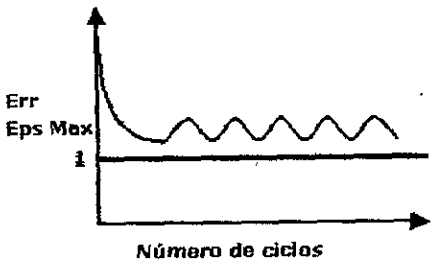


Figura 3. 5 Sin Convergencia.

Es importante observar que la determinación de funciones asociadas y reglas difusas es una tarea difícil para problemas del mundo real, por lo que el uso de una red neuronal provee de una aproximación bastante buena.

Como se vera en el siguiente apartado NeuFuz no requiere de un gran entendimiento sobre redes neuronales, lo más importante a saber es:

1. La convergencia sigue varios caminos.
2. Existen varios caminos para monitorear la convergencia.
3. Los parámetros de aprendizaje de la red neuronal son usados para provocar convergencia.

3.3. Funcionamiento de NeuFuz.

En el modelo de lógica difusa las relaciones de entrada/salida son definidas por funciones asociadas y reglas. Cada entrada se representa por lo menos con dos funciones de membresía. El propósito de la función de membresía es el de expresar a las entradas numéricas como variables difusas usando términos lingüísticos; tales como alto, medio y bajo. La relación existente entre las variables se expresa usando reglas, que en conjunto con las funciones asociadas pueden ser fácilmente codificadas en algoritmos que son eficientes en tamaño y velocidad.

NeuFuz analiza los datos de entrada usando una red neuronal, a partir de un conjunto de "patrones de entrenamiento" que la red "aprende". Una vez aprendido, la red neuronal puede

generar los valores de salida correctos para cualquier valor de entrada. El punto central de NeuFuz es la conversión de la red neuronal en el modelo equivalente de lógica difusa, realizando el trabajo más duro y difícil de la fase de diseño. Para asegurarse que el trabajo a sido realizado correctamente el modelo es probado y verificado, si el modelo parece adecuado para la aplicación se generan las reglas y las funciones asociadas.

3.3.1. Esquema de trabajo.

Para iniciar un proyecto en NeuFuz lo primero será determinar que datos se tomarán y como se organizarán estos. Dependiendo el proyecto los datos variarán pero generalmente serán obtenidos de una o más de las siguientes formas:

- Medición
- Experiencia
- Simulación
- Modelos matemáticos

Una vez que se a concluido con la obtención de los datos el procedimiento a seguir consta de cuatro pasos:

1. Creación del conjunto de datos de entrenamiento.
2. Entrenamiento de la red neuronal de NeuFuz.
3. Verificar y editar las funciones de membresía.
4. Generar e integrar el código máquina difuso.

Un diagrama de flujo muestra el proceso en mejor detalle, vea la figura 3.6. Las secciones siguientes tratan de los pasos involucrados en el desarrollo de la solución.

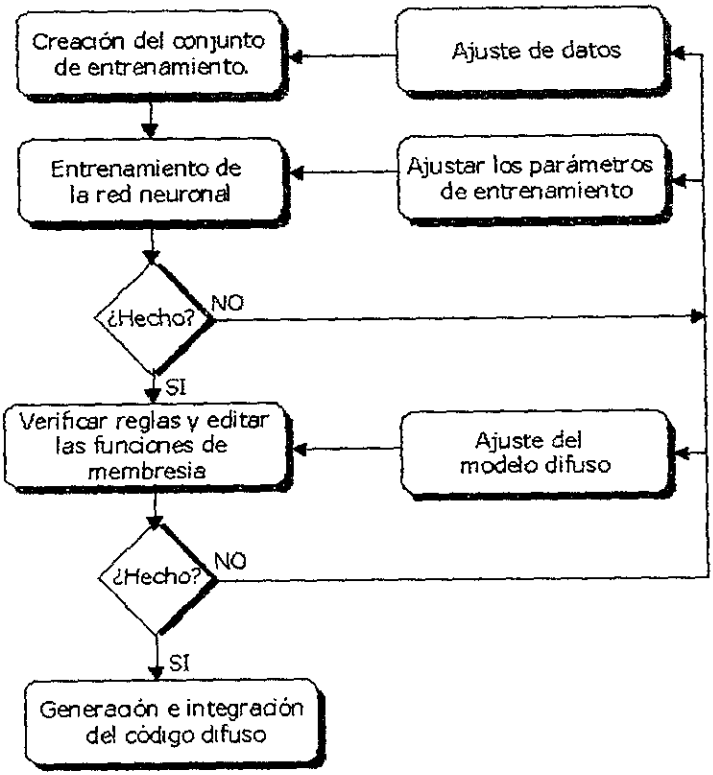


Figura 3. 6 Diagrama de flujo del proceso que sigue NeuFuz.

Lo primero que hay que hacer es crear un archivo que contenga los datos de entrenamiento. El conjunto de datos corresponde a los propios nodos de la aplicación. Consideremos el siguiente ejemplo, se tiene un sistema de control que cuenta con un controlador puesto en serie con la aplicación o planta, el sistema es realimentado, ver figura 3.7.

El conjunto de entrenamiento incluye como entradas I_1(error) e I_2(cambio de error = error presente – error previo), O es la salida. La relación entre I_1, I_2 y O pueden ser determinadas por medición, simulación o modelo matemático.

Una vez que NeuFuz a aprendido el conjunto de datos de entrenamiento, la solución es evaluada y si es adecuada, el código ensamblador es generado. En éste punto el controlador mostrado en la figura 3.7 es reemplazado por un procesador que corra dicho código.

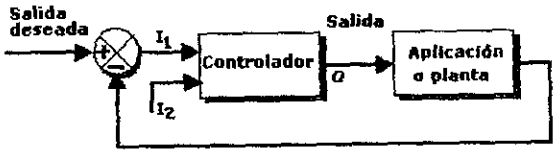


Figura 3. 7 Sistema de control.

3.3.2. Optimización de las funciones de membresía y las reglas difusas.

Funciones de membresía optimizadas

La figura 3.8 muestra una forma triangular usada en una función de membresía, también se pueden presentar otras formas típicas que incluyen rectángulos y trapezoides. El problema con todas estas representaciones es que al ser formas fijas limitan la flexibilidad al ser utilizadas.

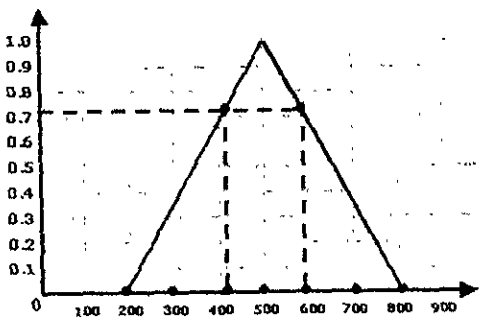


Figura 3. 8 Función de Membresía para un valor difuso 500.

NeuFuz funciona de forma diferente, implementa funciones de membresía como si fueran curvas, con formas de campana, tal y como se muestra en la figura 3.9 para la función de membresía *Tibio*. La forma suave de estas funciones de membresía supera los cambios de las transiciones características del mundo real de sistemas no lineales, lo que permite el uso más efectivo de las funciones de membresía y produce soluciones que requieren pocas reglas.

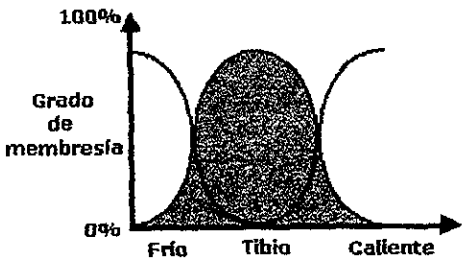


Figura 3. 9 Funciones de membresía con forma de campana.

Para que el sistema en cuestión sea robusto, obsérvese que las funciones de membresía se traslapan unas con otras. NeuFuz tiene la capacidad para aproximar la función de membresía con las formas comúnmente utilizadas en lógica difusa tradicional, con el propósito de hacer lo más pequeño que sea posible el código final.

Reglas difusas optimizadas

Una solución difusa con NeuFuz generalmente incluye *reglas marginales*, que presentan un nivel de contribución relativamente muy bajo con respecto de otras reglas. Estas reglas marginales son indeseables ya que a pesar de contribuir muy poco aumentan el tamaño del código final. En la figura 3.10 se muestra un fragmento de un archivo de reglas difusas creado por NeuFuz.

```
#Regla 1
Si entrada 1 es 1.7 y entrada 2 es 1.7 entonces salida es 2.2541

#Regla 2
Si entrada 1 es 1.7 y entrada 2 es 1.6 entonces salida es 4.2933

#Regla 3
Si entrada 1 es 1.7 y entrada 2 es 1.5 entonces salida es 0.0005
```

Figura 3. 10 Fragmento de un archivo de reglas.

La contribución de cada regla es determinada por la magnitud relativa del valor de salida. Si todas las reglas de este ejemplo presentan un valor de salida más grande que 2, entonces la regla 3 es considerada como marginal; como se ve este valor de salida es relativamente pequeño; Una ves que se desdifusifica el valor de contribución de esta regla es mínimo. Por lo tanto la regla 3 puede ser eliminada sin afectar significativamente el valor de la salida.

Factor de borrado

NeuFuz cuenta con una función para remover reglas marginales, esto se hace fijando un parámetro llamado *factor de borrado*. Los efectos de diferentes valores del factor de borrado pueden ser fácilmente evaluados y de esta manera obtener una solución optima.

Parámetros de aprendizaje

La red neuronal de NeuFuz es una red multicapas, internamente las capas y pesos representan las funciones asociadas y reglas de lógica difusa. Existen dos parámetros de aprendizaje que tienen que ver con la convergencia:

1. El porcentaje de aprendizaje.- Este porcentaje determina la cantidad de cambio de los pesos en la red neuronal, en cada paso de aprendizaje.
2. El factor de aprendizaje.- Determina los ajustes de los pesos de las capas interiores de las neuronas. Pequeños cambios en el factor de aprendizaje tienen un gran efecto en el proceso de aprendizaje que cambios en el porcentaje de aprendizaje.

3.3.3. Determinación de valores de entrada y parámetros de la fase de entrenamiento.

En NeuFuz un proyecto en específico es identificado por un nombre que puede tener hasta 8 caracteres. Todos los nombres de los archivos relacionados con un proyecto en particular utilizan como prefijo común el nombre del proyecto. Para distinguir los diferentes tipos de archivo se asignan a cada tipo una extensión única tal y como se muestra a continuación.

- " * . inp " archivo de datos de entrada
- " * . rec " archivo de memoria
- " * . neu " archivo de reglas de lógica difusa
- " * . wgt " archivo de parámetros y pesos de la red neuronal
- " * . asm " archivo de código ensamblador difuso

Guías para el conjunto de entrenamiento

NeuFuz aprende a partir de las relaciones de entrada/salida del conjunto de datos de entrenamiento. El conjunto de entrenamiento es determinante en el aprendizaje de las

características de la aplicación y de la calidad de la solución deseada. Además del conjunto de entrenamiento también se deben de crear los datos de memoria, que son usados para la evaluación y prueba de la red neuronal después que ha aprendido el comportamiento del sistema a través del conjunto de entrenamiento. A continuación se definen algunas guías útiles para la creación del conjunto de datos de entrenamiento.

Guía para datos de entrada

El archivo de datos de entrada debe ser descrito de la forma más sencilla como sea posible evitando discontinuidades. Se debe tener la plena seguridad que los datos son correctos ya que NeuFuz no puede determinar si se ha descrito convenientemente al sistema. Al momento del desarrollo del archivo de los datos de entrada se debe de tomar en cuenta las siguientes consideraciones:

- **Determinación de límites.-** El archivo de datos de entrada define los valores tanto del límite de entrada como el de salida. Hay que asegurar el incluir muestras antes y después de los límites normales de operación. La definición de los límites es una característica muy importante de los datos de entrada.

  Después que la red neuronal de NeuFuz ha aprendido el comportamiento de la aplicación, al dar valores de entrada de la red, fuera de los límites definidos por el archivo de datos de entrada, dará como resultado valores que se fundamentan en la capacidad de generalización de la red neuronal.

  Para asegurar valores de salida aceptables, hay que verificar que el archivo de datos de entrada incluye muestras contenidas en los extremos de operación del sistema.

- **Número de muestras.-**En general, a mayor número de muestras mejores resultados se obtendrán en el aprendizaje. Hay que asegurarse de incluir muestras a través de todo el rango de los límites tales como las posibles condiciones de error, ya que de esta forma se obtendrá una solución mucho más robusta.

  El número de patrones depende en cierto grado de las características propias de los datos. La figura 3.11 muestra los valores de entrada y salida de una función no lineal; las marcas "X" ilustran una muestra incluida en el archivo de entrada.

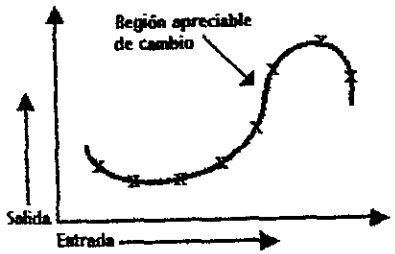


Figura 3. 11 Distribución de muestras.

El conjunto de muestras no esta del todo mal, pero la red neuronal puede encontrar dificultad al generalizar la solución. Esto se debe a que en algunos puntos el cambio se manifiesta de una forma considerable, también debido a que en los extremos máximo / mínimo de los rangos de entrada y salida algunas muestras no están incluidas. La figura 3.12 muestra la misma función de entrada/salida con más muestras.

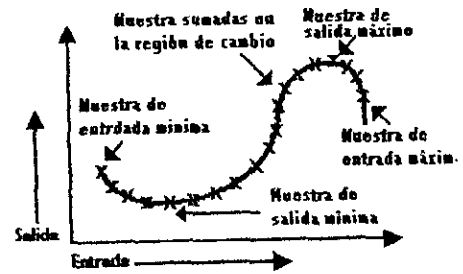


Figura 3. 12 Una mejor distribución de muestras.

Nótese que al incluir más muestras se reduce la diferencia entre muestras adyacentes, y en la región de cambio los datos son mejor representados, además de que los límites también son mejor presentados.

- **Delta entre muestras de salida.-** La diferencia entre los valores de las muestras consecutivas de salida deberá ser tan consistente como sea posible. Las pequeñas diferencias y variaciones menores no son un problema, pero diferencias de un orden de magnitud mayor deben ser evitadas.

- **Orden de muestras.-** El orden de las muestras puede ser muy importante para ciertas aplicaciones. Al organizar las muestras, clasificando por valores de salida, se muestran positivos efectos en la convergencia.

Guías para datos de memoria

Para pruebas de gran amplitud sobre la capacidad de generalización de la red neuronal, los datos de memoria deben contener diferentes muestras a partir de datos del archivo de entrada. Si pruebas completas muestran malos resultados, hay que sumar más patrones al archivo de datos de entrada y volver a entrenar la red.

La primera guía que hay que observar en la creación del archivo de datos de memoria es la siguiente:

- No incluir datos de entrada fuera de los límites definidos por el archivo de datos de entrada.

Archivo de datos de entrada

El formato del archivo de entrada es importante ya que los datos de éste contienen las muestras de entrenamiento. Cada muestra puede contener hasta cuatro valores de entrada y un valor deseado de salida.

- **Nombre del archivo de datos de entrada.-** Un archivo de datos de entrada es un archivo con formato ASCII que puede ser creado a partir de un simple editor de textos. Este archivo debe tener el mismo nombre que se uso para el modelo y su extensión debe ser ".inp".
- **Creando el archivo de datos de entrada.-** La creación del archivo de datos de entrad es como sigue:

    1. Crear el archivo de datos de entrada con un formato ASCII, sin ningún formato adicional

2. Anotar los valores para los vectores de entrada tal y como se muestra a continuación en la figura 3.13:

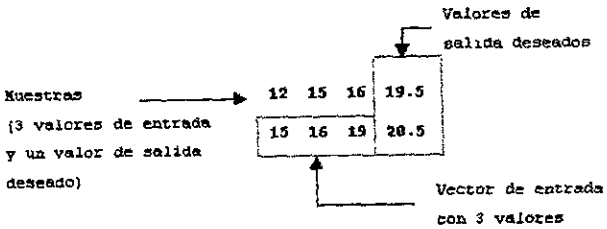


Figura 3. 13 Archivo de datos de entrada.

Estos valores deben de estar colocados en cuatro columnas por cada fila. Se puede utilizar números enteros(positivos o negativos) o números de punto flotante.

La última columna a la derecha contiene la variable correspondiente al valor de salida deseado. La fila completa es llamada "*muestra*" y cada muestra consiste de 1 a 4 valores de entrada y 1 valor de salida deseado, en nuestro ejemplo solo hay 3 valores de entrada y 1 valor de salida. El archivo de datos de entrada puede contener hasta 1200 muestras (30 es el mínimo recomendado). Líneas vacías en el archivo son ignoradas.

3. Se pueden adicionar comentarios en cada línea usando el símbolo "#". NeuFuz ignora todos los caracteres delante de este símbolo y hasta el final de la línea.
4. Se debe salvar el archivo con el mismo nombre del modelo pero con la extensión ".inp".

Cuando NeuFuz usa los datos de este archivo comprueba que no ocurran los siguientes errores:

- Que no haya muy pocos valores en una muestra(de menos debe haber dos, un valor de entrada y un valor de salida).
- Que no haya mas de 1200 muestras.

Archivo de datos de memoria

El archivo de datos de memoria provee valores de entrada para probar los resultados del entrenamiento de la red neuronal de NeuFuz.

- **Nombre del archivo de datos de memoria.-** Se debe usar una extensión ".rec" para este archivo. Obsérvese que no es necesario usar el nombre del modelo ya que se puede contar con múltiples archivos de memoria cada uno con un nombre diferente.
- **Creando y dando formato al archivo de memoria.-** Para crear el archivo de datos de memoria hay que seguir los siguientes pasos:
    1. Con un editor de textos se crea el archivo asegurándose de no utilizar ningún formato especial, solamente en formato ASCII.
    2. Escriba los valores para los vectores de entrada en las primeras cuatro columnas de la fila. Se pueden usar enteros(positivos o negativos) o números de punto flotante. Los valores de los vectores del archivo de memoria deben estar dentro del rango de los valores de los datos del archivo de entrada.
       Cada línea consiste de cuatro valores. El archivo puede contener cualquier número de líneas de vectores. Las líneas vacías son ignoradas.
       La figura 3.14 muestra un ejemplo de un archivo de datos de memoria.

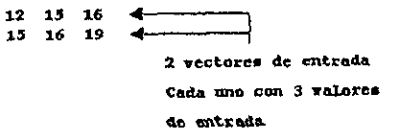


Figura 3. 14 Archivo de datos de memoria.

    3. Se pueden agregar opcionalmente comentarios, para ello hay que preceder el comentario con el símbolo "#" por ejemplo:

       12 15 16 # Primera línea

    4. Guardar el archivo usando un nombre con una extensión ".rec"

NOTA: Se debe asegurar que los archivos "*.inp" y "*.rec" estén localizados en el mismo directorio.

Configuración de parámetros

Seis parámetros de configuración describen la estructura y el medio de aprendizaje del modelo de la red neuronal de NeuFuz.

I. Localización del directorio para el archivo de datos de entrada del modelo.
II. Especificar el nombre del modelo.
III. Especificar Epsilon.
IV. Especificar el porcentaje de aprendizaje.
V. Especificar el factor de aprendizaje.
VI. Especificar el número de funciones de membresía.

La figura 3.15 muestra la ventana de entrenamiento de la red neuronal.

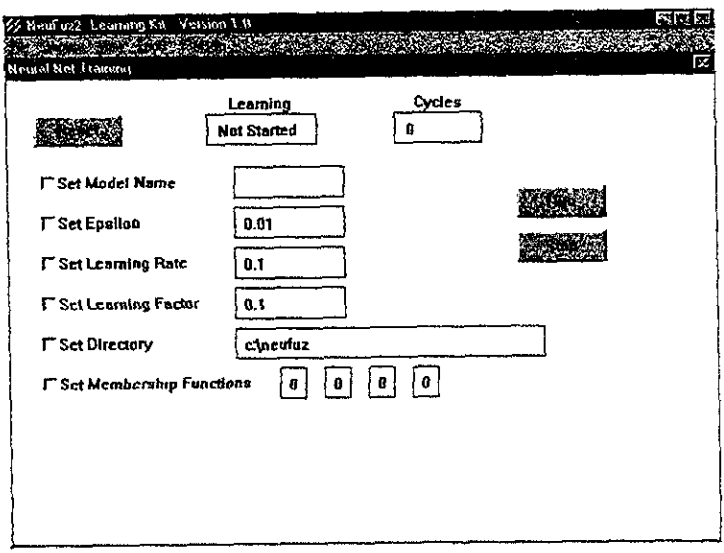


Figura 3. 15 Ventana de entrenamiento de la red neuronal.

Para cargar un modelo, dos de los parámetros de configuración deben ser establecidos en el siguiente orden:

I. Localización del directorio para el archivo de datos de entrada del modelo.

El directorio especificado delante de la caja de chequeo Set Directory donde el archivo de datos de entrada y el archivo de datos de memoria del modelo en cuestión se deben localizar.

Para el directorio de trabajo se debe hacer lo siguiente:

- Seleccionar la caja de chequeo a la izquierda de Set Directory y una ventana de dialogo aparece.
- Escribir el nombre del directorio en el cual se localicen los archivos del modelo. Seleccionar Set para guardar los cambios.

II. Especificar el nombre del modelo.

- Seleccionar la caja de chequeo a la izquierda de Set Model Name
- Escribir el nombre del modelo y para salvar los cambios seleccionar Set.

NOTA: Cualquier valor previamente fijado para este modelo en la ventana de entrenamiento para el Epsilon, Porcentaje de entrenamiento, Factor de entrenamiento, y funciones de membresía son vueltas a cargar y mostrados automáticamente.

III. Especificar Epsilon.

Epsilon es la diferencia máxima aceptable entre el valor de salida computado por la red neuronal y el valor deseado en una muestra. Después de converger, la salida de error de la red neuronal no debe exceder el valor de Epsilon. Se debe especificar un valor inicial de Epsilon apropiado para el modelo.

- Seleccione la caja de chequeo a la izquierda de Set Epsilon.
- Proporcione el valor para Epsilon y seleccione Set.

IV. Especificar el porcentaje de aprendizaje.

El porcentaje de aprendizaje en el cual NeuFuz aprende de los datos suministrados. Este parámetro determina la velocidad y precisión de la convergencia.

Para el porcentaje de aprendizaje se debe hacer lo siguiente:

- Seleccionar la caja de chequeo a la izquierda de Set Learning Rate y una ventana de dialogo aparece.
- Escribir el porcentaje de aprendizaje y entonces seleccionar Set para guardar los cambios.

V. Especificar el factor de aprendizaje.

El factor de aprendizaje determina el porcentaje de aprendizaje relativo entre las diferentes capas internas de la red neuronal. El factor de aprendizaje tiene un alto efecto no lineal sobre la convergencia. Pequeños cambios en este valor pueden tener un gran efecto sobre la convergencia, mucho más que pequeños cambios hechos en el porcentaje de aprendizaje. El cambiar el factor de aprendizaje solo debe hacerse si primero se ha tratado cambiar el porcentaje de aprendizaje. Para garantizar la convergencia de la red, se debe iniciar con un valor pequeño del factor de aprendizaje, tal como 0.01.

Para el factor de aprendizaje se debe hacer lo siguiente:

- Seleccionar la caja de chequeo a la izquierda de Set Learning Factor y una ventana de dialogo aparece.
- Escribir el factor de aprendizaje y entonces seleccionar Set para guardar los cambios.

VI Especificar el número de funciones de membresía.

La primera vez que un modelo es cargado, NeuFuz asigna un valor por defecto para el número de funciones de membresía para cada entrada. Si este valor es adecuado no se debe fijar el valor de este parámetro Si se esta especificando un modelo existente que la red neuronal de NeuFuz ya ha aprendido, este paso debe omitirse.

Para el número de funciones de membresia se debe hacer lo siguiente:

- Seleccionar la caja de chequeo a la izquierda de Set Membership Functions y una ventana de dialogo aparece.
- Escribir el número de funciones de membresía para cada variable de entrada (los únicos valores validos son 2 a 7 y 0) Se debe empezar por la izquierda y los lugares no ocupados deben ser 0
- Entonces seleccionar Ok para guardar los cambios

Asignación de etiquetas

Para facilitar la lectura del archivo de las reglas difusas generadas se pueden asignar *etiquetas* a las variables de entrada, las funciones de membresía y a la salida. Las etiquetas son usadas en lugar de los nombres que por defecto son dados por NeuFuz.

Para fijar las etiquetas hay que hacer lo siguiente:

1) Activar la ventana de entrenamiento(Ver figura 3.15).
2) Especificar el nombre del modelo.
3) Seleccionar File del menú y después Set Label. Aparece una ventana como la mostrada en la figura 3.16.

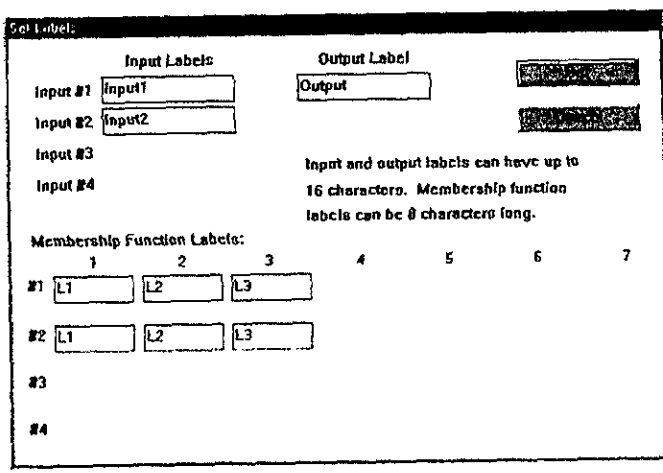


Figura 3. 16 Ventana de especificación de etiquetas.

4) El valor por defecto de las variables de entrada son Input1, Input2, Input3 e Input4 Se selecciona el texto y se cambia por el deseado.

5) Las etiquetas por defecto para las funciones de membresía asociadas con cada variable son L1, L2, L3,..., L7. Las líneas son numeradas de la siguiente forma:

La línea #1 es la función asociada con la 1ra.variable de entrada

La línea #2 es la función asociada con la 2da.variable de entrada

La línea #3 es la función asociada con la 3ra.variable de entrada

La línea #4 es la función asociada con la 4ta.variable de entrada

Cuando se ha terminado de la asignación se selecciona Set para guardar los cambios.

3.3.4. Entrenamiento de la red neuronal.

El entrenamiento de la red neuronal es el paso clave para obtener una solución optima de lógica difusa, para la aplicación en cuestión. Una vez que NeuFuz ha aprendido los datos del archivo de entrada, solo entonces, se cuenta con una solución para ser evaluada.

Si la primera solución satisface los requerimientos de la aplicación, entonces la tarea de diseño esta completa. Sin embargo, generalmente se deberán evaluar varias soluciones con diferentes costos, desempeño y consideraciones de precisión.

También, se debe considerar que en el primer intento de aprendizaje del conjunto de entrenamiento se puede fallar, la red neuronal podria oscilar, y los parámetros de entrenamiento deben ser ajustados para forzar a la convergencia.

Selección de los valores iniciales para los parámetros de entrenamiento

El entrenamiento de la red involucra la preparación de valores iniciales de los siguientes parámetros de entrenamiento:

- **Epsilon.**

Como una guía general selecciónese un valor de epsilon que refleje los requerimientos de la aplicación. Si la aplicación tolera una desviación de 20% en los valores de salida, escójase el correspondiente valor de epsilon. El uso de un pequeño epsilon no significa necesariamente alcanzar una mejor convergencia, además de que puede resultar en oscilaciones, lo que

incrementa el tiempo requerido para alcanzar la convergencia o incluso en ausencia de convergencia.

No se debe seleccionar un epsilon más grande que el requerido por la aplicación.

- **Número de funciones de membresía.**

A mayor número de funciones de membresía que se especifiquen mayor será el grado de robustez y precisión que tendrá la solución. También en algunos casos, con más funciones de membresía se obtiene una convergencia muy rápida. Sin embargo más funciones de membresía significan más reglas con el resultado en el incremento de requerimientos de memoria y costos adicionales.

Hay dos puntos a considerar para la selección del número de funciones de membresía:

1) Selección para precisión y robustez, o
2) Selección para desempeño y costo.

La primera selección incrementa el número de funciones de membresía y el costo de la solución. La segunda selección usa el mínimo número de funciones de membresía, lo que produce resultados aceptables en el desempeño y uso de memoria.

Aunque debe tomarse en cuenta que el número de funciones de membresía por entrada pueden variar, algunas pueden seleccionarse para aumentar la robustez, otras por razones de consto y desempeño.

- **Porcentaje de aprendizaje.**

En general, el porcentaje de aprendizaje es inversamente proporcional a la diferencia entre valores del conjunto de muestras del conjunto de entrenamiento.

Si la diferencia, en valores consecutivos de salida, de las muestras del conjunto de entrenamiento es grande, tales como 10 a 100, úsese un valor pequeño para el porcentaje de aprendizaje, por ejemplo 0 005 ó 0.0005. Para pequeñas diferencias, úsese valores grandes, por ejemplo 0.5 ó 0.1.

Por las dudas inícicse con un porcentaje grande, y si la red oscila entonces redúzcase el valor

En general, redúzcase el porcentaje de aprendizaje si la red oscila(una forma de hacerlo sería el reducir el porcentaje por un factor de 10) De otra manera, si la red esta convergiendo lentamente, trátese incrementando este porcentaje.

- **Factor de aprendizaje.**

En general, el factor de aprendizaje es inversamente proporcional a la diferencia en los valores de salida del conjunto de muestra de entrenamiento.

Como una regla general, si la diferencia en los valores de salida del conjunto de muestras de entrenamiento es grande, como 10 a 100, úsese un valor pequeño para el factor de entrenamiento tal como 0.005 ó 0.0005. Para pequeñas diferencia, tómese un valor grande para el factor tal como 0.5 ó .01.

Para ajustar el factor de aprendizaje hay que seguir los pasos siguientes:

1) Reducir el factor de aprendizaje si el ajustar el porcentaje de aprendizaje falla en reducir la oscilación.
2) Reducir el factor de aprendizaje si después que la red converge las funciones de membresía muestra un cambio dramático en su forma. Cambios dramáticos o distorsión en las formas pueden resultar incluso en valores nulos o negativos del grado de las funciones de membresía. Los posibles valores negativos son consecuencia del manejo matemático y no tienen ningún significado físico

Entrenamiento de la red

Es necesario que durante el entrenamiento de la red se monitorice la convergencia, debido a que generalmente la primera vez se presentan oscilaciones, entonces se debe detener el proceso para realizar los ajustes necesarios.

- Inicio del entrenamiento

Para iniciar el entrenamiento hay que hacer lo siguiente.

1. Iniciar la ventana de entrenamiento.
2. Dar el directorio, nombre del modelo y los valores de Epsilon, Porcentaje de aprendizaje, Factor de aprendizaje y Funciones de membresía
3. Seleccionar el botón Run para iniciar el proceso, tal y como se muestra en la figura 3.17

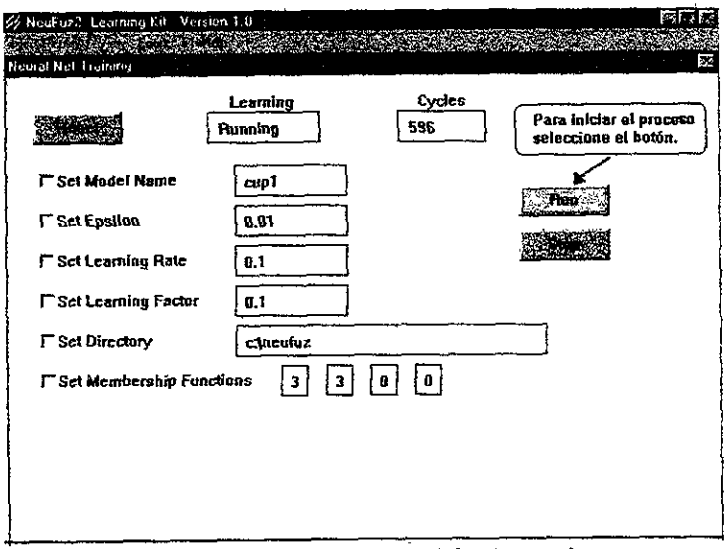


Figura 3. 17 Ventana de entrenamiento de la red neuronal.

El cuadro debajo de Learning cambia a Running y el cuadro debajo de Cycles se incrementa.

Un ciclo es un paso sobre todas las muestras de entrenamiento. El tiempo para ejecutar un ciclo dependerá del número de muestras de entrenamiento, el número de funciones de membresía por cada variable de entrada y el número de muestras por cada error que rebase el valor de Epsilon.

- Monitoreo para la convergencia

Durante cada ciclo en el proceso de aprendizaje, un número de muestras tendrá errores. Una muestra se considera con error si su valor de salida calculado difiere del valor deseado de salida por un valor más grande que el especificado para el Epsilon. El camino más sencillo para monitorear el avance de la convergencia es el observar el error máximo NeuFuz muestra el error de la muestra como el cociente del error actual sobre el Epsilon, éste es llamado el cociente Err/Eps. El primer Err/Eps es llamado el valor Err/Eps Máx.

NeuFuz cuenta con una ventana donde se muestran los errores generados durante el proceso de entrenamiento, para mostrar la ventana hay que seleccionar Errors del menú de la ventana de entrenamiento. La figura 3.18 muestra la ventana de error, donde los errores son mostrados por tamaño.

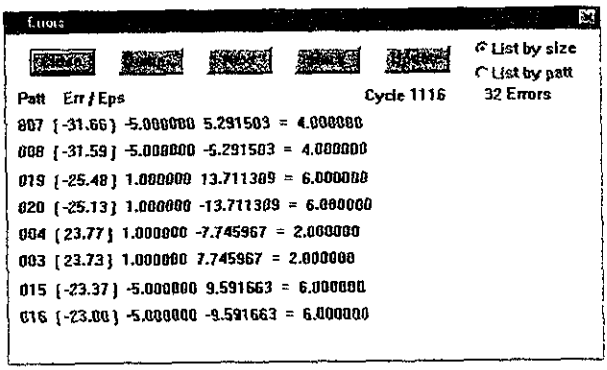


Figura 3. 18 Ventana de error.

Cada línea muestra lo siguiente:

- La posición de la muestra en el archivo ".inp"
- El valor Err/Eps (encerrado en corchetes cuadrados)
- La muestra de entrada, la cual consiste de hasta cuatro valores de entrada y un valor deseado de salida.

Inicialmente el valor Err/Eps decrementa rápidamente, después que el valor inicial decrece quizá acercándose a 1, pero esto no es muy probable. Es más probable que el valor Err/Eps Máx comience a oscilar. Sin embargo, mientras el camino sea hacia 1, la red convergerá
Durante la oscilación, el camino hacia 1 puede ser imperceptible (o posiblemente no existir). En este caso el valor Err/Eps parece oscilar entre dos extremos fijos

¿Cuánto tiempo esperar para ajustar los parámetros de entrenamiento? No se sabe a ciencia cierta. El número de ciclos para la convergencia no puede predecirse, muestras extraidas del mundo real pueden tomar mucho tiempo.

Hay que tener en cuenta que NeuFuz cuenta con características de ayuda para que rápidamente se traten diferentes parámetros para la convergencia. Se puede hacer uso de la función Copy para copiar el modelo y de la función Dump de la ventana de error para coleccionar datos de diferentes sesiones de entrenamiento.

- Problemas con la sesión de entrenamiento.

Existen tres problemas básicos que se pueden encontrar durante la sesión de entrenamiento:

1. La red oscila y no se ve ningún progreso hacia la convergencia después de muchos ciclos.

   En el caso de las oscilaciones normalmente se necesita reducir el porcentaje de aprendizaje y/o el factor de aprendizaje. Primero se debe tratar de ajustar el porcentaje de aprendizaje, si esto no minimiza las oscilaciones se debe entonces ajustar el factor de aprendizaje. Hay que tener en cuenta que deberá restablecerse la red neuronal para que tomen efecto los cambios al porcentaje y factor de aprendizaje

   Si el ajuste de los parámetros de aprendizaje no mejora el camino de la convergencia, entonces se debe tratar incrementando el Epsilon. Si la red converge ahora pero el Epsilon es muy grande, para la aplicación en cuestión, se necesitará aumentar más muestras al conjunto de entrenamiento.
2. La red es cerrada para la convergencia, pero al final del proceso algunas muestras permanecen por mucho tiempo en la ventana de error.

   Se presentan algunos problemas con las muestras cuando algunas de ellas (alrededor del 10%) toman muchos ciclos (alrededor del 50% de tiempo) durante la convergencia. Nótese que si a pesar de este fenómeno la solución resulta adecuada, entonces no se debe tomar ninguna acción.

   Sin embargo, si se desea mejorar esta situación entonces se debe tratar sumando muestras al conjunto de entrenamiento.
3. Una solución es evaluada, pero su exactitud es inadecuada a pesar que el Epsilon es razonable.

   La exactitud de una solución es medida por la evaluación de los resultados de una prueba realizada al archivo de datos de memoria. Si se conocen los valores de salida

deseados para los vectores del archivo de entrenamiento, se pueden comparar estos contra los valores de salida generados por NeuFuz.

Si la red neuronal de NeuFuz predice valores de salida para vectores de entrada no aprendidos(vectores de entrada no incluidos en el conjunto de entrenamiento) que resultan no aceptables, entonces primero se debe tratar sumando más muestras al archivo de datos de entrada contrarrestando el echo de una Epsilon pequeña. Sí el problema persiste, se debe tratar haciendo el Epsilon un bit más pequeño, y volver a entrenar la red neuronal.

3.3.5. Evaluación y prueba de la solución.

Tan pronto como la red neuronal de NeuFuz ha aprendido el comportamiento de la aplicación se cuenta con una solución lista para ser evaluada Se evalúa con ayuda de las funciones de prueba de NeuFuz y si pasa las pruebas, entonces se esta listo para generar e integrar el código de lógica difusa.

Proceso de evaluación

El proceso de evaluación consiste en examinar las relaciones de entrada/salida para corregir, adecuar y optimizar el número de reglas en la solución difusa. NeuFuz cuenta con dos características básicas para asistir este proceso:

1. Un modo grafico interactivo, donde el cursor puede fácilmente manipular los valores de entrada y el resultado de salida es mostrado inmediatamente
2. Un modo automático de entrada donde NeuFuz lee el archivo de memoria y crea un archivo de salida para evaluación.

Estas dos características pueden ser accedidas en la ventana "Rules Verification"

- Pasos básicos para la evaluación

  Los siguientes son los pasos básicos de la evaluación:

1. Para procesadores como el COP8, se edita la aproximación lineal de las funciones de membresía.
2. Evaluar los valores de salida del sistema difuso para un adecuado funcionamiento de la aplicación.
3. Optimizar el número de reglas lógicas difusas. Esto involucra configurar un valor del factor de borrado. Basado en este valor NeuFuz elimina las reglas marginales.
4. Repetir el paso 2, depurar el paso 3 si es necesario.

Hay que observar que el proceso de evaluación es iterativo, y que para procesadores como el COP8 pruebas reales con el código difuso generado, puede revelar inadaptabilidad introducida por la aproximación lineal de las funciones de membresía. Si esto ocurre hay que regresar al paso 1 y depurar la aproximación. Pero si la depuración falla para un rendimiento aceptable, entonces, hay que reducir el valor de Epsilon ligeramente y volver a entrenar la red.

- Valores de salida

NeuFuz presenta tres tipos de salidas para la solución:

1. Salida de la red neuronal
2. Salida del sistema difuso
3. Salida del sistema difuso usando la aproximación lineal de las funciones de membresía.

Estas tres salidas se muestran tanto en la ventana "Rules Verification" como en el archivo de salida del archivo de memoria de prueba.

- Factor de borrado

El factor de borrado se configura en la ventana "Rules Verification". NeuFuz usa este valor para eliminar reglas marginales. El factor de borrado puede fijarse a un valor que elimine, tanto como le sea posible, las reglas marginales mientras que el sistema difuso produzca resultados aceptables. Menor número de reglas significa mejor desempeño y menor requerimiento de memoria. Una regla es considerada marginal si su contribución a la salida es relativamente pequeña comparada con las otras reglas. Obsérvese que es posible que algunas soluciones no presenten reglas marginales, es decir, que todas las reglas contribuyen significativamente y/o equitativamente a la salida.

Edición de las funciones de membresía

La ventana "Function Editing" es usada para editar las funciones de membresía. NeuFuz genera trapezoides que son aproximaciones lineales de las curvas en forma de campana de las funciones de membresía. Se deben editar las aproximaciones trapezoidales para minimizar el error, tanto como lo requiera la aplicación.

Para editar la aproximación trapezoidal, primero hay que activar la ventana "Neural Net Training" y entonces hacer lo siguiente:

1. Seleccionar del menú "Display" y entonces Editing1 para mostrar las curvas con forma de campana que representan todas las funciones de membresía asociadas con la primera variable de entrada.
   La figura 3.19 muestra las curvas con forma de campana y la aproximación lineal.

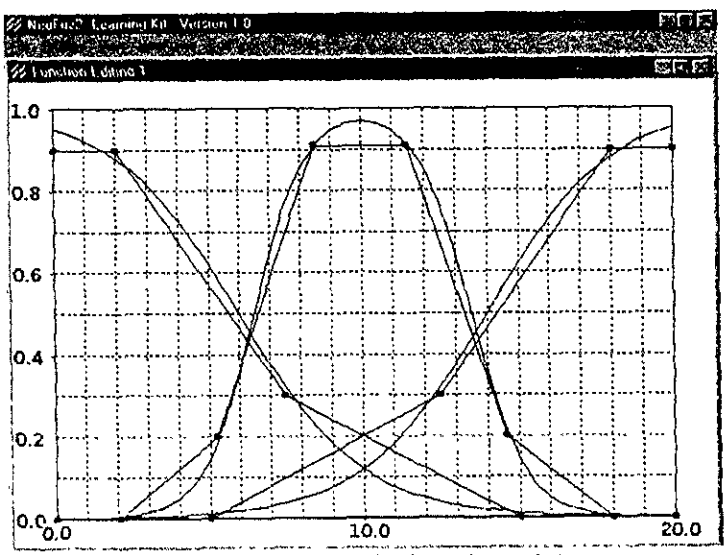


Figura 3. 19 Funciones de membresía para la entrada 1.

2. Cuando una entrada es representada por un gran número de funciones de membresía, por lo general es fácil editar individualmente cada una de ellas. Para editarlas hay que hacer lo siguiente:

   a) Seleccionar None del menú MemFuncs.
   b) Seleccionar 1 del menu MemFuncs.
   c) Seleccionar None del menú EditFuncs.

   La figura 3.20 muestra la ventana "Function Editing 1" con una curva en forma de campana representando la primera función de membresía.

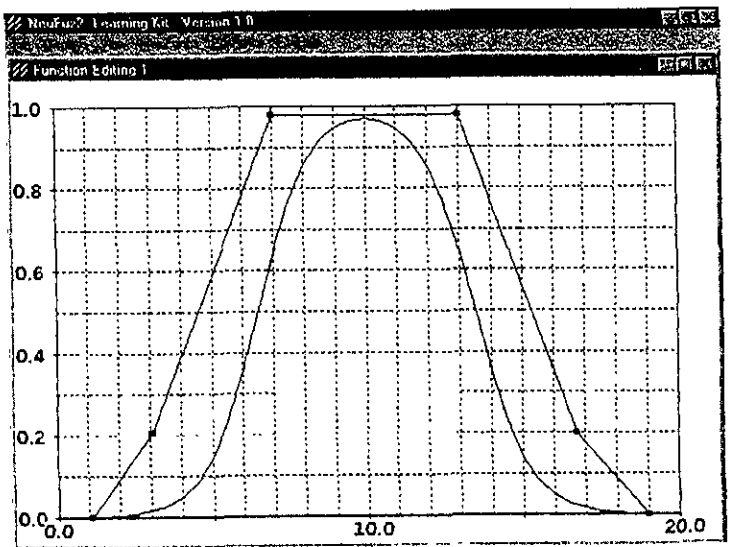


Figura 3. 20 Una función de membresía.

3 Para mostrar la aproximación de la curva, hay que seleccionar 1 del menú EditFuncs. Un trapezoide se muestra sobre la curva Nótese que el trapezoide es una pobre aproximación de la curva de la función de membresía.

4. Para cambiar la forma del trapezoide, hay que hacer clic sobre el vértice que se desee mover. Mover el cursor hacia el punto deseado y volver hacer clic. Se deben mover los

vértices restantes para mejorar la aproximación de la curva. La figura 3.21 muestra el trapezoide después de ser editado.

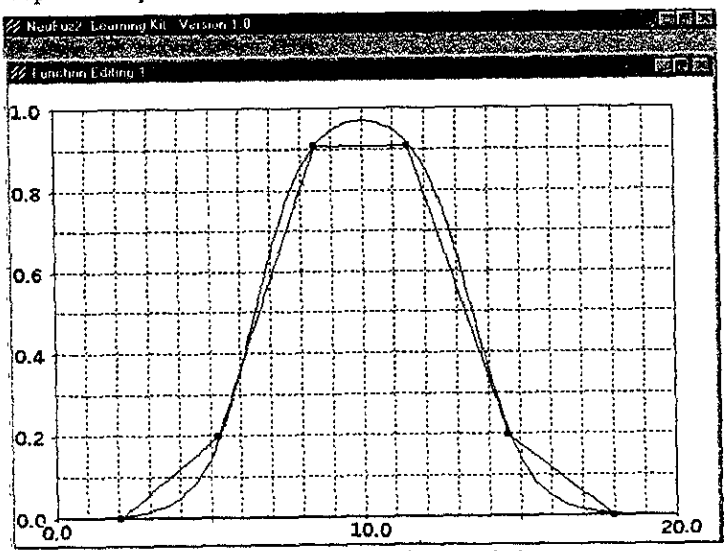


Figura 3. 21 Trapezoide después de ser editado.

5. Para editar los trapezoides del resto de las funciones de membresía para la primera entrada, se debe repetir el paso 1 y seleccionar el resto de las funciones de membresía.

Una vez se ha terminado de editar los trapezoides se debe seleccionar el botón de control del menú de la ventana y seleccionar Cerrar, de esta forma cualquier cambio será guardado.

Verificación de las reglas

Se puede usar la ventana "Rules Verification" para ver gráficamente el aprendizaje de las relaciones de entrada y salida. Esta ventana permite realizar el análisis siguiente:

1. La diferencia entre la salida de la red neuronal contra la salida de las reglas difusas no lineales de las funciones de membresía
2. La cantidad de error introducida por la aproximación lineal de las reglas difusas no lineales de las funciones de membresía

Para iniciar el proceso de verificación hay que:

1. Seleccionar "Rules Verification" en el menu Display. La ventana se muestra tal y como se ve en la figura 3.22.

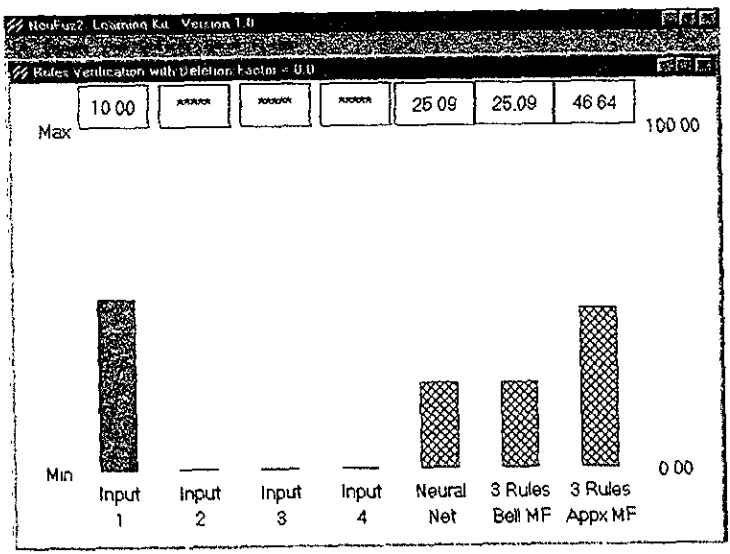


Figura 3. 22 Ventana de verificación de reglas.

Los valores que aparecen a la izquierda de la ventana son los valores de entrada. En este ejemplo se uso una entrada nada más. Tres columnas de salida aparecen en el extremo de la derecha, cada columna representa:

- Salida de la red neuronal (Neural Net).
- Salida de la evaluación de las reglas difusas usando las formas de campana de las funciones de membresía.(Bell MF)
- Salida de la evaluación de las reglas difusas usando las funciones de membresía editadas(Appx MF)

Se puede revisar que los valores de salida están dentro de un rango de exactitud aceptable, al comparar la diferencia entre estas tres salidas.

2. Se puede revisar la exactitud para diferentes valores de entrada. Para hacerlo se debe colocar el puntero de cursor dentro del cuerpo de las entradas y mientras se mantiene presionado el botón del ratón se debe subir o bajar la columna seleccionada. La figura 3.23 muestra la ventana de verificación de reglas después de cambiar la entrada 1 a 5.84. Nótese que los tres valores de salida son casi idénticos.

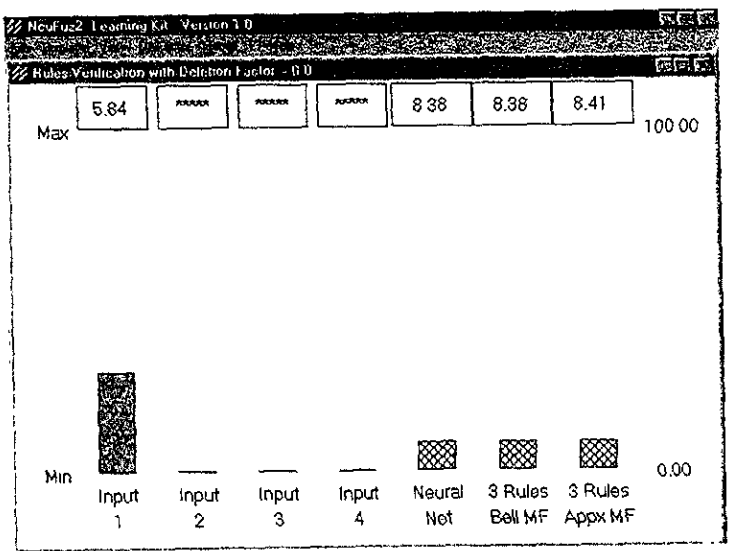


Figura 3. 23 Verificación de reglas(modificación de valores de entrada).

Borrado de reglas marginales

Como ya se menciono el factor de borrado es usado para optimizar el número de reglas lógicas difusas. Pocas reglas significan poco espacio de código y un mejor desempeño en el código de lógica difusa.

El factor de borrado es un valor comprendido entre 0 y 1, mientras más grande sea el número borrara más reglas El valor a usar depende del número marginal de reglas, y que tan marginales

son. Un pequeño incremento en el factor de borrado puede remover un importante número de reglas, o bien podría no remover ninguna.

Previo a la configuración del factor de borrado, se debe examinar el archivo de reglas para conseguir un diseño preliminar de cuantas reglas marginales deben considerarse. Este archivo puede verse haciendo clic en la opción "View" del menú de la ventana de entrenamiento, solamente hay que seleccionar el archivo con extensión ".neu".

Se debe iniciar con un valor pequeño tal como 0.1 y realizar los siguientes dos pasos hasta que se encuentre un factor de borrado que resulte en un aceptable equilibrio entre las reglas y la exactitud requerida para la solución.

1. Ingrese el valor para el factor de borrado, al aceptar el nuevo valor la ventana de verificación de reglas se actualiza automáticamente mostrando los cambios en el número de reglas que aparecen debajo de las barras "Bell MF" y "Appx MF".
2. Se deben revisar los resultados usando las herramientas de la ventana de verificación de reglas. Si el resultado a pesar de eso no es aceptable, hay que tratar con un valor grande para el factor de borrado.

Verificando el archivo de memoria

La salida de la función de verificación de memoria es un archivo con una extensión".out". El archivo contiene los valores de entrada y los tres valores de salida mostrados en la ventana de verificación de reglas Neural Net, Bell MF y Appx MF.

Para hacer uso de esta función se debe hacer:

1. Activar la ventana de verificación de reglas.
2. Seleccionar "Recall" en el menú.
3. Seleccionar "Run Recall"
4. Examinar el archivo "*.out"

Otras sugerencias para realizar pruebas

Para terminar este apartado hay que decir que la evaluación puede incluir mucho más que solo verificar las relaciones normales de entrada/salida usando los datos de memoria. Quizás sea útil el probar para condiciones de error, por ejemplo, la simulación del archivo de memoria para las siguientes situaciones pueden considerarse:

- Simulación de la entrada de ruido.
- Entrada muerta fija en un valor alto o bajo.
- Entrada fija en algún valor intermedio.
- Prueba de casos con los peores casos de entradas posibles.

3.3.6. Traducción de las reglas difusas en código ensamblador para el COP8.

NeuFuz cuenta con una herramienta para traducir las reglas difusas y las funciones de membresía en código ensamblador para el procesador COP8. El código generado combina la información de salida del proceso de aprendizaje de la red neuronal y parámetros de entrada, para producir un programa para el COP8. El programa difuso calcula un valor de salida a partir de un vector de entrada que se basa en un conjunto de reglas difusas y de funciones de membresía. Este programa difuso en combinación con la aplicación provee una solución completa para el sistema.

Archivos de entrada, parámetros de entrada y archivos de salida

Los archivos de entrada usados por el generador de código son automáticamente generados por NeuFuz, no es necesario crear o ver estos archivos. Los archivos de salida producidos son usados para desarrollar por completo la solución del sistema.

- Archivos de entrada

El generador de código requiere de dos archivos de entrada

- n4.lib
- *model*.neu

El generador de código usa la información almacenada en estos archivos para producir el código especifico del COP8, para implementar la solución difusa en el modelo en cuestión.

El archivo n4.lib contiene un conjunto de módulos de código genérico ensamblador del COP8, este archivo contiene datos binarios por lo tanto no se debe alterar la información usando un editor de textos ASCII.

El archivo *model*.neu es un archivo ASCII que contiene información que trata con la solución difusa del modelo tratado. Esta información incluye un conjunto de reglas difusas basadas en los pesos asignados a la red neuronal durante el aprendizaje, además de un conjunto de descripciones de las funciones de membresía basados en la edición que se hizo de éstas.

NeuFuz automáticamente crea el archivo *model*.neu la primera vez que la red neuronal hace alto. Esto ocurre cuando se alcanza la convergencia o se detiene el proceso de aprendizaje haciendo clic en el botón "STOP". NeuFuz actualiza el archivo *model*.neu cada vez que:

- Se detiene el proceso de aprendizaje.
- Cuando el factor de borrado es cambiado.
- Cuando se editan las funciones de membresía.

- Parámetros de entrada

Los parámetros de entrada usados por el generador de código incluyen: nombre del archivo de reglas, directorio de trabajo y nombre del archivo log. Se tiene que especificar el nombre del archivo de reglas para el modelo para el cual se desea generar el código. Este no tiene que ser el nombre del modelo actualmente cargado en NeuFuz, puede ser el nombre de cualquier modelo que tenga un archivo de reglas existente. El archivo log es por defecto el nombre del archivo de las reglas y tiene una extensión ".log". Si se desea se puede cambiar éste una vez que el archivo de reglas a sido especificado. El directorio de trabajo especifica el directorio que contiene el archivo de reglas.

Se puede especificar el tamaño de memoria ROM y RAM que usará el dispositivo COP8 seleccionado para la aplicación. Esta información permite al generador de código el verificar que la solución que produzca puede ser implementada en el dispositivo. Si la memoria es inadecuada, el generador de código imprime un aviso. En este punto, se puede seleccionar un nuevo dispositivo con más memoria o revisar la solución difusa.

- Archivos de salida

El generador de código produce dos archivos de salida:

- *model*.asm
- *model*.log

El archivo "*model*.asm" contiene el código ensamblador que implementa la solución difusa. El archivo "*model*.log" contiene información acerca del código del COP8 tal como la memoria ROM y RAM usada.

Generación del código

Una vez que se ha terminado con el entrenamiento de la red neuronal y con la verificación de las reglas difusas y de las funciones de membresía, se puede generar el código para el COP8 usando la rutina del generador de código. El código ensamblador del COP8 es generado usando las reglas y las funciones de membresía editadas que se describen en el archivo *model*.neu.

Para generar el código hay que hacer lo siguiente:

1. Activar la ventana de entrenamiento.
2. Verificar que el nombre correcto del modelo esta especificado.
3. Seleccionar la opción "Code Generation"en el menu "Display". Se muestra una venta, tal y como se muestra en la figura 3.24

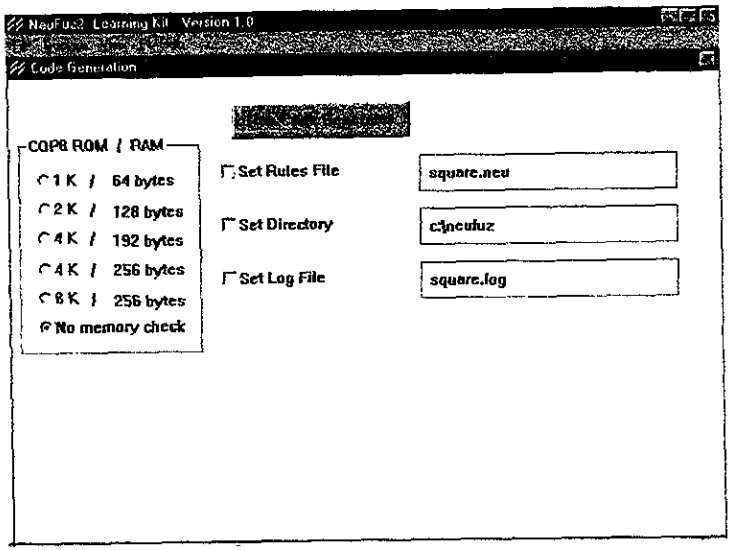


Figura 3. 24 Ventana de Generación de código.

4. Fijar el tamaño de memoria ROM y RAM para el dispositivo COP8 usado, seleccionado el botón apropiado de la izquierda.
5. Revisar los parámetros del generador de código para asegurar que contienen los valores correctos para los archivos de reglas, directorio de trabajo y el archivo log. Si los valores no son correctos, se pueden cambiar.
6. Iniciar la generación del código seleccionando "Run Code Generator". Una vez que el código a sido generado se muestra una ventana informando el resultado. El resultado incluye información tal como el número de reglas, la cantidad de memoria ROM y RAM usada, y los nombres de los archivos: reglas: asociadas(*model*.neu), log(*model*.log) y código(*model*.asm).

Interpretación del archivo log

Este archivo incluye información tal como: los rangos de la variable de entrada, los rangos de la variable de salida, el factor de escala de pesos, y la cantidad de memoria ROM y RAM usada. La figura 3.25 muestra un ejemplo del contenido de este archivo

```
Rules  "square.neu"
Log    "square.log"
Code   "square.asm"
Found 3 rules, 1 input, 1 output
  Variable name    Input1
       Range min   0.000000 ( h'00 )
       Range max   20.000000 ( h'FF )
  Mem Functions    L1  L2  L3
Asm code symbol    I1
  Variable name    Output
       Range min   0.000000 ( h'00000000 )
       Range max   100.000000 ( h'00190000
 Weights factor    h'80
Asm code symbol    OUT1    ; low order byte
                   OUT2
                   OUT3
                   OUT4
                   OUT5
ROM usage:
----------
            Code    518
MF Data    MFTBL     27
Rule Data RTBLn       9
            Total   554 bytes
RAM usage:
----------
         BASE RAM      6
          REG RAM      3
            Stack      8
        Other RAM     22
            Total     39 bytes
```

Figura 3. 25 Archivo log.

La primera sección del archivo muestra el número de reglas difusas, variables de entrada y variables de salida asociadas con el sistema. También se lista el nombre y el rango de cada variable. Se deberá usar los valores Range MIN y Range MÁX. de la variable de entrada para la ecuación de acondicionamiento para el sistema de entrada. Adicionalmente la primera sección lista el nombre de la variable de salida y el factor de grado del peso El factor de grado del peso es necesario para reacondicionar el valor de salida generado por el código del COP8.

La segunda sección del archivo delinea la memoria de programación usada por el código difuso. El valor MFTBL corresponde al número de bytes requeridos para almacenar la tabla de las funciones de membresia. El valor RTBLn es igual al número de bytes requeridos para almacenar las reglas difusas. El número total es el total de ROM usada para el almacenamiento del código y de los datos.

La tercera sección delinea los datos de memoria usados. La sección de datos de memoria BASE y REG RAM son especiales y deberá consultarse el *Assambler/Linker/Librarian User's Manual* para mayor información. El número siguiente a Stack se refiere a la cantidad de memoria usada para almacenar llamados de subrutina de retorno de direcciones. Antes de llamar la subrutina FUZZ se debe asegurar que el stack tiene suficiente espacio por expandir para este número de bytes. La otra sección RAM es para propósitos generales de datos de memoria que no tiene asignada una función especial. El número total es la suma de toda la RAM usada por el código difuso.

Interpretación del archivo de código COP8

El código de programación producido por el generador de código tiene tres componentes básicos:

- Una tabla que contiene las descripciones de las funciones de membresía.
- Una tabla que contiene las reglas difusas.
- Código para acceder a las tablas y ejecutar el proceso de evaluación de las reglas (obsérvese que el proceso de desdifusificación esta integrado con el proceso de evaluación de reglas)

Dentro del archivo, estos componentes están divididos en módulos de código ensamblador. Cada módulo contiene rutinas o datos para varias funciones específicas en el proceso de difusificación de las reglas de evaluación. Adicionales módulos de código son incluidos para la declaración de variables de almacenamiento La tabla siguiente lista los módulos de código y sus funciones básicas.

Nombre del Módulo (Sección de memoria)	Tipo de Memoria	Rutinas	Función
FUZZ	ROM	FUZZ	Rutina principal
		GOFUZ	Rutina principal; difusificación
		MFDCALC	Calcula el grado de membresía para una entrada
MFCODE	ROM	MFPLUP	Tabla de búsqueda de datos de funciones de membresía

MFTABLE	ROM Block	IMFTBL	Almacenamiento de la tabla de funciones de membresía
RULECODE	ROM	RULEVAL	Rutina principal, evaluación de reglas
		COUT	Calcula la salida a partir de cada regla y suma el resultado
		RDOM	Busca DoM para cada regla antecedente
RULE	ROM Block	RULELUP	Calcula la locación para el almacenamiento de reglas
RULEn	ROM Block	LUPRSn	Tabla de búsqueda y almacenaje de reglas
MATH	ROM	DIV	División de 16 bit por 8 bit
		MUL08	Multiplicación de 8 bit por 8 bit
		MUL16	Multiplicación de 16 bit por 8 bit
		MUL24	Multiplicación de 24 bit por 8 bit
FUZDAT	ROM	None	Variable de almacenamiento
FAST		None	Variable de almacenamiento
TOP		None	Variable de almacenamiento

Se puede usar el ensamblador COP800 para ligar e integrar estos módulos con los módulos de la aplicación y producir un archivo ejecutable. Además para los módulos de código, el archivo "*model*.asm" contiene un conjunto de declaraciones constantes. Estas definiciones son usadas como constantes globales dentro del archivo "*model*.asm".

Interconexión al código difuso del COP8

La interconexión entre el código de la aplicación y el código de programación difuso es muy simple Un conjunto de registros se define por el paso de parámetros y a partir del código difuso.

El número de rangos de registros de entrada desde uno a cuatro es equivalente al número de entradas en el sistema. Los registros de entrada son definidos como I1, I2, I3 e I4. Se deben cargar estos registros, tanto como se necesiten, con los valores de entrada para el sistema antes de llamar a la rutina FUZZ en el código difuso. Los registros de salida están definidos como OUT1, OUT2, OUT3, OUT4 y OUT5. Estos registros regresan un resultado de salida de un complemento a dos de 5 bytes. El byte de orden más pequeño es almacenado en OUT1.

- **Entradas acondicionadas**

Los valores de entrada deben ser acondicionados antes de ser pasados a la rutina FUZZ. El acondicionamiento de estos valores es muy importante por que el código difuso asume que factores específicos de ajuste están presentes en los valores de entrada. La siguiente formula puede ser usada para acondicionar cada entrada:

$$EntradaAcondicionada = \frac{(entrada - min) \times 256}{(max - min)}$$

donde:

EntradAcondicionada = valor que pasara a el código difuso
entrada = valor de entrada del sensor
min = valor mínimo en el rango de entrada(referido al archivo log para este valor)
max = valor máximo en el rango de entrada(referido al archivo log para este valor)

El resultado de esta ecuación es siempre un número decimal entre –1 y 255. Si el resultado es un número negativo, se redondea a cero. Este redondeo causa solamente un pequeño decremento en la exactitud para algunas entradas. Se deben cargar los valores de entrada acondicionados dentro de I1 a I4 antes de invocar el código difuso.

- Llamando al código difuso.

El código difuso en invocado al ejecutar la instrucción JSR FUZZ. El siguiente es un ejemplo simple de una rutina para configurar las entradas y llamar el código difuso.

```
LD      A,Input1        ;Almacena el valor de la entrada graduada 1 en I1
X       A,I1            ,
LD      A,Input2        ;Almacena el valor de la entrada graduada 2 en I2
X       A,I2            ;
JSR     FUZZ            ,Llama al código difuso
```

- Reacondicionando la salida

El número regresado de 5 bytes por la rutina FUZZ contiene dos factores de ajuste que deben ser removidos para producir un correcto valor de salida. Los dos factores de ajuste son: factor de ajuste de función de membresía y el factor de ajuste para pesos. El factor de ajuste de función de membresía es introducido en el acondicionamiento de los datos de la función de membresía. Este factor de ajuste es siempre el 128 decimal. El factor de ajuste de pesos es introducido en el acondicionamiento de los pesos de las reglas difusas. Este factor se basa en el rango de pesos de salida de las reglas difusas. Se puede encontrar el factor de pesos para la aplicación por la lectura del archivo log producida por generador de código.

La fórmula siguiente se usa para remover el factor de grado de un valor de salida:

$$\textbf{\textit{ValorSalida}} = \frac{\textbf{\textit{ValorSalidaAcondicionado}}}{128 \times \textbf{\textit{FactorPeso}}}$$

donde

ValorSalida = valor no graduado, valor de salida usado en el entrenamiento de la red
ValorSalidaAcondicionado = resultado de la operación difusa graduado por 128 × ***FactorPeso***
128 = descripción de factor de grado para función de membresía
FactorPeso = factor de grado aplicado a el peso por el generador de código
(referido al archivo " log")

Se debe recordar que la salida producida es un número de complemento dos Si la salida del sistema es siempre positiva, se necesitará tomar esto en cuenta cuando se evalúe la salida del código difuso.

Ensamblando / Uniendo el código difuso

El archivo *model*.asm producido por el generador de código debe ser ensamblado y unido con el código de la aplicación. Primero se debe ensamblar el archivo *model*.asm usando el ensamblador AMCOP de National Semiconductor's. Entonces se ensambla el código de los módulos de la aplicación. Finalmente se unen los archivos objetos producidos por el programa ensamblador. El programa LNCOP de National Semiconductor's se debe usar para este paso. Para unir completamente el código difuso y el código de la aplicación se tiene que declarar los registros difusos y las rutinas usadas por la aplicación tal y como se definieron externamente.

Para un sistema de 4 entradas, esta declaración es escrita como sigue:

```
.SECT FUZDAT,RAM
.EXTRN I1,I2,I3,I4
.EXTRN OUT1,OUT2,OUT3,OUT4,OUT5
.SECT FUZZ,ROM
.ENDSECT
```

Para sistemas de 1,2 ó 3 entradas, simplemente se remueve las declaraciones no utilizadas I2,I3,I4.

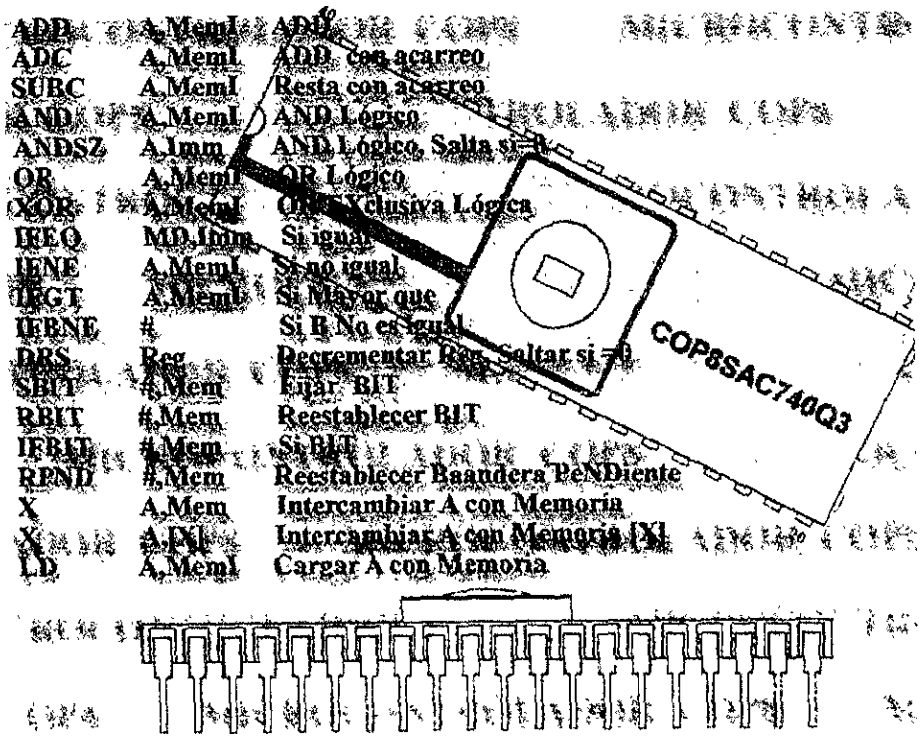
Si se desea, se pueden usar las rutinas de multiplicación y división suministradas como parte del código difuso. Estas rutinas y registros deben declararse también externamente Esta declaración se escribe como sigue:

```
.SECT FAST,RAM
.EXTRN MULT1,MULT2,MULT3
.EXTRN PRD1,PROD2,PROD3,PROD4,PROD5
.EXTRN DIVR1,DIVR2,DIVR3
.EXTRN QUO1,QUO2,REM1,REM2
.SECT MATH,ROM
.EXTRN DIV,MUL08,MUL16,MUL24
.ENDSECT
```

Las utilerías ASMCOP y LNCOP están descritas a detalle en el *COP8 Asasmbler/Linker /Librarian Manual*. El archivo map producido por el Linker da información sobre la asignación final de las localidades de ROM para módulos y de RAM para variables

CAPITULO 4

EL MICROCONTROLADOR COP8



CAPITULO 4 EL MICROCONTROLADOR COP8

4.1. Introducción

Es común en el mercado actual la aparición de nuevos microprocesadores, estos van desde microcontroladores de muy bajo costo y prestaciones mínimas, hasta microprocesadores de alto costo y elevado desempeño de hardware y software. El uso de un microprocesador o de un microcontrolador dependerá de la aplicación a la cual se vaya a destinar.

4.1.1 ¿Qué es un microcontrolador?

La categoría de los microcomputadores se divide en dos áreas: microprocesadores y microcontroladores. Esta distinción es por que en realidad son dos tipos diferentes de dispositivos. Los microcontroladores tienen una arquitectura de bus-dual a diferencia de la arquitectura de memoria mapeada Von Neumann(más adelante se trata a mayor detalle) común en la mayoría de los microprocesadores. Para aplicaciones de control los microcontroladores son más eficientes en el uso de memoria que los microprocesadores. El conjunto de instrucciones de un microcontrolador es diferente en naturaleza que el de un microprocesador. Los microcontroladores son invariablemente dispositivos de un solo chip y los microprocesadores son, generalmente, dispositivos de múltiples chips. Sin duda la división de microcontroladores y microprocesadores es un tanto vaga, pero la distinción es real sin duda.

Un microprocesador es un sistema abierto con el que se puede construirse una computadora con las características que se desee, acoplándole los módulos necesarios A su vez un microcontrolador es un sistema cerrado que contiene un computador completo y de prestaciones limitadas que no se pueden modificar.

Los microcontroladores, como otros productos considerados en retrospectiva han despuntado Surgen de dos caminos complementarios: necesidades del mercado y nueva tecnología La nueva tecnología se refiere a nuevos semiconductores con mas transistores y usando menos espacio, más productividad a un menor costo. La necesidad del mercado es la industria y el deseo del consumidor para tener herramientas más sofisticadas; abarcando una gran área.

Los microcontroladores son especializados, no se utilizan en computadoras per se, pero sí en la industria y productos dirigidos al consumidor final. Los usuarios de tales productos son muchas de las veces ignorantes de la existencia de microcontroladores, para ellos los componentes internos son detalles de diseño sin importancia. Como ejemplo considérese los hornos de microondas, termostatos programables, básculas electrónicas e inclusive automóviles. La electrónica dentro de cada uno de estos productos incorpora generalmente una interfase para comunicarse con el microcontrolador, ésta puede incluir botones de selección, interruptores, luces indicadoras, y alarmas en un panel frontal; aun más la operación del usuario remeda a su antecesor electromecánico. Por lo tanto el microcontrolador es invisible para el usuario

Algunos de los elementos que incluye un microcontrolador son:

- **CPU**

La Unidad Central de Proceso (CPU por sus siglas en Ingles) es el corazón del microcontrolador, donde todas las operaciones aritméticas y lógicas se desarrollan. Esta es la parte que realiza todos los cálculos. La CPU recibe instrucciones de su memoria de programa.

- **Programa en memoria**

El programa en memoria contiene un conjunto de instrucciones de la CPU organizadas en una secuencia en particular para realizar una tarea específica. La CPU es conocida como Read Only Memory (ROM) o OTP/EPROM. OTP o "One-Time Programmable" que puede programarse una sola vez almacenando el programa permanentemente, aun cuando el microcontrolador se apaga el programa de memoria permite arrancar el programa tan pronto que sea encendido el microcontrolador.

- **Memoria de datos**

Un tipo de memoria que puede ser de lectura o de escritura es requerida por el área de memoria de programa (program stack), almacenamiento de datos y las variables del programa. Este tipo de memoria es conocida como memoria Random Access Memory (RAM). Cada localidad de memoria tiene definida una dirección única que la CPU usa para encontrar la información que necesita.

Un controlador típico usa ambos tipos de memoria ROM y RAM.

- **Circuito temporizador**

Los microcontroladores usan una señal, llamada reloj, para proporcionar una referencia de tiempo para la ejecución del programa, y para determinar cuando los datos deberán ser escritos o leídos de la memoria. Esta referencia de tiempo puede usarse también por los periféricos.

- **Líneas de Entrada / Salida**

Los microcontroladores requieren secciones de interfase para poder comunicarse con los periféricos externos que controla. Los puertos de entrada permiten aceptar datos y condiciones de estado para ser leídos dentro del microcontrolador, mientras el puerto de salida permite al microcontrolador afectar sistemas lógicos externos. La interfase entre el microcontrolador y el mundo real varia dependiendo la aplicación, y puede incluir unidades de display, teclados, interruptores, sensores, relevadores, motores y así por el estilo.

La figura 4.1 muestra un diagrama de estos elementos.

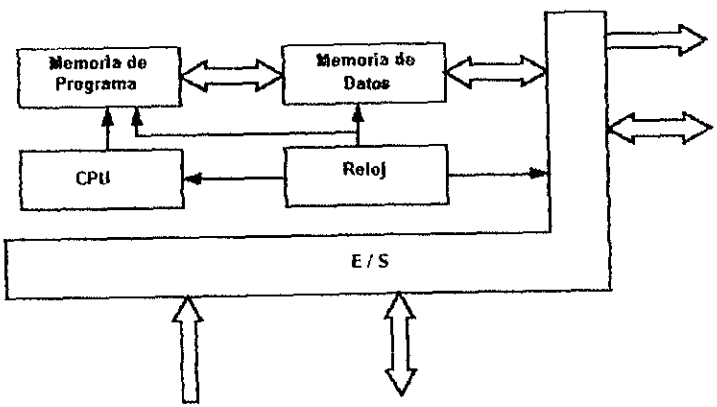


Figura 4. 1 Diagrama general de bloques de un microcontrolador.

4.1.2 Arquitectura de un microcontrolador

Los microcontroladores tienen dos tipos de arquitectura. Von Neumann o Harvard

instrucciones especiales para acceder a los datos RAM y ROM, haciendo la programación más difícil.

4.1.3 Operación interna de un microcontrolador

La CPU puede demandar información de la memoria (o lectura de un puerto de entrada) por la llamada de su dirección de memoria. La dirección con todos sus bits es almacenada en la CPU como un número binario en un dato cerrado (perteneciente a un tipo de memoria) llamado registro. Las salidas del registro son enviadas sobre múltiples líneas de conexión (o en una línea individual) a la memoria del microcontrolador y a los periféricos. El grupo de líneas (paralelo) o la línea individual (serial) que lleva la dirección es llamado el bus de direcciones. La palabra "bus" se refiere a una o más líneas que comparten un camino común a (o de) múltiples lugares. El registro de direcciones guarda los bits de dirección. El número de bits de dirección depende del tipo de microcontrolador usado.

El dato es enviado a la CPU sobre el bus de datos. El bus de datos es diferente del bus de direcciones en que la CPU usa el primero para leer o escribir información a la memoria o a los periféricos. Las señales en el bus de direcciones son originadas solamente en la CPU y son enviadas a los otros bloques por medio del bus. Las señales en el bus de datos pueden ser señales o entradas de la CPU. La información en el bus de datos es enviada y recibida en la CPU por el registro de datos En otras palabras, el bus de datos es bi-direccional y el bus de direcciones es uni-direccional. El ancho de la dirección y el bus de datos pueden también ser diferentes, dependiendo del tipo de microcontrolador y tamaño de memoria que se trate. La figura 4.2 muestra un diagrama de esto.

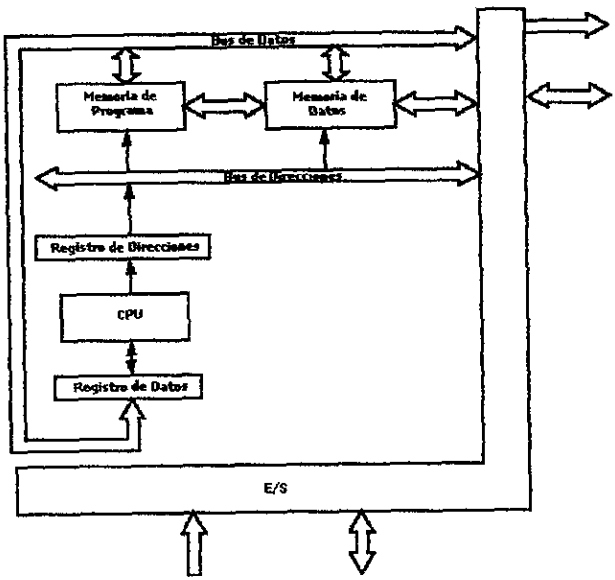


Figura 4. 2 Operación interna del Microcontrolador.

4.2 El microcontrolador COP8SAx7

El microcontrolador COP8SAx7 es un miembro de la familia COP8, familia que se caracteriza por usar un simple chip de 8-bit como corazón de su arquitectura. Este dispositivo es fabricado por National Semiconductor's mediante un proceso EPROM de alta densidad, y disponible en varios tipos de empaquetamiento, rangos de temperatura y voltaje, para satisfacer una gran variedad de aplicaciones.

Las características clave incluyen arquitectura mapeada de memoria de 8-bit, un contador y temporizador de 16-bit con dos registros de 16-bit que soportan tres modos: Generación PWM de Procesador Independiente, Contador de Externo de Eventos y Disponibilidad de Captura de Entradas, dos modos de bajo consumo HALT e IDLE con capacidad de activación o interrupción a partir de múltiples pines, un oscilador R/C interno, salidas de corriente elevadas, opciones seleccionables para el usuario, tales como WATCHDOG, configuración del oscilador y reestablecimiento de la activación.

4.2.1 Características a destacar

Las siguientes son características importantes a destacar del COP8SAx7:

- Microcontrolador de bajo costo de 8-bit OTP (Programable una vez)
- Espacio de programa OTP con protección de escritura y lectura.
- Diseño silencioso (baja irradiación electromagnética)
- Múltiples pines de entrada para activación con opción de interrupciones (4 a 8 pines).
- 8-bit de espacio para almacenamiento de instrucciones en memoria EPROM.
- Opciones de temporizador seleccionables por el usuario:
    - + Oscilador cristal/resonador.
    - + Oscilador cristal/resonador con resistor de polarización interno.
    - + Oscilador externo.
    - + Oscilador R/C interno.
- Selección interna del usuario del restablecimiento de la activación.
- Monitor de operación (WATCHDOG) y monitor del temporizador seleccionado por el usuario.
- Hasta 12 salidas de alta corriente.

La figura 4.3 muestra una tabla con algunos integrantes de la familia COP8SAx7 con algunas características importantes.

Dispositivo	EPROM	RAM	Encapsulado y E/S	
			Tipo de encapsulado	Número de E/S
COP8SAC7	4k	128	20 DIP/SO 28 DIP/SO 40 DIP 44 PLCC/PQFP	16 24 36 40
COP8SAB7	2k	128	20 DIP/SO 28 DIP/SO	16 24
COP8SAA7	1k	64	16 DIP/SO 20 DIP/SO 28 DIP/SO	12 16 24

Figura 4. 3 Integrantes de la familia COP8SAx7

4.3 Resumen de Hardware

Los microcontroladores COP8SAx7 ofrecen varios elementos importantes de hardware tanto en lo que se refiere a la arquitectura interna del núcleo de la CPU como en los periféricos incorporados en el chip.

4.3.1 Características de la CPU

- Conjunto de instrucciones versátil y fácil de usar.
- Ciclo del reloj por instrucción 1 µs
- 8 interruptores de vector de múltiples fuentes
    - + Interruptor externo
    - + Temporizador de inactividad T0
    - + Un temporizador (con dos interruptores)
    - + Interfase Serial MICROWIRE/PLUS
    - + Activación de múltiples entradas
    - + Trampa de Software
    - + VIS de fábrica (interruptor de fábrica)
- Puntero de 8 bit SP (pila en RAM)
- Dos punteros de memoria de datos indirectos de 8-bit
- Manejo de bit real
- Mapeo de memoria de E/S
- Instrucciones aritméticas BCD

4.3.2 Características periféricas

- Lógica de activación de entradas múltiples
- Un temporizador de 16-bit de dos registros de 16-bit que soportan:
    - + Un modo PWM de procesador independiente
    - + Un contador de eventos externos
    - + Un modo de captura de entrada
- Un temporizador de inactividad
- Interfase serial MICROWIRE/PLUS (compatible con SPI)

4.3.3 Características de E/S

- Opciones de E/S seleccionables por software
  - + Salida de tercer estado (TRI-STATE)
  - + Salida de push-pull
  - + Entrada débil pull-up
  - + Entrada de impedancia alta
- Entradas Schmitt trigger en los puertos G y L
- Hasta 12 salidas de corriente alta
- Eficiencia de pines (Ej. 40 pines en un paquete de 44 pines, son destinados a E/S)

4.3.4 Diseño CMOS estático completo

- Drenaje bajo de corriente (normalmente < 4 μ)
- Alimentación de 2.7V a 5.5V
- Dos modos de bajo consumo: HALT e IDLE

4.3.5 Rangos de temperatura

0°C a +70°C, -40°C a +85°C, y -40°C a +125°C

4.3.5 Emisiones EMI

- El diseño total ha sido realizado con un control de velocidad (slew rate) y niveles de señal internas, orientado a la generación de bajos niveles de irradiación electromagnética (EMI· electromagnetic interference), lo que lo hace ideal para su uso en el área de comunicaciones e instrumentación. Esta tecnología permite considerar niveles de generación de interferencias en el orden de 20dB respecto a procesadores similares que operan en la misma frecuencia

4.3.6 Soporte para desarrollo

- Paquetes con marcos para DIP y PLCC
- Emulación en tiempo real y un programa de depuración completo ofrecido por MetaLink Development System.

- Herramientas de evaluación para el COP8 (algunos ejemplos se listan abajo incluyendo el Kit COP8-EPU que se utiliza en este trabajo)

  + COP8-NSEVAL: Software de evaluación gratis para Windows. Un completo medio de evaluación integral para el COP8, incluyendo versiones de WCOP8 IDE un medio de desarrollo integral.
  + COP8-DM· Módulo de depuración de costo medio de MetaLink. Basado en ventanas, herramienta de simulación en circuito de tiempo real con un dispositivo programador del COP8. Incluye COP8-NSDEV, Demo de introducción del COP8, Depurador de MetaLink, fuente de poder, cables de simulación y adaptadores.
  + COP8-EPU· Unidad de programación y evaluación de muy bajo costo. Evaluación basada en ventanas y herramientas de simulación de hardware, con programación de dispositivos COP8 y ejemplos de ensamblaje, incluye COP8-NSDEV, demo introducción del COP8, programa de depuración (debugger) de MetaLink, cables de E/S y fuente de poder.
  + Etc...

  En la figura 4.4 se da el diagrama en bloques para el microcontrolador COP8SAx7.

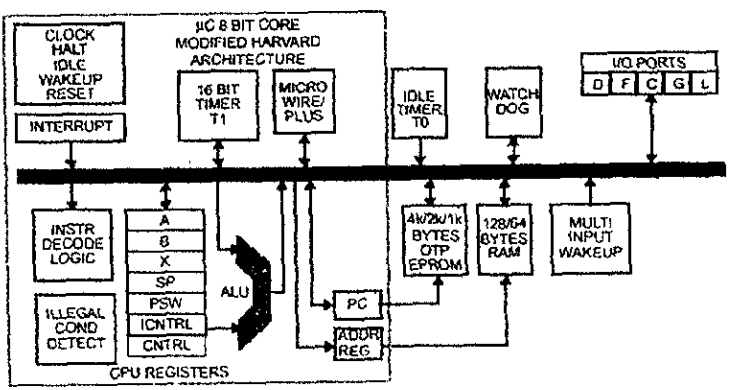


Figura 4. 4 Diagrama de bloques del COP8SAx7.

4.4 Arquitectura

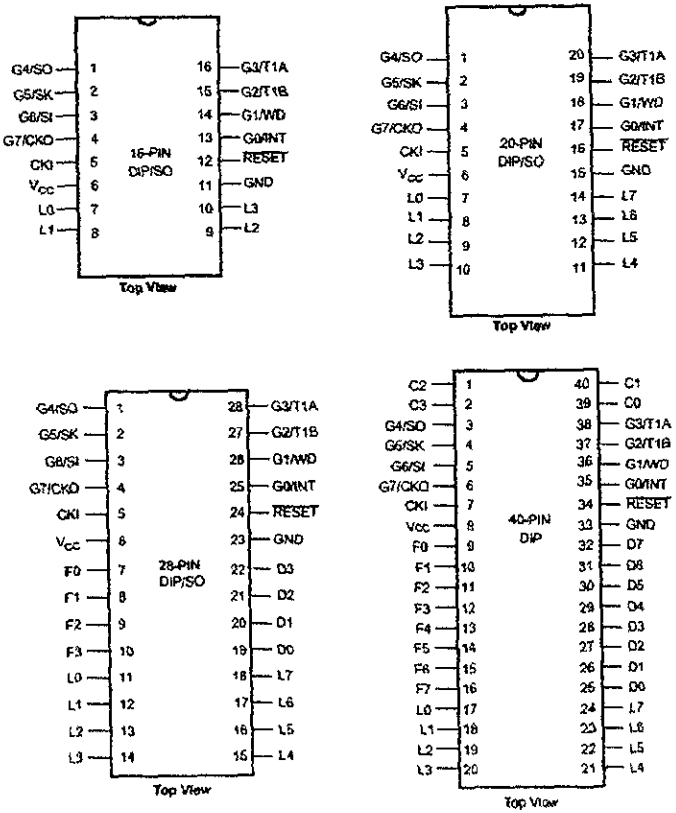
La familia de microcontroladores COP8SAx7 esta basada en una arquitectura Harvard modificada, la cual permite que las tablas de datos se puedan acceder directamente de la memoria de programa. Esto tiene mucha importancia en las aplicaciones con microcontroladores modernos, ya que la memoria de programa es normalmente ROM o EPROM mientras que la memoria de datos es normalmente RAM. En consecuencia generalmente las tablas de datos necesitan estar contenidas en ROM o EPROM, de tal forma que no se pierden cuando el microcontrolador es apagado. En una arquitectura Harvard modificada, las instrucciones llamadas y la transferencia de datos se pueden superponer por medio de una etapa de doble trayectoria, lo que permite que la siguiente instrucción sea llamada de la memoria de programa mientras la instrucción actual esta ejecutándose usando la memoria de datos. Esto no es posible con una arquitectura de bus de direcciones simple como la Von Neumann.

La familia COP8SAx7 soporta un esquema de programación apilada que permite que al usuario incorporar muchos llamados de subrutina. Esta capacidad es importante cuando se usan lenguajes de alto nivel. Con una pila de hardware, el usuario esta limitado a números fijos y pequeños de niveles de pila.

4.5 Descripción de pines

La estructura de los microcontroladores COP8SAx7 minimiza el requerimiento de componentes externos. Por medio del software los diseñadores habilitan las E/S para reconfigurar las funciones de E/S del microcontrolador con una simple instrucción. Cada pin individual de E/S puede ser configurado independientemente como un pin de salida baja, una salida alta, una entrada con impedancia alta o una entrada con un dispositivo de pull-up (envío de datos a petición). Un ejemplo típico es el uso de los contactos de entrada-salida de las líneas de la matriz de un teclado Las líneas de entrada se pueden programar con pull-up de modo que lean un estado lógico alto cuando las teclas están todas para arriba. Con una tecla presionada, la línea de entrada correspondiente leerá un cero lógico entonces el pull-up puede fácilmente ser sobre-mandado. Cuando la tecla es liberada, el pull-up interno envía la línea de entrada de regreso a un estado lógico alto. Esta flexibilidad elimina la necesidad de resistores pull-up externos. Las opciones de corriente alta están disponibles para el manejo de LEDs, motores y bocinas. Esta flexibilidad ayuda para asegurar un diseño más limpio, con menos componentes externos y costos más bajos. La

figura 4.5 muestra un esquema de la disposición de los pines. Enseguida está la descripción general de todos los contactos disponibles.



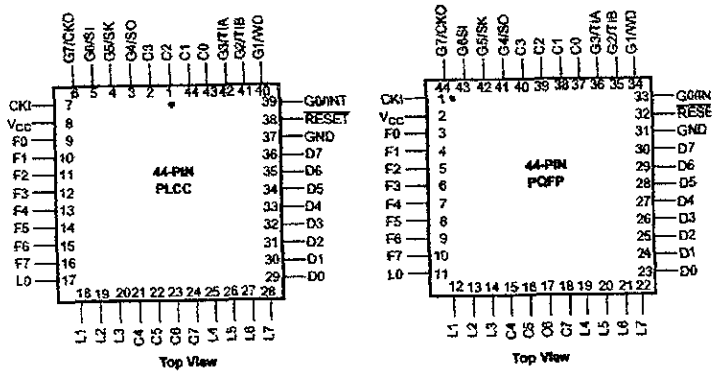


Figura 4. 5 Diagramas de conexión.

Vcc y **GND** son los pines de la fuente de poder. Todos los pines Vcc y GND deben ser conectados.

CKI es la salida del temporizador. Esta puede venir de le oscilador interno R/C, externo, o un oscilador de cristal (en conjunción con CKO)

$\overline{\text{RESET}}$ es la entrada de re-inicialización principal.

El dispositivo contiene cuatro puertos bi-direccionales de entrada-salida de 8-bit (C, G, L y F), donde cada dígito binario individual se puede configurar independientemente como entrada de información (entradas de información del tipo Schmitt trigger en los puertos L y G), salida o el control de programa de triple estado. Tres localidades de dirección de memoria de los datos se asignan para cada uno de estos puertos Cada puerto tiene dos registros asociados de memoria de 8-bit, el registro de CONFIGURACIÓN y el registro de DATOS de salida. Una dirección asociada de memoria también se reserva para los contactos de entrada de cada puerto (véase el mapa de memoria para las diversas direcciones asociadas cada puerto) La figura 4.6 muestra la configuración de los puertos. Los registros de los DATOS y de la CONFIGURACIÓN permiten que cada dígito binario portuario sea configurado individualmente bajo control de software según lo mostrado a continuación:

Registro de CONFIGURACION	Registro de DATOS	Configuración de puerto
0	0	Entrada Hi-Z (Salida TRI-STATE)
0	1	Entrada con Weak Pull-Up
1	0	Salida cero Push-Pull
1	1	Salida única Push-Pull

El puerto L es un puerto de 8-bit de entrada-salida. Todos los pines L son del tipo Schmitt trigger, en los 8 contactos soporta la característica de activación por múltiples entradas. El dispositivo de 16bit no tiene por completo los ocho contactos del puerto L. Los pines no disponibles no están terminados, por lo que la operación de lectura en uno de estos pines regresara valores impredecibles. Para minimizar la corriente drenada, el pin no disponible debe programarse como salida.

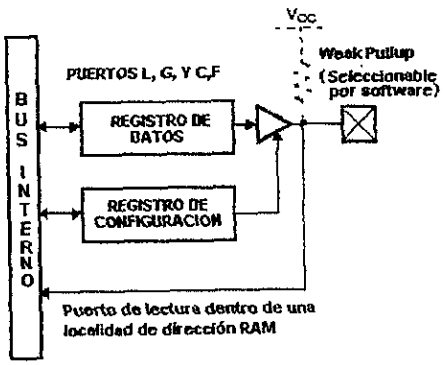


Figura 4. 6 Configuración de puertos de E/S.

El puerto G es un puerto de 8-bit Los pines G0, G2-G5 son puertos bi-direccionales de entrada-salida. El pin G6 siempre es una entrada de información de propósitos generales Hi-Z. Todos los pines tienen Schmitt trigger en sus entradas. El pin G1 sirve como la salida del WATCHDOG (monitor de operación) dedicado WDOUT con el pull-up débil si la característica de monitor de operación es seleccionada con el registro ECON. El pin es de entrada-salida y de propósito general si la característica de "monitor de operación" no se selecciona. Si se seleccionan la característica "monitor de operación", el bit 1 del registro de configuración y el registro de datos

del puerto G, entonces, no tiene ningún efecto el pin G1. El pin G7 puede ser de entrada o salida dependiendo de la opción de oscilador seleccionada. Con la opción de oscilador-cristal seleccionada, G7 sirve como pin dedicado de salida del temporizador de CKO. Con el R/C interno o la opción del oscilador externo seleccionada, G7 sirve como un pin de entrada de propósito general Hi-Z y también se utiliza para traer al dispositivo del modo de bajo consumo HALT con una transición de bajo a alto en G7. Hay dos registros asociados al puerto G, un registro de datos y un registro de configuración. Usando estos registros, cada uno de los 5 pines de entrada-salida (G0, G2-G5) pueden ser configurados individualmente bajo control de software

Como G6 es solamente un pin de entrada y el pin 7 es un pin de salida dedicado del temporizador CKO (opción temporizador-cristal) o una entrada de propósito general (R/C u opción de temporizador externo), los bits asociados en los registros de datos y configuración para G6 y G7 son usados para funciones de propósito general como se enmarca abajo. La lectura de los bits de datos G6 y G7 regresarán ceros

	Reg. Configuración	**Reg. Data**
G7	CLKDLY	HALT
G6	Alternativo SK	IDLE

El puerto G posee las siguientes características alternativas:

G0 INTR (External Interrupt Input)
G2 T1B (Timer T1 Capture Input)
G3 T1A (Timer T1 I/O)
G4 SO (MICROWIRE Serial Data Output)
G5 SK (MICROWIRE Serial Clock)
G6 SI (MICROWIRE Serial Data Input)

Port G has the following dedicated functions:

G1 WDOUT monitor de operación y/o monitor de temporizador WATCHDOG habilitado, si no es una entrada-salida de propósitos generales

G7 CKO de salida dedicado o entrada de propósito general

El puerto C es un puerto de 8-bit de entrada-salida. El dispositivo de los 40-pines no tiene un complemento completo de los pines del puerto C. Los pines inasequibles no están terminados. Una operación leída en éstos pines volverá valores imprevisibles. Solamente el dispositivo COP8SAC7 contiene los pines completos del puerto C. Los dispositivos de 20/28 pines no ofrecen el puerto C. En estos dispositivos, los datos asociados del puerto C y los registros de la configuración no deben ser utilizados.

El puerto F es un puerto de 8-bit de entrada-salida. Los dispositivos de 28 pines no tienen un complemento completo de los pines del puerto F. Los pines inasequibles no se terminan. Una operación leída en éstos indeterminados pines volverá valores imprevisibles.

La figura 4.7 muestra el modo de configuración de salida para el puerto de E/S.

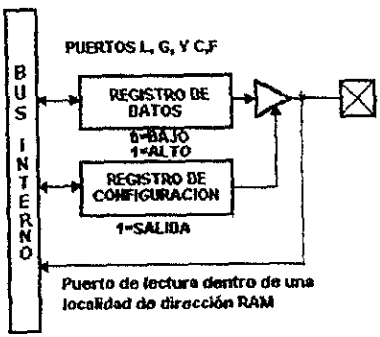


Figura 4. 7 Configuración del modo de salida para el puerto de E/S.

El puerto D es un puerto de salida de 8-bit que se preestablece alto cuando $\overline{\text{RESET}}$ pasa a bajo. El usuario puede atar dos o más salidas del puerto D (excepto D2) juntas para conseguir un mecanismo impulsor más alto.

NOTA: Debe tenerse cuidado con la operacion del pin D2. En el RESET, las cargas externas en este pin deben asegurarse para que los voltajes de salida permanezcan sobre 0,7 Vcc para

evitar que el chip entre en modos especiales. También cuide que la carga externa en D2 sea menos de 1000 pico faradios.

La figura 4.8 muestra la configuración del modo de salida del puerto E/S.

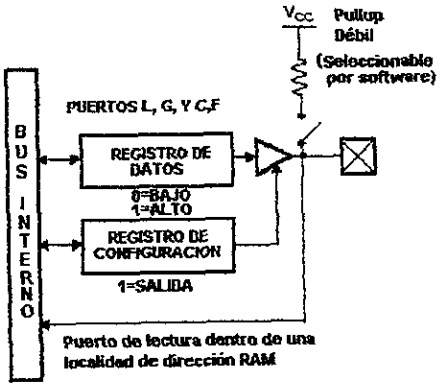


Figura 4. 8 Configuración del modo de salida del puerto E/S.

4.6 Operación del temporizador

El dispositivo contiene un conjunto muy versátil de temporizadores (T0 y T1). T1 y sus registros de encendido de auto-carga y captura (auto-reload /capture) contienen datos aleatorios.

4.6.1 Temporizador T0 (Temporizador IDLE)

El dispositivo soporta con el modo IDLE aplicaciones que requieren mantenerse en tiempo real y potencia baja. El modo IDLE es proporcionado para el temporizador T0. El temporizador T0 corre continuamente en la velocidad establecida del ciclo de reloj de la instrucción, t_c. El usuario no puede leer o escribir al temporizador T0, cuando el temporizador realiza una cuenta hacia abajo.

El temporizador T0 soporta las funciones siguientes;

- Salida del modo Idle.
- Perro guardián lógico (WATCHDOG).
- Activación del retraso de salida del modo HALT.
- Medida (cronometrar) del ancho de la re-inicialización interna.

El temporizador T0 IDLE puede generar una interrupción cuando el doceavo bit conmuta. Este cambio es conservado dentro de la bandera pendiente T0PND, y ocurre cada 4.096 ms en la frecuencia de reloj máxima (t_C= 1 µs). Una bandera de control T0EN acepta la interrupción del bit doceavo del temporizador T0 para ser habilitado o deshabilitado. Estableciendo T0EN se habilita la interrupción, mientras que restableciendo éste se deshabilita.

4.6.2 Temporizador T1

Una de las principales funciones de un microcontrolador es el de proporcionar la posibilidad de ser temporizador y contador, para tareas de control en tiempo real La familia del COP8 cuenta con una muy versátil estructura temporizador y contador de 16-bit, y dos registros auto-carga y captura (R1A y R1B), optimizados para reducir el peso del software en aplicaciones en tiempo real. El bloque de reloj tiene dos pines asociados con ellos, T1A y T1B. El pin T1A soporta E/S requeridas por el bloque del reloj, mientras que el pin T1B es una entrada al bloque del reloj.

El bloque del reloj tiene tres modos de operación; PWM independiente del procesador, Contador de eventos externos y el de Captura de entradas.

Los bits de control TIC3, TIC2 u TIC1 permiten seleccionar los diferentes modos de operación.

- **Modo 1. PWM independiente del procesador**

En este modo el temporizador genera una señal PWM independiente del procesador una vez que el temporizador esta instalado, ninguna acción adicional es requerida para la CPU la cual lo traduce en costos menores de software y mayor desempeño. El usuario del software utiliza el bloque del temporizador solamente cuando los parámetros PWM requieren actualizarse
El temporizador puede generar la salida PWM con el ancho y ciclo de trabajo controlada por los valores almacenados en el registro de recarga Los registros de recarga controlan los valores de la cuenta regresiva y los valores recargados son automáticamente escritos dentro del temporizador cuando su cuenta llega a 0, generando una interrupción en cada recarga. Bajo el control de software y con un costo mínimo, la salida PWM es útil para el control de motores, triac, la intensidad de display y en salidas proporcionadas para la adquisición de datos y generación de ondas sinusoidales.

En este modo, el temporizador T1 realiza una cuenta regresiva en un porcentaje fijo de t_C. En cada flujo bajo el temporizador es recargado alternando su contenido con los registros de soporte, R1A y R1B. El primer flujo bajo del temporizador causa la recarga tomando el registro R1A. Subsecuentes flujos bajos causan la recarga del temporizador tomando alternativamente los registros y comenzando con el registro R1B.

La figura 4.9 muestra un diagrama de bloques del temporizador en el modo PWM.

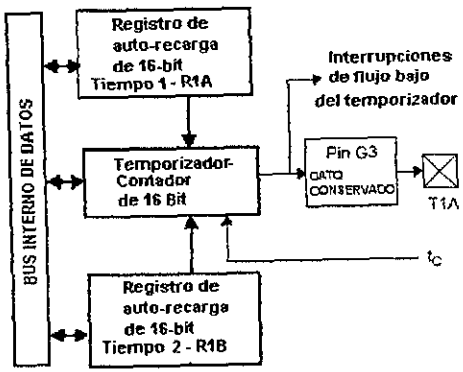


Figura 4. 9 Temporizador en el modo PWM.

El flujo bajo puede ser programado para conmutar el pin de salida T1A, además de también poder ser programado para generar interrupciones.

Flujos bajos d el temporizador son alternativamente conservados en dos banderas que quedan pendientes, T1PDNA y T1PDNB. Se pueden restablecer estas banderas bajo el control de software. Dos banderas habilitadas de control, T1ENA y T2ENB, permiten que las interrupciones del temporizador sean habilitadas o deshabilitadas. Establecer la bandera T1ENA causa una interrupción cuando un flujo bajo causa que el registro R1A sea recargado en el temporizador. El establecer la bandera T1ENB causa una interrupción cuando un flujo bajo causa que el registro R1B sea recargado en el temporizador. Al reestablecer las banderas habilitadas se deshabilitan las interrupciones asociadas

Cualquiera o ambos de las interrupciones de los flujos bajos del temporizador pueden ser habilitados. Esto da al usuario flexibilidad de interrumpir por cada periodo PWM o cualquier flanco de subida o bajada de la salida PWM. Alternativamente, se puede escoger la interrupción para ambos flancos de la salida PWM.

- **Modo 2. Contador de eventos externos**

Este modo es similar al modo PWM independiente del procesador descrito antes. La principal diferencia es que el temporizador T1 es registrado por la señal de entrada del pin T1A. Los bits de control del temporizador T1, TIC3, TIC2 y TIC1 permiten registrar un flanco positivo o negativo del pin T1A. Flujos bajos del temporizador son conservados en la bandera T1PNDA. Al establecer la bandera T1ENA se provoca una interrupción cuando el temporizador tiene flujos bajos.

En este modo el pin de entrada T1B puede ser usado como un flanco positivo independiente, sensible a entradas de interrupción, si la bandera de control T1ENB es establecida. En el caso de un flanco positivo en el pin de entrada T1B es almacenado en la bandera TIPNDB.

La figura 4.10 muestra un diagrama de bloques del temporizador en el modo "Contador de eventos externos".

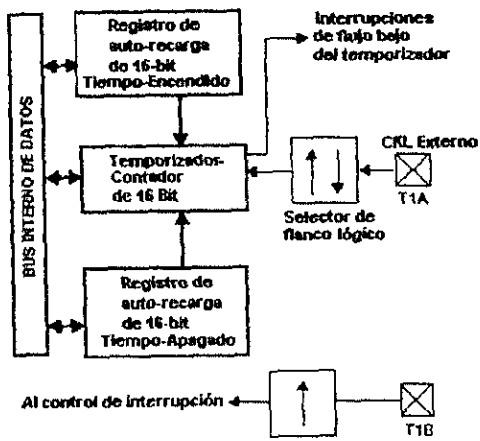


Figura 4. 10 Temporizador en el modo Contador de eventos externos.

- **Modo 3. Captura de entrada**

El dispositivo puede medir con precisión frecuencias externas o eventos externos de tiempo poniendo el bloque del temporizador, T1, en el modo de captura de entrada. En este modo, la recarga de registro sirve como un registro independiente de captura de entrada, capturando el contenido del temporizador cuando un evento externo ocurre (transición en el pin de entrada en el temporizador). El registro de captura puede ser leído mientras se mantiene el conteo, una característica que permite medir al tiempo transcurrido y el tiempo entre eventos. Al guardar el valor del temporizador cuando el evento externo ocurre, el tiempo del evento es recordado. La mayoría de los microcontroladores tienen un tiempo de recuperación de datos porque ellos no pueden determinar el valor del temporizador cuando un evento externo ocurre. El registro de captura elimina el tiempo de captura de datos, por esa razón permite al programa de la aplicación el recuperar el valor almacenado en el registro de captura.

En este modo, el temporizador T1 esta constantemente corriendo en un porcentaje fijo de t_c. Los dos registros R1A y R1B, actúan como registros de captura. Cada registro actúa en conjunción con un pin. El registro R1A actúa en conjunción con el pin T1A y el registro R1B actúa en conjunción con el pin T1B

El valor del temporizador obtenido es copiado en el registro cundo un evento es disparado en su correspondiente pin. Los bits de control, TIC3, TIC2 y TIC1, permiten al evento disparado ser específicamente un flanco positivo o negativo. Las condiciones de disparo para cada pin de entrada pueden ser especificadas independientemente.

Las condiciones de disparo pueden también ser programadas para generar interrupciones. El evento de la condición de disparo especificada en los pines T1A y T1B son respectivamente almacenados en las banderas pendientes, T1PNDA y T1PNDB. La bandera de control T1ENA permite la interrupción en T1A para ser habilitada o deshabilitada. Establecer la bandera T1ENA permite que las interrupciones sean generadas cuando la condición de disparo seleccionada ocurre en el pin T1A. Similarmente, la bandera T1ENB controla las interrupciones del pin T1B.

La Figura 4.11 muestra diagrama de bloques del modo de captura de entrada.

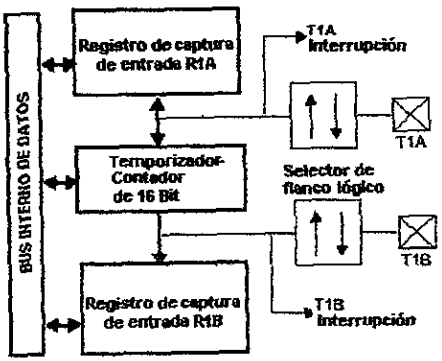


Figura 4. 11 Modo de captura de entrada del temporizador.

4.6.3 Banderas de control

Los bits de control y sus funciones se resumen a continuación:

T1C0	Control de Inicio/Termino en los Modos 1 y 2 (PWM independiente del procesador y Contador de eventos externos), donde 1 = Inicio, 0 = Termino Bandera de interrupción de bajo flujo del temporizador en modo 3 Captura de entrada)
T1PNDA	Bandera pendiente de interrupción
T1PNDB	Bandera pendiente de interrupción
T1ENA	Bandera de habilitación de interrupción
T1ENB	Bandera de habilitación de interrupción 1 = Habilitar interrupción 0 = Deshabilitar interrupción
T1C3	Control del modo del temporizador
T1C2	Control del modo del temporizador
T1C1	Control del modo del temporizador

T1C3	T1C2	T1C1	Modo-Temporizador	Fuente de Interrupción A	Fuente de Interrupción B	Contador
0	0	0	MODE 2 (Contador de eventos externos)	Temporizador Under-flow	Flanco Pos T1B	T1A Flanco Pos.
0	0	1	MODE 2 (Contador de eventos externos)	Temporizador Under-flow	Pos T1B Flanco	T1A Flanco Neg.
1	0	1	MODE 1 (PWM) T1A Tog-gle	Auto-carga RA	Auto-carga RB	t c
1	0	0	MODE 1 (PWM) No T1Atoggle	Auto-carga RA	Auto-carga RB	t c
0	1	0	MODE 3 (Captura) Capturas T1A Flanco Pos T1B Flanco Pos	Flanco Pos. T1A o Temporizador Underflow	Flanco Pos. T1B	t c
1	1	0	MODE 3 (Captura) Capturas· T1A Flanco Pos T1B Flanco Neg	Flanco Pos T1A o Temporizador Under-flow	Flanco Neg. T1B	t c
0	1	1	MODE 3 (Captura) Capturas: T1A Flanco Neg. T1B Flanco Pos.	Neg. T1A Flanco or Temporizador Underflow	Flanco Pos. T1B	t c
1	1	1	MODE 3 (Captura) Capturas: T1A Neg Flanco T1B Neg. Flanco	Flanco T1A Neg. o Temporizador Underflow	Flanco Neg T1B	t c

4.7 Características de bajo consumo

Hoy en dia la proliferación de las aplicaciones que utilizan baterías tienen puestas nuevas demandas sobre los diseñadores para manejar bajos consumos de energía Los sistemas operados con baterías no son los únicos tipos de aplicaciones que demandan baja potencia. Las restricciones de presupuesto son impuestas también en aquellas aplicaciones industriales donde regulaciones adecuadas y altos costos de fuentes de voltaje no pueden ser tolerados

El COP8SAx7 ofrece a los diseñadores de sistemas una variedad de características de bajo consumo de energía que les permite satisfacer los requerimientos de demanda actuales de las aplicaciones de bajo consumo de energía. Estas características incluyen operaciones de bajo voltaje, bajo drenaje de corriente, y características de bajo consumo como HALT, IDLE y activación en múltiples entradas(MIWU- Multi-Input Wakeup).

El dispositivo ofrece dos modos de bajo consumo de operación HALT e IDLE. En el modo HALT, todas las actividades del microcontrolador son detenidas. En el modo IDLE, el circuito oscilador interno y el temporizador T0 están activos pero las demás actividades del microcontrolador son detenidas. En cualquiera de los modos, la memoria RAM, los registros, estados de E/S, y los temporizadores (excepto T0) permanecen inalterados.

El monitor de reloj si esta habilitado puede ser activado en ambos modos.

4.7.1 Modo HALT

El dispositivo puede ser puesto en el modo HALT escribiendo un "1" en la bandera HALT (bit de dato G7). Todas las actividades del microcontrolador, incluidos el reloj y los temporizadores, son detenidos. El WATCHDOG lógico es deshabilitado durante el modo HALT. Sin embargo. El circuito del monitor de reloj, esta habilitado, permaneciendo activo y causando un valor bajo en el pin de salida(WDOUT) del WATCHDOG. Si el modo HALT es usado y no se desea activar el pin WDOUT, el monitor del reloj debe deshabilitarse. En este modo los requerimientos de voltaje (Vcc) son disminuidos a Vr (Vr = 2.0V) sin alterar el estado del dispositivo.

El dispositivo soporta tres diferentes formas de salir del modo HALT. El primer método es con la característica de Activación Multi-Entrada en el puerto I. El segundo método es con una transición de bajo a alto en el pin CKO (G7). Este método imposibilita el uso de la configuración del reloj de cristal (porque CKO esta dedicado a la salida), y solamente puede ser usado con una configuración de reloj externo o R/C. El tercer método de salida del modo HALT es aplicar un valor lógico bajo al pin $\overline{\text{RESET}}$.

Puesto que un resonador de cerámica o cristal puede ser seleccionado como el oscilador, la señal de activación no es permitida para que el chip trabaje inmediatamente dado que el resonador

tiene un tiempo de retardo para alcanzar la amplitud completa y la estabilidad de la frecuencia. El temporizador IDLE es usado para generar un retardo fijo para asegurar que el oscilador efectivamente tiene estabilidad permitiendo la ejecución de la instrucción. En este caso, al detectar una señal valida de activación, únicamente el circuito del oscilador es habilitado. El temporizador IDLE es cargado con un valor de 256 y es registrado con la instrucción de ciclo de reloj t_C. El valor t_C se origina de la división de la caída del oscilador por un factor de 10. El siguiente Schmitt trigger del CKI en el chip asegura que el temporizador IDLE es registrado solamente cuando el oscilador tiene suficiente amplitud para tener las especificaciones del Schmitt trigger. Al término de la activación, del temporizador IDLE, habilita las señales de reloj para que sean enviadas al resto del chip.

La figura 4.12 muestra la activación mediante el modo HALT.

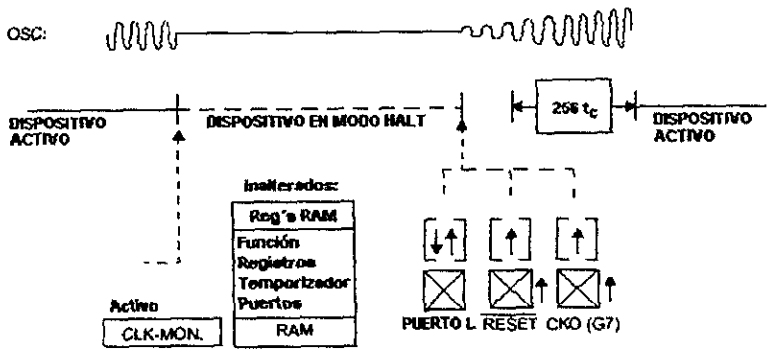


Figura 4. 12 Activación en modo HALT.

Si una opción de reloj R/C es usada, el retardo fijo es introducido opcionalmente. Un control CLKDLY, es mapeado como bit de configuración G7, controla tanto como si el retardo es introducido o no. El retardo es incluido si CLKDLY es establecido, y excluido si CLKDLY es re-inicializado. El bit CLKDLY es limpiado en la re-inicialización.

El dispositivo tienen dos opciones asociadas con el modo HALT. La primera establece la característica propia del modo HALT, mientras que la segunda deshabilita el modo HALT seleccionado a través del bit 0 del registro ECON. Con la opción habilitada del modo HALT, el

dispositivo puede entrar y salir del modo HALT como se describe adelante. Con la opción deshabilitada, el dispositivo no puede ser puesto en el modo HALT (escribir un "1" en la bandera HALT no tendrá efecto, la bandera permanece en "0").

El circuito detector WATCHDOG es inhibido durante el modo HALT. Sin embargo, el circuito del monitor de reloj permanece activo durante el modo HALT a fin de asegurar un error en el monitor de reloj si el dispositivo inadvertidamente registra el modo HALT como resultado de un programa fuera de control o una falla de energía imprevista.

Si el dispositivo es puesto en el modo HALT, con el oscilador R/C seleccionado, el pin de entrada del reloj (CKI) es forzado a un valor interno "alto". Con el oscilador de cristal o externo, el pin CKI queda establecido como de tercer-estado.

4.7.2 Modo IDLE

El dispositivo es puesto a modo IDLE por la escritura de un "1" en la bandera IDLE (bit de dato G6). En este modo todas las actividades excepto el circuito asociado del oscilador interno y el temporizador T0, son detenidas.

Así como en el modo HALT, el dispositivo puede regresar a la operación normal con una re-inicialización, o con la activación de entrada-múltiple del puerto L. Alternativamente el microcontrolador reanuda la operación normal de el modo IDLE cuando el bit doceavo de el temporizador IDLE conmuta.

Esta condición del bit doceavo del temporizador IDLE es guardada en la bandera pendiente T0PND.

Se cuenta con la opción de comenzar la interrupción con una transición en el bit doceavo del temporizador T0. La interrupción puede ser habilitada o deshabilitada vía el bit de control T0EN. Estableciendo la bandera T0EN se habilita la interrupción y viceversa.

Se puede entrar al modo IDLE con la habilitación de la interrupción de temporizador T0. En este caso, cuando el bit T0PND llega a establecerse, el dispositivo primero ejecuta la rutina de

servicio de interrupción de T0 y entonces regresa a la instrucción siguiente, la instrucción "Enter Idle Mode" (entrar al modo IDLE).

La figura 4 13 muestra la activación desde el modo IDLE.

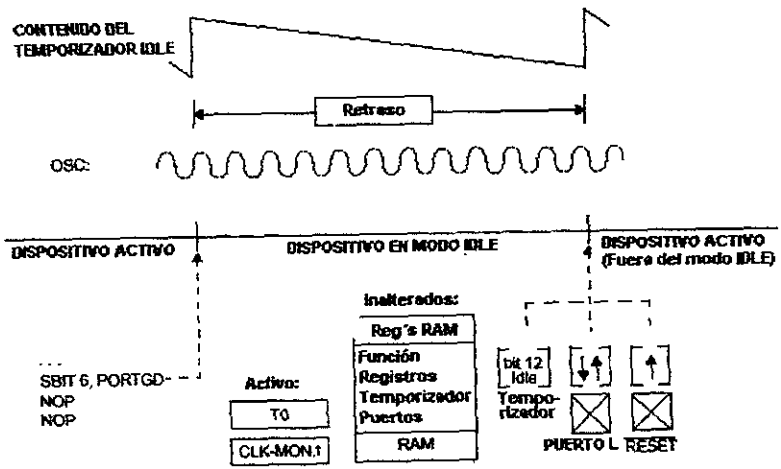


Figura 4.13 Activación en modo IDLE.

4.7.3 Activación Multi-entradas

La característica de Activación Multi-entradas es usada para regresar (activar) el dispositivo de cualquier modo HALT e IDLE. Alternativamente esta característuca puede también ser usada para generar hasta 8 flancos seleccionables de interrupciones externas.

La figura 4.14 muestra la lógica de Activación Multi-entradas

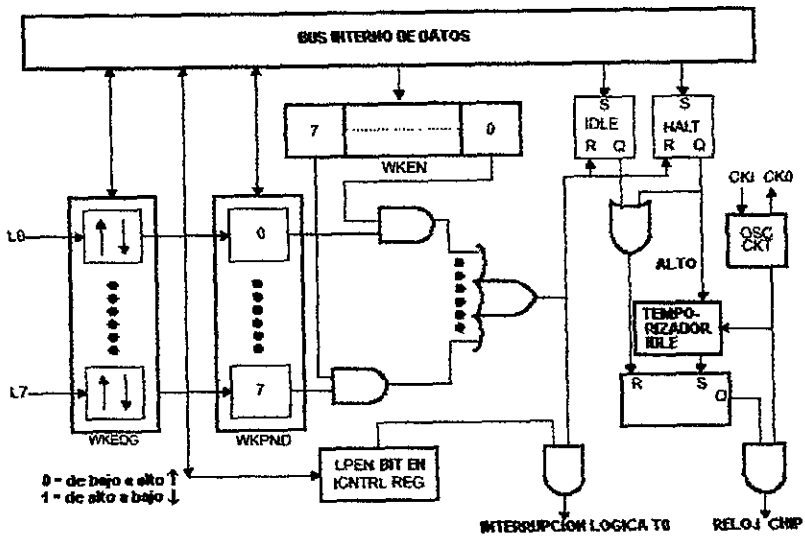


Figura 4. 14 Lógica de Activación Multi-entradas

La Activación Multi-entradas (MIWU por sus siglas en ingles) permite a los diseñadores especificar cualquiera de los 8 pines del controlador necesitados para activar el proceso de instrucciones. Esta capacidad es habilitada durante una transición en el pin de entrada – de bajo a alto, o de alto a bajo – el cual es recordado internamente en los registros del chip. Sin la activación multi-entradas, el microcontrolador requerirá guardar su software y código de operación constantemente (ver la Figura 4.15).

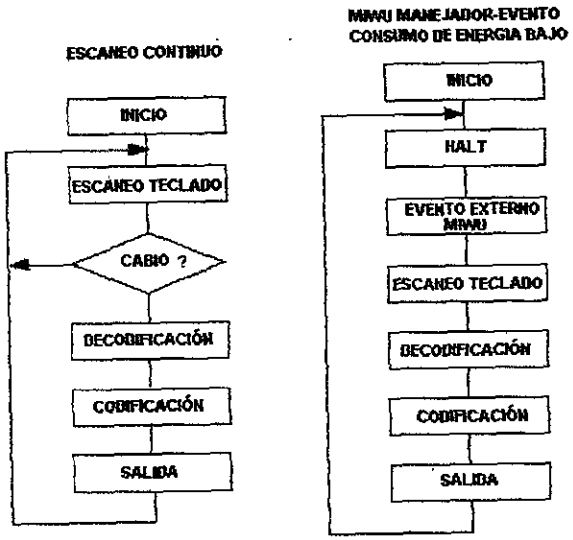


Figura 4. 15 Escaneo de teclado.

MIWU normalmente es usado para reducir el consumo de corriente bajando a menos de 4μA durante el estado inactivo comparado con los 6μA de consumo de corriente si el código permanece corriendo. Además MIWU hace que el diseño de hardware sea fácil y más eficiente al reducir la cantidad de componentes.

Tómese por ejemplo la aplicación del teclado de una laptop, el puerto L es usado como la entrada de líneas del teclado.

El microcontrolador esta en modo HALT solamente el presionar una tecla provocando una transición de alto a bajo en una de las líneas, obligando al dispositivo a salir del modo HALT. El teclado es escaneado para detectar cual fue la tecla presionada y el código apropiado es enviado a la computadora. Cuado el escaneo es completado, el microcontrolador es conmutado de regreso al modo HALT (ver figura 4 15)

Con características tales como MIWU, modo HALT e IDLE, voltaje bajo y posibilidad de drenaje de corriente, la familia de microcontroladores COP8SAx7 de National realiza su parte para

reducir el consumo de energía. Y con la tendencia "actual" apunta a un medio donde menos es efectivamente más, regresa el mando a las manos del diseñador de sistemas de control de la actualidad.

La característica MIWU utiliza el puerto L. Se puede seleccionar cual bit del puerto L en particular (o una combinación de bits del puerto L) causará que el dispositivo salga del modo HALT o IDLE. La selección es echa a través del registro WKEN. El registro WKEN es un registro de lectura – escritura de 8-bit; el cual contiene un bit de control para cada bit del puerto L. Estableciendo un bit WKEN en particular habilita una activación del pin asociado del puerto L.

4.8 Interrupciones

El dispositivo soporta ocho vectores de interrupción. Las fuentes de interrupción incluyen Temporizador 1, Temporizador T0, Puerto L, Activación, Software de diagnostico, MICROWIRE /PLUS y salida externa.

Todas las interrupciones fuerzan una bifurcación a la localidad hexadecimal 00FF en la memoria de programa. La instrucción VIS puede ser usada por un vector para el servicio de rutina de la localidad 00FF Hex.

El software de diagnostico tienen una prioridad alta mientras VIS por defecto tiene una prioridad baja.

Cada una de las seis entradas enmascarables tienen fijo un grado arbitrario y un vector. La figura 4.16 muestra el diagrama de bloques de Interrupción

simultáneamente, la interrupción con una mayor prioridad será atendida primero, y las otras condiciones pendientes quedaran en espera.

Al reinicializar, todas los bits pendientes, bits habilitados individualmente y el bit GIE son puestos a cero. Así una condición de interrupción enmascarada no puede disparar una interrupción hasta que el programa la habilite estableciendo el bit GIE y el bit individual. Cuando se habilita una interrupción, se puede considerar o no una activación del bit hasta que el bit sea reconocido. Si, en el momento una interrupción es habilitada, cualquier ocurrencia previa de la interrupción se ignorará, el bit pendiente asociado requerirá ser puesto a cero previamente para habilitar la interrupción. De otra manera, la interrupción simplemente será habilitada; sí el bit pendiente se a establecido, éste dispara inmediatamente una interrupción. Una interrupción enmascarada es activada sí sus bits de habilitación y pendiente son establecidos.

Una interrupción es un evento asincrono el cual ocurre antes, durante o después de un ciclo de instrucción. Cualquier interrupción que ocurre durante la ejecución de una instrucción no es reconocida hasta el inicio normal de la siguiente instrucción ejecutada siendo ésta omitida (saltada), el salto es realizado antes de que la interrupción pendiente sea reconocida.

Al comienzo del reconocimiento de la interrupción, las siguientes acciones ocurren:

1. El bit GIE es automáticamente puesto a cero, previniendo cualquier subsiguiente interrupción enmascarada de la rutina actual de servicio de interrupciones. Esta característica previene una interrupción enmascarada de cualquier otra interrupción que este en servicio.
2. La dirección de la instrucción próxima a ser ejecutada es colocada en la pila de información.
3. El contador de programa (PC) es cargado en 00FF Hex, causando un salto a aquella localidad de la memoria de programa.

El dispositivo requiere siete ciclos de instrucción para realizar las acciones anteriores.

Si se desea permitir interrupciones anidadas, la rutina del servicio de interrupciones fija el bit GIE a 1 para escribir el registro PSW, y así permitir otras interrupciones enmascaradas para interrumpir la rutina de servicio actual. Si las interrupciones anidadas están permitidas, se debe tener precaución. Se debe escribir el programa en el cual se prevea un posible overflow (desbordamiento) en la pila de información, resultando en la perdida de información.

La rutina de servicio de interrupción almacenada en la localidad 00FF Hex usa la instrucción VSI para determinar la causa de la interrupción, y salta a la rutina del manejo de interrupción correspondiente para habilitar la de una mayor prioridad y activar la interrupción. Alternativamente, se puede escoger el sondear todas las interrupciones pendientes y habilitar los bits para determinar el o los orígenes de la interrupción. Si más de una interrupción esta activada, el programa debe decidir cual interrupción atender.

Dentro de una rutina especifica de servicio de interrupción, el bit pendiente asociado deberá limpiarse. Esto se hace normalmente tan pronto como sea posible en la rutina de servicio que esta en funcionamiento, para evitar perder la siguiente ocurrencia del mismo tipo de evento. Así, si el mismo evento ocurre una segunda vez, aún mientras la primera ocurrencia esta todavía en servicio, la segunda ocurrencia será atendida inmediatamente una vez que se regrese de la rutina de interrupción actual.

Una rutina de servicio de interrupción normalmente necesita terminar con una instrucción RETI. Esta instrucción fija el bit GIE a 1, salta a la dirección almacenada en la pila de información, y restaura la dirección del contador de programa. La ejecución del programa entonces procede con la siguiente instrucción que deberá ejecutarse si no hubiera sido interrumpida. Si hay alguna interrupción pendiente, la interrupción de más alta prioridad es atendida inmediatamente al regreso de la interrupción previa.

4.8.2 Instrucción VIS

La rutina de servicio de interrupción general, la cual empieza en la dirección 00FF Hex, debe ser capaz de manejar todos los tipos de interrupción. La instrucción VIS, junto con una tabla de vectores de interrupción, le dice al dispositivo la rutina especifica que manejara la interrupción basándose en la causa de la interrupción.

VSI es una instrucción de un byte, normalmente en el principio de la rutina de servicio de interrupción en la dirección 00FF Hex, o enseguida de ése punto, inmediatamente después del código usado para el cambio del contexto. La instrucción VIS determina cual interrupción habilitada y pendiente tiene mayor prioridad y causa un salto indirecto a la dirección correspondiente a aquella fuente de interrupción. El salto de direcciones (vectores) para todas las posibles fuentes de interrupción es almacenado en una tabla de vectores.

La tabla de vectores puede tener una longitud de 32 bytes (16 vectores como máximo) y reside en la parte alta del bloque de 256 bytes que contienen a la instrucción VIS. Sin embargo, la instrucción VIS esta en la parte más alta del bloque (tal como en 00FF Hex), la tabla del vector reside en la parte alta de los siguientes bytes del bloque. Así, si la instrucción VIS se encuentra en algún lugar entre 00FF y 01DF Hex (el caso normal), la tabla del vector se encuentra en las direcciones 01E0 y 01FF Hex. Si la instrucción se encuentra entre 01FF y 02DF Hex, entonces la tabla del vector esta localizada entre las direcciones 02E0 y 02FF Hex, así sucesivamente.

Cada vector tiene 15 bit de tamaño y apunta al comienzo de una rutina de servicio de interrupción especifica en alguna parte de los 32 Kbytes del espacio de memoria. Cada vector ocupa dos bytes de la tabla de vector, con el byte de orden más alto en las direcciones inferiores. Los vectores son organizados en orden de mayor prioridad de interrupción. El vector de interrupción enmascarada con el orden más bajo es colocado en 0yE0 (byte de orden más alto) y 0yE1 (byte con orden más bajo). La siguiente interrupción en orden de prioridad es colocada en 0yE2 y 0yE3, y a si en adelante al ir incrementando el grado El software de diagnostico tienen el margen más alto y su vector esta siempre localizado en 0yFE y 0yFF. El número de interrupciones que pueden ser activadas define el tamaño de la tabla.

La tabla siguiente muestra los tipos de interrupción, la clasificación arbitraria de la interrupción y la localidad del vector correspondiente en la tabla de vectores.

Clasificación Arbitraria	Fuente	Descripción	Dirección de vectores (Byte Alto-Bajo)
(1) Más alto	Software	Instrucción INTR	0yFE - 0yFF
(2)	Reservado	Futuro	0yFC - 0yFD
(3)	Externo	G0	0yFA - 0yFB
(4)	Temporizador T0	Desbordamiento bajo	0yF8 - 0yF9
(5)	Temporizador T1	T1A/ Desbordamiento bajo	0yF6 - 0yF7
(6)	Temporizador T1	T1B	0yF4 - 0yF5
(7)	MICROWIRE/PLUS	Activo bajo	0yF2 - 0yF3
(8)	Reservado	Futuro	0yF0 - 0yF1
(9)	Reservado	Futuro	0yEE - 0yEF
(10)	Reservado	Futuro	0yEC - 0yED
(11)	Reservado	Futuro	0yEA - 0yEB
(12)	Reservado	Futuro	0yE8 - 0yE9
(13)	Reservado	Futuro	0yE6 - 0yE7
(14)	Reservado	V	0yE4 - 0yE5
(15)	PuertoL/Wakeup	Puerto L Edge	0yE2 - 0yE3
(16) Más bajo	Por defecto	Instrucción VIS ejecutada sin ninguna interrupción	0yE0 - 0yE1

*y es una variable que representa el bloque VIS y la tabla de vector debe localizarse en el mismo bloque 256-byte excepto si VIS esta localizado en la última dirección de un bloque. En este caso, la tabla debe estar en el bloque siguiente.

La tabla de vector es llenada con las localidades de memoria de las rutinas de servicio de vectores especificas. Por ejemplo, la rutina del Software de diagnostico esta localizada en 0310 Hex, entonces el vector localizado en 0yFE y –0yFF contienen el dato 03 y 10 Hex respectivamente. Cuando una interrupción de Software de diagnostico ocurre y la instrucción VIS es ejecutada el programa salta a la dirección especificada por la tabla de vector.

La fuente de interrupción esta listada en orden jerárquico en la tabla de vectores, de la de mayor a la de menor prioridad. Si dos o más interrupciones habilitadas y pendientes son detectadas al mismo tiempo, la primera con mayor grado de prioridad es atendida primero. Al regresar de la rutina de servicio de interrupción, la siguiente interrupción pendiente con el más alto grado de prioridad es atendida.

Ejecución de VIS

Cuando la instrucción VIS es ejecutada activa la lógica arbitraria. La lógica arbitraria genera un número impar entre E0 y FE (E0, E2, E4, E6,...,etc) dependiendo de cual interrupción activa tiene el grado arbitrario más alto al momento del 1er. ciclo de la ejecución de VIS. Por ejemplo, si el Software de diagnostico esta activo, FF es generado. Si la interrupción externa esta activa y el Software de diagnostico no lo esta, entonces E0 es generado. Este número remplaza el byte más pequeño del PC. El byte superior del PC permanece sin cambio. El nuevo PC es en consecuencia dirigido al vector de la interrupción activa con el grado arbitrario más alto. Este vector es leído de la memoria de programa y puesto en el PC el cual esta ahora apuntando a la 1ra. Instrucción de la rutina de servicio de la interrupción activa con el grado arbitrario más alto.

La figura 4.17 muestra los diferentes pasos realizados por la instrucción VIS.

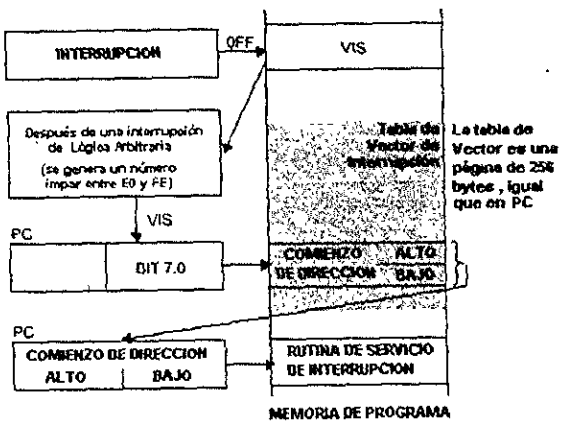


Figura 4. 17 Operación de VIS

La figura 4.18 muestra un diagrama de flujo para la instrucción VIS.

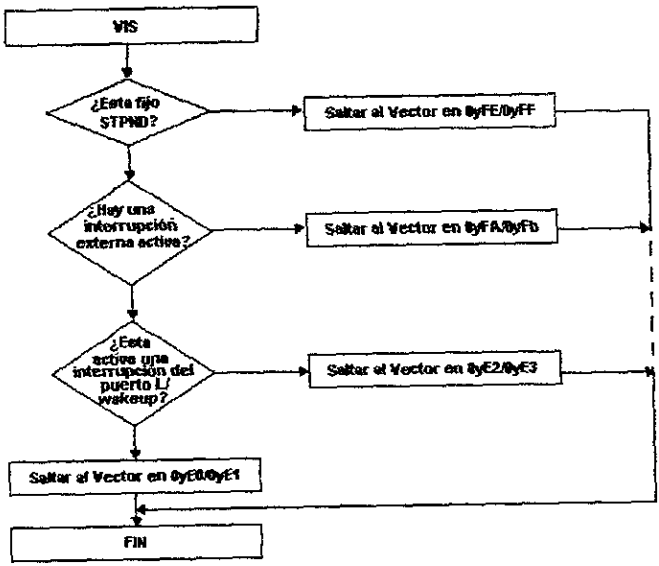


Figura 4. 18 Diagrama de flujo de VIS.

La bandera pendiente de la interrupción no enmascarada es deshabilitada por la instrucción RPND (Reinicializar Bit pendiente de Interrupción no enmascarada) y por RESET.

4.8.3 Interrupción no enmascarada

Bandera pendiente

Este es un bit de bandera pendiente asociado con la interrupción no enmascarada, llamado STPND. Esta bandera no esta mapeada en la memoria y no puede ser accedida directamente por el software

La bandera pendiente es puesta a cero cuando un dispositivo de Reset tiene efecto. Cuando la interrupción no enmascarada ocurre, el bit pendiente asociado es puesto a 1. La rutina de servicio de interrupción contiene una instrucción RPND para poner la bandera pendiente a cero. La instrucción RPND siempre restablece la bandera STPND.

Software de diagnostico

El software de diagnostico es un tipo especial de interrupción no enmascarada la cual ocurre cuando la instrucción INTR (usada para reconocer la interrupción) es llamada por la memoria de programa y colocada en el registro de instrucción. Esto puede ser de diversas formas, y normalmente a causa de una condición de error. Algunos casos se comentan a continuación;

Si el contador de programa (PC) apunta incorrectamente a una localidad de memoria más allá del espacio de memoria disponible, la memoria no existente o sin uso regresa ceros lo cual se interpreta como la instrucción INTR.

Si la pila es desbordada más allá del límite permitido (direcciones 02F y 06F Hex), una interrupción (Software de diagnostico) es disparada.

Un software de diagnostico puede ser disparado por una condición de hardware temporal tal como una baja de tensión o una falla imprevista.

El Software de diagnostico tiene la prioridad más alta de todas las interrupciones. Cuando un Software de diagnostico ocurre, el bit STPND es establecido. El bit GIE no es afectado y el bit pendiente (no accesible por el usuario) es usado para inhibir otras interrupciones y para dirigir el programa a la rutina de servicio ST, por medio de la instrucción VIS. Nada puede interrumpir la rutina de servicio Software de diagnostico excepto otra interrupción de Software de diagnostico. El STPND puede ser restablecido solamente por la instrucción RPND o un Reset en el chip

El software de diagnostico indica una condición inusual o desconocida de error Normalmente, regresa a la ejecución normal en el punto donde ocurrió el Software de diagnostico sin hacerlo con plena seguridad. Por lo tanto, la rutina de servicio de Software de diagnostico re-inicializará el puntero de la pila y realiza un procedimiento de recuperación que re-inicia el software en algún punto conocido, similar a un dispositivo de Reset, pero no necesariamente realizando todas las mismas funciones como dicho dispositivo. La rutina puede también ejecutar la instrucción RPND para re-inicializar la bandera. Por otra parte, todas las demás interrupciones deben ser cerradas. Para extender las posibilidades, la rutina de servicio de interrupción debe grabarse o indicar el contexto del dispositivo para que la causa del Software de diagnostico sea determinada.

Si se desea regresar a la ejecución normal a partir del punto en el cual fue disparado el Software de diagnostico, se tiene que ejecutar primero RPND, seguido de RETSK antes que RETI o RET. Esto es debido a que la dirección devuelta (almacenada en la pila) es de la instrucción INTR que dispara interrupción. El programa debe omitir primero esa instrucción para proceder con la siguiente. Por lo tanto, un ciclo infinito del Software de diagnostico y direcciones devueltas pueden presentarse.

El programar un regreso a la ejecución normal requieren consideraciones cuidadosas. Si la rutina de Software de diagnostico es interrumpida por otra interrupción del mismo tipo, la instrucción RPND en la rutina de servicio para el segundo Software de diagnostico re-inicializará la bandera STPND; al regreso de la primera rutina del Software de diagnostico, la bandera STPND tendrá erróneo su estado. Esto permite que las interrupciones enmascaradas sean reconocidas durante el primer Software de diagnostico. Para evitar problemas tales como éstos, el programador debe incluir la rutina del Software de diagnostico para permitir un procedimiento de recuperación más que un regreso a la ejecución normal.

4.8.4 Interrupción del puerto L

El puerto L da adicionalmente ocho interrupciones completas y sensibles a flancos, las cuales son localizadas en la misma subrutina.

La interrupción del puerto L comparte lógica con el circuito de activación (wake up). El registro WKEN permite interrupciones del puerto L que son individualmente habilitadas o deshabilitadas. El registro WKEDG especifica las condiciones de disparo par cualquier flanco positivo o negativo. Finalmente, el registro WKPND se conserva en las condiciones de disparo pendientes

Una bandera de control, LPEN, funciona como una habilitación de interrupción global (GIE) para interrupciones del puerto L. Establecer la bandera LPEN permite habilitar interrupciones y viceversa. Una bandera global pendiente por separado no es necesitada por que el registro WKPND es suficiente.

Ya que el puerto L es usado para activar el dispositivo de salida de los modos HALT e IDLE, se puede optar por salir del modo HALT o IDLE con o sin la interrupción habilitada. Si se elige deshabilitar la interrupción, entonces el dispositivo re-iniciará la ejecución de la instrucción inmediatamente siguiendo la instrucción que puso al microcontrolador en el modo HALT o IDLE. En el otro caso, el dispositivo primero ejecuta la rutina de servicio de interrupción y entonces regresar a su operación normal.

4.8.5 Resumen de Interrupciones

El dispositivo emplea los siguientes tipos de interrupciones, listadas a continuación en orden de prioridad:

1. La interrupción de software de diagnostico no enmascarada, es disparada por la instrucción INTR (código de operación 00). El software de diagnostico es reconocido inmediatamente. Esta rutina de servicio de interrupción puede ser interrumpida solamente por otra rutina de Software de interrupción. El Software de diagnostico finaliza con las dos instrucciones RPND seguidas por un procedimiento de re-inicialización
2. Las interrupciones enmascaradas, disparadas por un bloque periférico interno o un dispositivo externo conectado al chip. Bajo condiciones normales, una interrupción enmascarada no interrumpirá ningún otra rutina de interrupción en progreso. Una rutina de interrupción no enmascarada en progreso puede ser interrumpida por la petición de una interrupción no enmascarada. Una rutina de interrupción enmascarada puede terminar con una instrucción RETI.

4.9 Monitores WATCHDOG y de reloj

El dispositivo contiene un monitor WATCHDOG y de reloj que es seleccionado por el usuario La siguiente sección aplica únicamente si la opción de WATCHDOG a sido seleccionada en el registro ECON. El WATCHDOG esta diseñado para detectar que el programa del usuario presenta un ciclo infinito, resultando en la perdida del control del programa o programas descontrolados.

La lógica del WATCHDOG presenta dos ventanas de servicio independientes. Mientras el usuario programa la ventana superior escogiendo el tiempo de servicio del Watchdog, la ventana

inferior da protección contra un ciclo de programa infinito que contiene la instrucción de servicio del watchdog.

El COP8SAx7 incluye un programa de trampa (software trap) que da protección contra saturación de la pila y localidades de direccionamiento fuera de valores validos del espacio del programa.

El monitor de reloj es usado para detectar la ausencia de un reloj o de un reloj muy lento que este debajo del porcentaje especificado en el pin CKI.

El WATCHDOG consiste en dos bloques lógicos independientes: WD UPPER y WD LOWER. El primero establece el límite superior y el segundo define el límite inferior en la ventana de servicio.

El servicio de WATCHDOG consiste en escribir un valor especifico en el registro llamado WDSVR el cual esta mapeado en la memoria RAM. Este valor se compone de tres campos: un Window Select (selector de ventana) de 2-bit, un campo Key Data (dato reservado) de 5-bit, y el campo Clock Monitor Select (Selecctor del monitor de reloj) de 1-bit. En la tabla siguiente se muestra el registro WDSVR.

Window Select		**Key Data**					**Clock Monitor**
X	X	**0**	**1**	**1**	**0**	**0**	**Y**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**

El límite inferior de la ventana de servicio esta fijado en 256 ciclos de instrucción. Los bits 7 y 6 del registro WDSVR permiten escoger un límite superior de la ventana de servicio

La tabla de abajo muestra las cuatro posibles combinaciones de los límites superior e inferior para la ventana de servicio del WATCHDOG. La flexibilidad de escoger la ventana de servicio evita cualquier sobrecarga indebida en el software del usuario.

Los bits 5,4,3,2 y 1 del registro WDSVR representan los 5-bit del campo Key Data. Este dato de reserva esta fijado en 01100. Por último el bit 0 de este registro es el bit del Clock Monitor Select.

WDSVR Bit 7	WDSVR Bit 6	Service Window (Límite Inferior-Superior)
0	0	256–8k t c Ciclos
0	1	256–16k t c Ciclos
1	0	256–32k t c Ciclos
1	1	256–64k t c Ciclos

4.9.1 Monitor de reloj

El monitor de reloj interno del dispositivo puede ser seleccionado o cancelado bajo el control del programa. El monitor de reloj garantuza no rechazar el reloj si el ciclo de instrucción de éste ($1/t_c$) es mayor o igual a 10KHz. Esto es equivalente a un porcentaje en la entrada de reloj sobre CKI mayor o igual a 100 KHz.

4.9.2 Resumen de WATCHDOG y del monitor de reloj.

Los siguientes son los puntos relevantes de WATCHDOG y del monitor de reloj, que deben ser tomado en cuenta:

- Los circuitos detectores de WATCHDOG y del monitor de reloj son inhibidos durante el RESET.
- Enseguida del RESET, el WATCHDOG y el monitor de reloj son habilitados, teniendo seleccionado el WATCHDOG la ventana de servicio más grande.
- La ventana de servicio de WATCHDOG y la opción de habilitar-deshabilitar del monitor de reloj pueden solamente ser cambiadas una vez, durante el servicio inicial de WATCHDOG enseguida de un RESET
- El servicio inicial de WATCHDOG debe corresponder al valor del dato reservado en el registro WDSVR a fin de evitar un error WATCHDOG.
- Los sucesivos servicios de WATCHDOG deben corresponder con los tres campos de datos en WDSVR a fin de evitar errores WATCHDOG

- El valor correcto del valor de dato reservado no puede ser leído del registro WDSVR. Cualquier intento para leer este valor de dato reservado de 01100 a partir de WDSVR leerá como valor de dato reservado solamente ceros.
- El circuito detector de WATCHDOG es inhibido tanto en el modo HALT como en el modo IDLE.
- El circuito del Monitor de Reloj es activado durante los modos HALT e IDLE. En consecuencia, el dispositivo inadvertidamente al ingresar el modo HALT será detectado como un error del monitor de reloj (suponiendo que la opción de habilitación del monitor de reloj haya sido seleccionada por el programa).
- Con la opción del oscilador R/C seleccionada y la restauración del bit CLKDLY, la ventana de servicio de WATCHDOG reasumirá el seguir en el modo HALT de donde fue apagado.
- Con la opción seleccionada del oscilador de cristal, o con la opción del oscilador R/C y el bit restaurado CLKDLY, la ventana de servicio WATCHDOG será puesta a su valor seleccionado de WDSVR próximo de HALT. Consecuentemente, el WATCHDOG no será servido al menos en el ciclo de instrucción 256 próximo de HALT, pero será atendido dentro de la ventana seleccionada para evitar un error WATCHDOG.
- El temporizador IDLE T0 no es inicializado con un RESET externo.
- El usuario puede sincronizar en el contador de ciclo IDLE con una interrupción del contador (T0) IDLE o por el monitoreo de la bandera T0PND. La bandera T0PND es fijada en cualquiera de los veinte bits del contador IDLE (cada 4096 ciclos de instrucción) El usuario es responsable de reestablecer la bandera T0PND.
- Un servicio WATCHDOG de hardware ocurre solamente hasta que el dispositivo salga del modo IDLE. Por lo tanto, el WATCHDOG no será servido por al menos en el ciclo de instrucción 256 próximo de HALT, pero será atendido dentro de la ventana seleccionada para evitar un error WATCHDOG.
- Al siguiente RESET, el servicio inicial de WATCHDOG (donde la ventana de servicio y el monitor de reloj deben ser seleccionados) puede ser programado en cualquier parte dentro de la ventana de servicio (65,536 ciclos de instrucción) inicializado por el RESET. Nótese que este servicio inicial WATCHDOG debe ser programado dentro de los primeros 256 ciclos de instrucción sin causar un error WATCHDOG.

4.9.3 Detección de Condiciones Ilegales

El dispositivo puede detectar varias condiciones ilegales obtenidas de errores de codificación, ruido transitorio, caídas en la fuente de voltaje, programas sin control, etc.

Al leer ROM no definida se obtienen ceros. El código de operación para la interrupción de software es 00. Si el programa hace llamadas a partir de la memoria ROM no definida, esto forzara a una interrupción de software, señalando de esta manera que una condición ilegal a ocurrido.

La subrutina de la pila disminuye para cada llamada (saltando a la subrutina), interrupción, o PUSH, y aumenta para cada regreso o POP. El puntero de la pila es inicializado a la localidad de RAM 06F Hex durante la re-inicialización. Por lo tanto, si hay más regresos que llamadas, el puntero de la pila apuntará a la dirección 070 y 071 Hex (la cual es indefinida RAM). Indefinida RAM de direcciones 070 a 07F, y todos los otros segmentos (p.e., Segmento 4... etc) son leídos como 1s, el cual al volver puede causar que el programa regrese a la dirección 7FFF Hex. Esta es una localidad indefinida ROM y la instrucción buscada (todos 0s) de esta localidad puede generar una interrupción de software señalando una condición ilegal.

Así, el chip puede detectar las siguientes condiciones ilegales:

1. Ejecución de una ROM indefinida.
2. Más POP en la pila al tomar más regresos que llamadas.

Cuando la interrupción de software ocurre, el usuario puede re-inicializar el puntero de la pila y hacer un procedimiento de recuperación antes de re-inicializar (esta recuperación del programa es probablemente similar para el próximo RESET, pero no contienen el mismo procedimiento de inicialización del programa). La recuperación del programa re-inicializará el bit de interrupción de software pendiente usando la instrucción RPND.

4.10 Interfase de entrada MICROWIRE/PLUS

La MICROWIRE/PLUS es una interfase de comunicaciones sincrona compatible, SPI serial Las capacidades de esta interfase habilitan al dispositivo para conectarse con MICROWIRE/PLUS

o periféricos SPI (p.e. convertidores A/D, manejadores de visualizadores, EEPROMs, etc) y con otros microcontroladores que puedan soportar estas mismas características. Esta consiste de un registro de desplazamiento serial de 8-bit (SIO) con entrada de datos serial (SI), salida de dato serial (SO) y reloj de desplazamiento seial (SK). La figura 4.19 muestra un diagrama de bloques de la lógica MICROWIRE/PLUS.

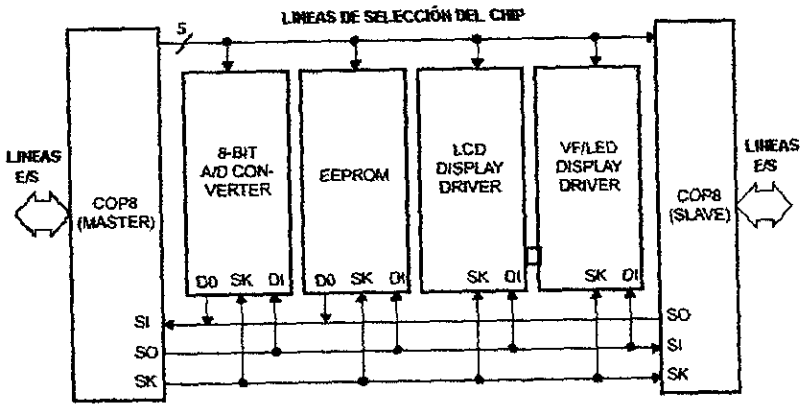


Figura 4. 19 Aplicación de MICROWIRE/PLUS.

El desplazamiento de reloj puede ser seleccionado de una fuente interna o externa. La operación del MICROWIRE/PLUS en conjunto con la fuente de reloj interna es llamada el modo maestro (Master) de operación. De forma similar la operación del MICROWIRE/PLUS en conjunto con un desplazamiento de reloj externo es llamada el modo de operación esclavo (Slave).

El registro CNTRL es usado para configurar y controlar el modo del MICROWIRE/PLUS Para usar el MICROWIRE/PLUS, el bit MSEL en el registro CNTRL es puesto a uno En el modo maestro, el índice del reloj SK es seleccionado por los dos bits, SL0 y SL1, en el registro CNTRL. La tabla siguiente detalla los diferentes índices de reloj que pueden seleccionarse:

SL1	SL0	periodo SK
0	0	2 X t c
0	1	4 X t c
1	x	8 X t c

Donde t c es la instrucción del ciclo de reloj

4.10.1 Operación de MICROEIRE/PLUS

Fijar el bit BUSY en el registro PSW causa que el MICROWIRE/PLUS comience a desplazar los datos. Este consigue un restablecimiento (reset) cuando los ocho bits de datos han sido desplazados El usuario puede restablecer el bit BUSY por software o permitiendo que los 8 bits sean desplazados. El dispositivo puede ingresar al modo como maestro o como esclavo. La figura 4.19 muestra como dos microcontroladores y algunos periféricos están interconectados haciendo uso del arreglo del MICROWIRE/PLUS.

Modo de operación MAESTRO.

En este modo de operación el desplazamiento de reloj (SK) es generado internamente. El MICROWARE Maestro permite inicializar todos los intercambios de datos. El bit MSEL en el registro CNTRL debe establecerse para habilitar las funciones SO y SK sobre el puerto G. Los pines SO y SK deben también ser seleccionados como salidas por la configuración apropiada de los bits en el registro de configuración del puerto G. En el modo Esclavo, el desplazamiento del reloj se detiene después de 8 pulsos de reloj La siguiente tabla resume la configuración del bit requerido para la operación del modo Maestro:

Esta tabla asume que la bandera de control MSEL esta establecida.

G4 (SO) Config. Bit	G5 (SK) Config. Bit	G4 Fun.	G5 Fun.	Operación
1	1	SO	Int. SK	MICROWIRE/PLUS Master
0	1	TRI-STATE	Int. SK	MICROWIRE/PLUS Master
1	0	SO	Ext. SK	MICROWIRE/PLUS Slave
0	0	TRI-STATE	Ext. SK	MICROWIRE/PLUS Slave

Modo de operación ESCLAVO.

En el modo de operación Esclavo el desplazamiento de reloj SK es generado por una fuente externa. La configuración del bit MSEL en el registro CNTRL habilita las funciones SO y SK sobre el puerto G. El pin SK debe seleccionarse como una entrada y el pin SO es seleccionado como una salida para la configuración y establecimiento apropiado de los bits en el registro de configuración del puerto G. La tabla anterior resume la configuración requerida para ingresar al modo de operación Esclavo.

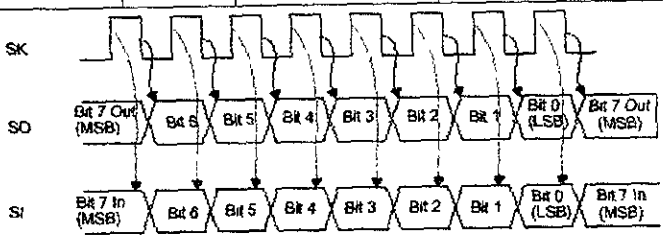
Operación de fase (Phase Operation) SK y Polaridad Inactiva (Idle Polarity) SK alternativas.

El dispositivo permite tanto el desplazamiento de reloj normal como un desplazamiento de reloj de fase alternativo para desplazar los datos fuera del registro SIO. En ambos modos la polaridad inactiva SK puede tener un valor alto o bajo. La polaridad es seleccionada por el bit 5 del registro de datos del puerto G. En el modo normal el dato es desplazado-dentro en el flanco de subida del reloj SK y el dato es desplazado-fuera en el flanco de bajada del reloj SK. El registro SIO es desplazado en cada flanco de bajada en el reloj SK. En la operación de fase SK alternativa, el dato es desplazado-dentro en el flanco de bajada del reloj SK y desplazado-fuera en el flanco de subida del reloj SK. El bit 6 del registro de configuración del puerto G selecciona el flanco SK.

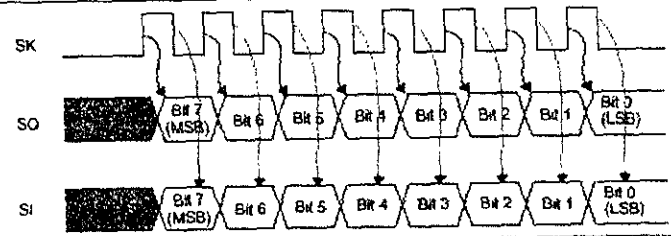
Una bandera de control SKSEL, permite que el modo normal o alternativo del reloj SK sea seleccionado. Reestablecer SKSEL provoca que la lógica MICROWIRE/PLUS sea cronometrada de la señal SK normal. Estableciendo la bandera SKSEL se selecciona el reloj SK alternativo. El SKSEL es mapeado dentro del bit de configuración G6. La bandera SKSEL puede activarse en la condición de restablecimiento de la señal SK normal.

La tabla siguiente muestra la polaridad del desplazamiento de reloj y fase de Muestreo - Desplazamiento

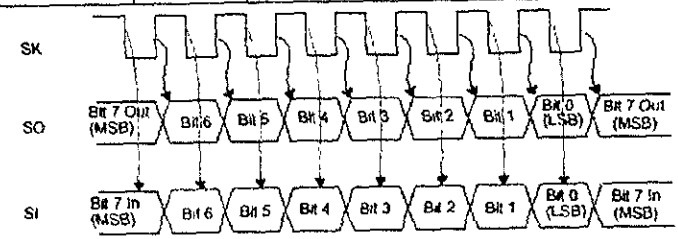
Fase SK	Puerto G		SO Cronometrado afuera en:	SI muestreado en:	Fase inactiva SK
	G6 (SKSEL) Config. Bit	G5 Bit-Dato			
Normal	0	0	Flanco de bajada SK	Flanco de subida SK	Bajo



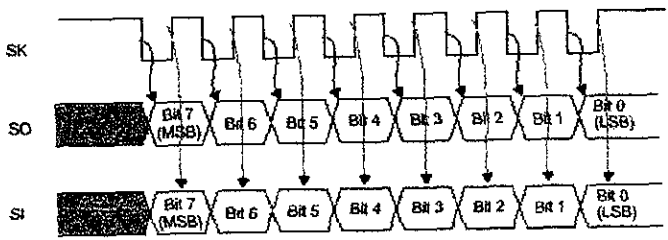
Alternativo	1	0	Flanco de subida SK	Flanco de bajada SK	Bajo



Alternativo	0	1	Flanco de subida SK	Flanco de bajada SK	Alto



Normal	1	1	Flanco de bajada SK	Flanco de subida SK	Alto



4.11 Mapa de memoria

Toda la memoria RAM, los puertos y registros (excepto A y PC) están mapeados en el espacio de direcciones de memoria de datos:

RAM	Direcciones ADD REG	Contenido
64 Bytes de RAM interna Seleccionada por el registro ECON	00 a 2F	RAM interna (48 Bytes)
	30 a 7F	RAM no utilizada (Leída como unos)
128 Bytes de RAM interna	00 a 6F	RAM interna (112 Bytes)
	70 a 7F	RAM no utilizada (Leída como unos)
	94	Puerto F Registro de Datos
	95	Puerto F Registro de Configuración
	96	Puerto F Pines de Entrada (solo lectura)
	97	Reservado
	C7	Registro de Servicio WATCHDOG (Reg. WDSVR)
	C8	MIWU Registro Selector de Flanco (Reg. WKEDG)

	C9	MIWU Registro de Habilitación (Reg. WKEN)
	CA	MIWU Registro Pendiente (Reg, WKPND)
	CB a CF	Reservado
	D0	Puerto L Registro de Datos
	D1	Puerto L Registro de Configuración
	D2	Puerto L Pines de Entrada (solo lectura)
	D3	Reservado
	D4	Puerto G Registro de Datos
	D5	Puerto G Registro de Configuración
	D6	Puerto G Pines de Entrada(solo lectura)
	D7	Reservado
	D8	Puerto C Registro de Datos
	D9	Puerto C Registro de Configuración
	DA	Puerto C Pines de Entrada(solo lectura)
	DB	Reservado
	DC	Puerto D
	DD a DF	Reservado
	E0 a E5	Reservado
	E6	Temporizador T1 Registro de Auto carga T1RB Byte Bajo
	E7	Temporizador T1 Registro de Auto carga T1RB Byte Alto
	E8	Registro ICNTRL

	E9	MICROWIRE/PLUS Registro de desplazamiento
	EA	Temporizador T1 Byte Bajo
	EB	Temporizador T1 Byte Alto
	EC	Temporizador T1 Registro de Auto carga T1RA Byte Bajo
	ED	Temporizador T1 Registro de Auto carga T1RA Byte Alto
	EE	Registro de control CNTRL
	EF	Registro PSW
	F0 a FB	RAM interna mapeada como Registro
	FC	Registro X
	FD	Registro SP
	FE	Registro B
	FF	Registro de Segmento

La lectura de cualquier localidad de memoria no definida en el rango de direcciones 0080H-00FFH regresa datos indefinidos

4.12 Resumen de Instrucciones

El conjunto de instrucciones cuenta con 58 diferentes instrucciones La mayor parte de las operaciones aritméticas, de comparación y de transferencia de datos (cargar, cambiar) operan con tres diferentes modos de direccionamiento (registro indirecto con el puntero B, memoria directa e inmediato) Estos diversos modos de direccionamiento incrementan el conjunto de instrucciones a un total de 87.

4.12.1 Modos de direccionamiento

El conjunto de instrucciones ofrece una variedad de métodos para especificar direcciones de memoria Cada método es llamado un modo de direccionamiento. Estos modos están clasificados en dos categorías. el modo de direccionamiento de operandos y el modo de direccionamiento de transferencia de control El primero es una serie de métodos de direccionamiento específico para

accesar (lectura o escritura) datos. El segundo modo es usado en conjunción con instrucciones de salto para controlar la secuencia de ejecución del programa de software.

Modos de direccionamiento de operandos

El operando de una instrucción indica que localidad de memoria es la afectada por esa instrucción. Varios modos de direccionamiento de operandos están disponibles, permitiendo que las localidades de memoria sean especificadas en diferentes formas. Una instrucción puede especificar una dirección directamente dando la dirección específica o indirectamente especificando el registro. El contenido del registro (o en algunos caso, dos registros) apunta a la localidad de memoria deseada. En el modo inmediato, el byte de dato usado esta contenido en la misma instrucción.

Cada modo de direccionamiento tiene sus propias ventajas y desventajas con respecto a flexibilidad, velocidad de ejecución y compactación de programación. No todos los modos están disponibles para todas las instrucciones. La instrucción Cargar (LD) presenta el mayor número de modos de direccionamiento.

Los modos de direccionamiento disponibles son

- **Directo**
  El direccionamiento de memoria es especificado como un byte en la instrucción. En lenguaje ensamblador, el direccionamiento directo es un valor numérico (o una etiqueta que ha sido definida en alguna parte del programa como valor numérico).

- **Registro Indirecto B o X**
  El direccionamiento de memoria es especificado por el contenido en el registro B o el registro X (registro apuntador) En lenguaje ensamblador, la notación [B] o [X] especifica cual registro sirve como apuntador.

- **Registro Indirecto B o X con Post-Incremento/Decremento**
  El direccionamiento de memoria relativa es especificado por el contenido del registro B o el registro X (registro apuntador). El registro apuntador es automáticamente incrementado o decrementado después de la ejecución, permitiendo una fácil manipulación de los

bloques de memoria con ciclos de software. En lenguaje ensamblador, la notación [B+], [B-], [X+], o [X-] especifica cual registro sirve cono el apuntador, y si debe incrementarse o decrementarse el apuntador.

- **Inmediato**

  El dato de la operación viene detrás del código de operación en la memoria de programa. En lenguaje ensamblador, el número con el símbolo (#) indica un operando inmediato.

- **Inmediato Corto**

  Este es un caso especial de instrucción inmediata. En la instrucción "Load B inmmediate", los 4-bits del valor inmediato en la instrucción son cargados en los 4-bit menos significativos del registro B. Los 4-bit más significativos d el registro B son puestos a ceros binarios.

- **Indirecto de Memoria de Programa**

  Este es un caso especial de una instrucción indirecta que permite accesar la tabla de datos almacenada en la memoria de programa. En la instrucción "Load Accumulator Indirect" (LAID), los bytes altos y bajos del Contador de Programa (PCU y PCL) son usados temporalmente como un apuntador para la memoria de programa. Para propósitos del acceso a la memoria de programa, el contenido del Acumulador y de PCL son intercambiados. El dato apuntado por el Contador de Programa es cargado en el Acumulador, y simultáneamente, el contenido original de PCL es almacenado para que el programa pueda iniciar la ejecución normal.

Modos de direccionamiento de transferencia de Control

Las instrucciones de programa son normalmente ejecutadas en orden secuencial. Sin embargo, las instrucciones JUMP pueden ser usadas para cambiar la secuencia de la ejecución normal. Varios modos de direccionamiento de transferencia de control están disponibles para direccionamiento de saltos específicos.

Un cambio en el flujo del programa requiere un cambio no incremental en el contenido del Contador de Programa. El Contador de Programa consiste de dos bytes, designados a los 4-bit altos

(PCU) y a los 4-bit bajos(PCL). Los bits más significativos de PCU no son usados, dejando 15 bits para direccionar la memoria de programa.

Diferentes modos de direccionamiento son usados para especificar el nuevo direccionamiento del Contador de Programa. La elección del modo de direccionamiento depende básicamente en la distancia del salto. Más que saltos, algunas veces se requieren más bytes de instrucción a fin de especificar completamente el nuevo contenido del Contador de Programa.

Los modos de direccionamiento de transferencia de control disponibles son:

- **Salto Relativo**
  En esta instrucción de 1 byte, seis bits del código de operación de instrucción especifican la distancia del salto de la localidad actual de memoria de programa. La distancia del salto puede ser de –31 a +32. Una instrucción JP+ no es permitida. El programa puede usar en su lugar un NOP.

- **Salto Absoluto**
  En esta instrucción de 2-bytes, 12 bits del código de operación de la instrucción, especifican el nuevo contenido del Contador de Programa. Los tres bytes superiores del PC permanecen sin cambio, limitando el nuevo direccionamiento del PC para el mismo espacio de direccionamiento de 4-Kbytes como la instrucción actual (ésta limitación es relevante solamente en dispositivos que usen más de 4-Kbytes de espacio en memoria de programa).

- **Salto Absoluto Largo**
  En esta instrucción de 3-bytes, 15 bites del código de operación de la instrucción especifican el nuevo contenido del Contador de Programa

- **Salto Indirecto**
  En esta instrucción de 1-byte, el bit bajo de la dirección de salto es obtenido de una tabla almacenada en la memoria de programa, con el servicio del Acumulador como el byte bajo de un apuntador dentro de la memoria de programa Para propósitos de acceso a la

memoria de programa el contenido del Acumulador es escrito a PCL (temporalmente). El dato apuntado por el PC (PCH/PCL), es almacenado en PCL, mientras PCH permanece sin almacenarse.

4.12.2 Resumen de instrucciones

Las siguientes abreviaturas representan la nomenclatura usada en la descripción de las instrucciones y en el ensamblador del COP8:

A	Acumulador.
B	Puntero B, localizado en el registro de memoria RAM en la localidad 00FE.
[B]	Contenido de la localidad de memoria de dato RAM indicado por el puntero B.
[B+]	Lo mismo que [B], excepto que puntero es post-incrementado.
[B-]	Lo mismo que [B], excepto que puntero es post-decrementado.
C	Bandera de acarreo, localizada en el bit 6 del registro PSW en la localidad de memoria 00EF Hex.
HC	Bandera de acarreo media, localizada en el bit 7 del registro PSW en la localidad de memoria 00EF Hex
MA	Dirección de memoria de 8-bit para almacenamiento de datos RAM.
MD	Memoria Directa, la cual puede ser representada por una etiqueta implícita (B, X, SP), una etiqueta definida (TEMP, COUNTER, etc.), o una dirección de memoria directa (12, 0EF, 027,etc. donde un 0 indica hexadecimal).
PC	Contador de Programa (15 bits, con un rango de direccionamiento de memoria de programa de 32768).
PCU	Contador de Programa Alto, el cual contiene los 7 bits superiores de PC.
PCL	Contador de Programa Bajo, el cual contiene los 8 bits inferiores de PC.
PSW	Registro de palabra del estado del procesador, ubicado en la localidad de memoria 00EF.
REG	Registro Seleccionado (1 de 16) de la memoria de datos RAM en la dirección 00F0-00FF.
REG#	# el registro de memoria para ser usado (# = 0-F hexadecimal).
#	El símbolo # es usado para indicar un valor inmediato, con un cero (0) indica hexadecimal. Ejemplos: #045 = valor inmediato del hexadecimal 45 #45 = valor inmediato del decimal 45 # puede también ser usado para indicar la posición del bit, donde # = 0-7

Ejemplo:

RBIT #, [B]

SP	Puntero de Pila, localizado en el registro de memoria RAM en la localidad 00FD.
X	Puntero X, localizado en el registro de memoria RAM en la localidad 00FC.
[X]	Contenido de la localidad de memoria de datos RAM indicado por el puntero X.
[X+]	Lo mismo que [X], excepto que el puntero X es post-incrementado.
[X-]	Lo mismo que [X], excepto que el puntero X es post-decrementado.

El conjunto de instrucciones esta formado por una gran variedad de éstas. Las instrucciones disponibles se resumen en una tabla abajo, en el apéndice A se listan a detalle cada una de las instrucciones

Resumen del Conjunto de Instrucciones

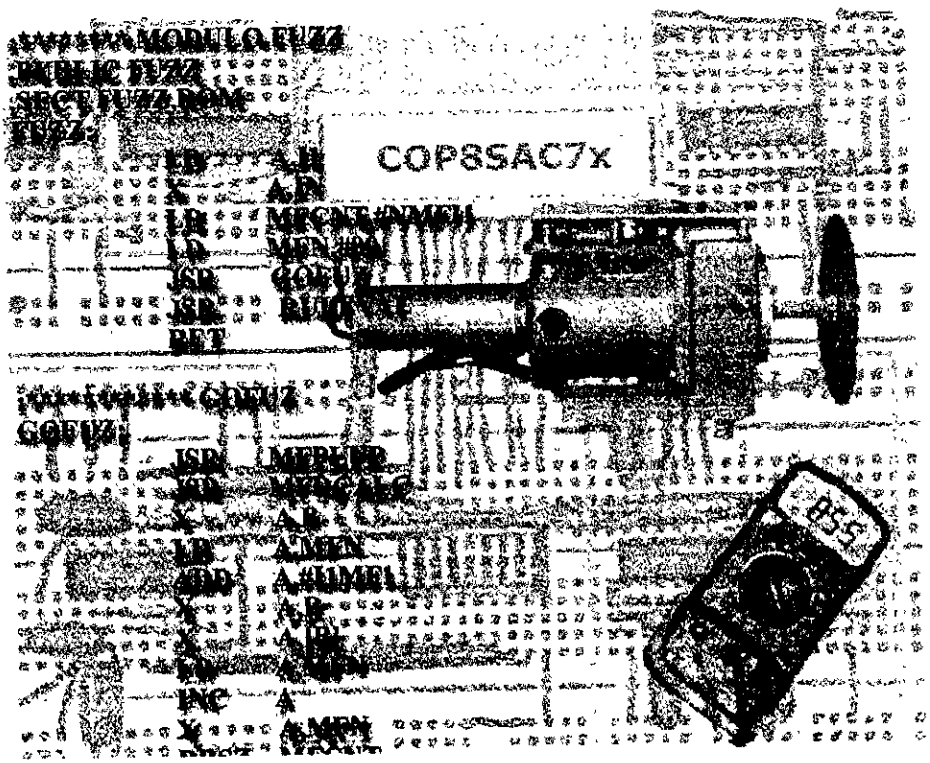
ADD	A, Meml	ADD	A→A + Meml
ADC	A, Meml	ADD con Acarreo	A→A +Meml +C, C→Acarreo, HC→Medio Acarreo
SUBC	A, Meml	Sustracción con Acarreo	A→A - Meml +C, C→ Acarreo, HC→Medio Acarreo
AND	A, Meml	AND Lógico	A→A y Meml
ANDSZ	A, Imm	AND Lógico Inmed , Salta si = 0	Saltar siguiente si (A y Imm) = 0
OR	A, Meml	OR Lógico	A←A o Meml
XOR	A, Meml	OR EXclusiva Lógica	A←A o exclusivamente Meml
IFEQ	MD, Imm	Si Igual	Comparar MD y Imm, Hacer el siguiente si MD = Imm
IFEQ	A, Meml	Si Igual	Comparar A y Meml, Hacer el siguiente si A = Meml
IFNE	A, Meml	Si No Igual	Comparar A y Meml, Hacer el siguiente si A ≠ Meml
IFGT	A, Meml	Si Mayor que	Comparar A y Meml, Hacer el siguiente si A > Meml
IFBNE	#	Si B No Igual	Hacer el siguiente si los 4 bits bajos de B ≠ Imm
DRS	Reg	Decrementar Reg , Saltar si = 0	Reg←Reg - 1, Saltar si Reg = 0
SBIT	#, Mem	Fijar BIT	1 a bit, Mem (bit = 0 a 7 inmediato)

RBIT	#, Mem	Restablecer BIT	0 a bit, Mem
IFBIT	#, Mem	Si BIT	Si bit #, A o Mem es verdad hacer la siguiente instrucción
RPND		Restablecer Bandera PeNDiente	Reestablecer la Interrupción de Software Pendiente
X	A, Mem	Cambiar A con Memoria	A↔Mem
X	A, [X]	Cambiar A con Memoria [X]	A↔ [X]
LD	A, Meml	Cargar A con Memoria	A←Meml
LD	A, [X]	Cargar A con Memoria [X]	A← [X]
LD	B, Imm	Cargar B con Inmediata.	B←Imm
LD	Mem, Imm	Cargar Memoria Inmediata.	Mem←Imm
LD	Reg, Imm	Cargar Registro Memoria Inmed.	Reg←Imm
X	A,[B±]	Cambiar A con Memoria [B]	A↔[B], (B←B ±1)
X	A,[X±]	Cambiar A con Memoria [X]	A↔[X], (X←X ±1)
LD	A, [B±]	Cargar A con Memoria [B]	A← [B], (B←B ±1)
LD	A, [X±]	Cargar A con Memoria [X]	A← [X], (X←X±1)
LD	[B±],Imm	Cargar Memoria [B] Inmediata .	[B]←Imm, (B←B±1)
CLR	A	Borrar A	A←0
INC	A	INCrementar A	A←A +1
DEC	A	DECrementar A	A←A - 1
LAID		Cargar A InDirecta de ROM	A←ROM (PU, A)
DCOR	A	Decimal CORrecto A	A←BCD corrección de A (seguir ADC, SUBC)
RRC	A	Rotar A a la Derecha hasta C	C→A7→ .→A0→C
RLC	A	Rotar A a la Izquierda hasta C	C←A7←.. ←A0←C, HC←A0
SWAP	A	SWAP nibbles de A	A7 . A4↔A3.. A0
SC		Fijar C	C←1, HC←1
RC		Restablecer C	C←0, HC←0
IFC		Si C	Si C si es verdad, has la siguiente instrucción
IFNC		Si No C	Si C si no verdad, has la siguiente instrucción
POP	A	POP la pila dentro de A	SP←SP +1,A← [SP]

PUSH	A	PUSH A sobre la pila	[SP]←A, SP←SP - 1
VIS		Vector para la Rutina del Servicio de Interrupción	PU← [VU], PL← [VL]
JMPL	Addr.	Salto Absoluto Largo	PC←u (ii = 15 bits, 0 a 32k)
JMP	Addr	Salto Absoluto	PC9...0←i (i =12 bits)
JP	Disp.	Salto Relativo Corto	PC←PC +r (r es -31to+32, excepto 1)
JSRL	Addr.	Subrutina de Salto Largo	[SP]←PL, [SP-1]←PU, SP-2, PC←u
JSR	Addr.	Subrutina de Salto	[SP]←PL, [SP-1]←PU, SP-2, PC9...0←i
JID		Salto Indirecto	PL←ROM (PU, A)
RET		Regreso de una subrutina	SP + 2, PL←[SP], PU←[SP-1]
RETSK		Regreso y Salto	SP + 2, PL←[SP],PU←[SP-1], saltar a la siguiente instrucción
RETI		Regreso de una Interrupción	SP + 2, PL← [SP],PU← v[SP-1],GIE←1
INTR		Generar una Interrupción	[SP]←PL, [SP-1]←PU, SP-2, PC←0FF
NOP		No Operacion	PC←PC +1

CAPITULO 5

APLICACION



CAPITULO 5 APLICACIÓN

5.1 Introducción.

El objetivo de este capitulo es el desarrollar una aplicación de control haciendo uso de la técnica de lógica difusa y la implantación de la solución de ésta en un microcontrolador de 8 bits de la National Semiconductor COP8.

Para el método de lógica difusa se utiliza la herramienta de desarrollo (NeuFuz) creada por National Semiconductor. NeuFuz utiliza una red neuronal y un conjunto de datos de entrenamiento para la creación automática de las reglas difusas. Además posee una rutina que permite la evaluación y la prueba de la solución obtenida. Otra ventaja adicional es que NeuFuz cuenta con un módulo capaz de generar el código ensamblador una vez que el diseño de lógica difusa se ha completado.

El dispositivo utilizado es el COP8SAC740Q3 (dispositivo OTP) que pertenece a la familia de los COP8, microcontrolador de 8-bits, también de National Semiconductor. Para la implantación de la lógica difusa, así como del programa principal, se uso la unidad de evaluación y programación (EPU) desarrollada por iceMASTER para esta familia de microcontroladores. La EPU utiliza un circuito simulador el cual combina un simulador y un circuito emulador, el cual permite depurar el software y corregir los llamados del hardware externo usado en la aplicación.

La aplicación que se desarrolla en el presente capítulo trata del diseño de un esquema de control de un motor de corriente continua (CD), basado en la tecnología de lógica difusa. En muchas aplicaciones el control de la precisión de la posición y la velocidad en un motor es muy importante Por ejemplo en el auto-enfoque de una cámara de fotos, de una cámara de video o en el apropiado posicionamiento de un servo motor en un disco duro. En estos ejemplos el algoritmo de control es clave en el correcto funcionamiento o falla del producto.

Ya que este diseño utiliza un conjunto de datos de entrenamiento éste a sido obtenido del propio motor de corriente directa mediante la respuesta a un sistema de lazo abierto Una ventaja de

este diseño es que puede usarse para diferentes características de motor sin tener que volver escribir el código del algoritmo. El motor de CD forma parte del servomecanismo modular MS 150. Este sistema es un dispositivo mecánico y entre sus módulos cuenta con una fuente de alimentación (SA 150 E), un amplificador (SA 150 D) y un generador de corriente directa, acoplado directamente al eje del motor para medir su velocidad.

El apartado "NeuFuz" trata de la construcción de la solución de lógica difusa, haciendo uso del software de desarrollo, que será implantada en el microcontrolador. Primero se entrena la red neuronal con el conjunto de datos obtenidos del motor de CD. Una vez que la red neuronal alcanza la convergencia se esta listo para evaluar las reglas obtenidas y generar el código de lenguaje ensamblador, de la lógica difusa. El paso siguiente, "Diseño de Software", es incluir en el programa principal el código generado en el punto anterior, con el objeto de hacer un correcto llamado a las subrutinas de lógica difusa. A continuación se describen los elementos de hardware que participan en el esquema de control, tal como los convertidores ADC0804 (Convertidor Analógico-Digital) y DAC0800 (Convertidor Digital-Analógico), el generador que proporcione la velocidad actual del motor de CD y los módulos del servomecanismo que participan en el funcionamiento del motor. Por último se aplican en conjunto el hardware y software realizando los ajustes finales y una evaluación completa de la operación del motor para determinar el desempeño de la solución.

5.2 Entrenamiento de la red neuronal

Esta parte es la primera en la construcción de la solución del problema de control que se ha mencionado en párrafos anteriores. El proceso comienza con la selección de los datos que se utilizan en el aprendizaje de la red neuronal, el paso siguiente es el aprendizaje de estos datos mediante NeuFuz. El paso final de esta parte es la edición y evaluación de las funciones de membresía para mejorar la aproximación de la solución.

5.2.1 Conjunto de datos

Para la obtención del conjunto de datos de entrenamiento se utilizó el sistema modular del servomecanismo MS 150, usado en las prácticas del laboratorio de Control. Los módulos utilizados en este proceso fueron:

Fuente de alimentación PS 150 E.- La alimentación para que todo el sistema funcione es provista por este módulo. Necesita a la entrada un voltaje de línea de 127 V con una frecuencia de 60 Hz. Los voltajes entregados en sus salidas son de: 24 volts de CD no regulada, 15 volts de CD regulada y –15 volts de CD regulada. Cuenta con un amperímetro que permite observar la condición de sobrecarga de corriente alcanzada por el motor. En la figura 5.1 se muestra este módulo

Servo Amplificador SA 150 D.- Este módulo permite configurar los dos tipos de control que permite el motor: Control por armadura y Control por campo. Cuenta con dos entradas las cuales determinan el giro del motor. No necesita de alimentación externa ya que la recibe directamente de la unidad de alimentación PS 150 E En la figura 5.1 se muestra este dispositivo.

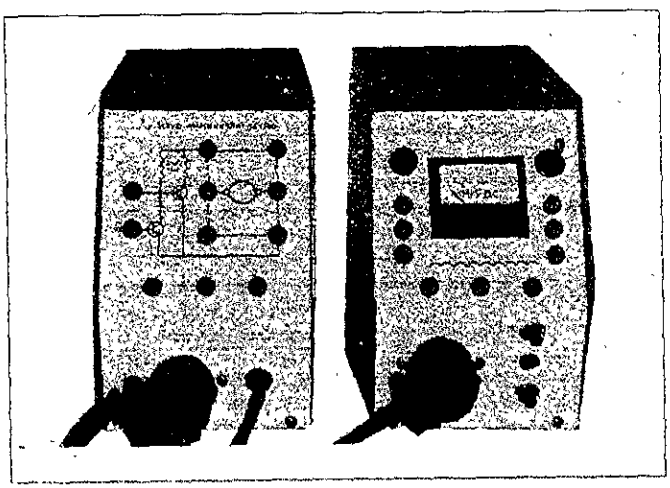


Figura 5.1 Servo Amplificador y Fuente de Alimentación.

Unidad atenuadora AU 150 B.- Dos potenciómetros lineales de 10 Kohms son los que constituyen a este módulo Su función es servir como simples divisores de voltaje a la entrada del servo amplificador.

Motor de Corriente Directa MT 150 F.- Este servomotor contiene dos devanados independientes que utilizan voltajes de CD, uno para la armadura (rotor) y el otro para el de campo (estator) Cuenta con conexiones externas, colocadas en el módulo amplificador SA 150 D, para configurar el tipo de control (Armadura o de Campo) Un tacómetro inter-construido, formado por un pequeño

generador acoplado al eje principal del motor, permite corregir la velocidad angular del motor, ver la figura 5.2 . El generador produce una relación aproximada de 3 mV/rpm.

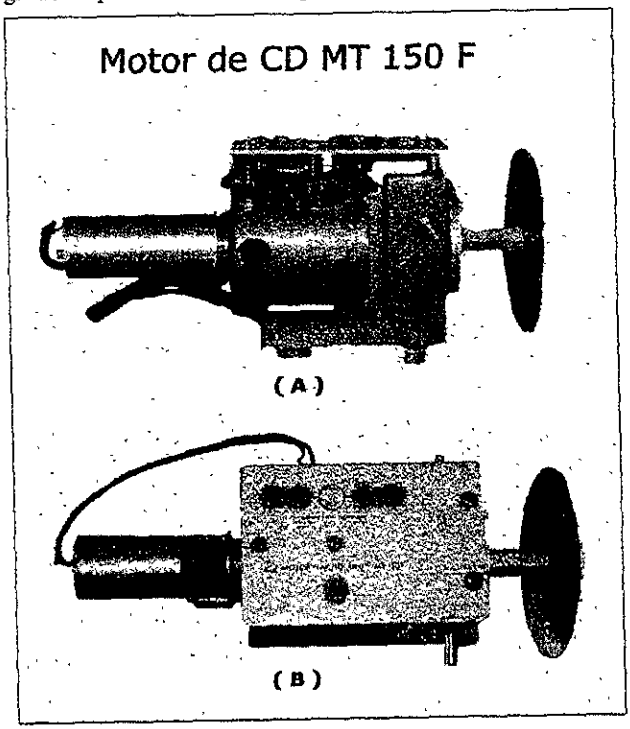


Figura 5.2 (A) Vista lateral, (B) Vista superior del motor de CD.

Configurando el motor en alimentación por armadura se tomaron 59 muestras variando el voltaje de entrada en incrementos de 0.1 volts, en un rango de valores de 4 2 a 10 volts. El motor presenta una zona muerta (voltaje de entrada en el cual el motor no arranca) de 0 a 4.2 volts. Los datos obtenidos se presentan en la siguiente lista:

Ve	Vs	Wm
4 2	0	0
4 3	1.187	396
4 4	1.26	420
4 5	1 335	445
4 6	1.449	483

Ve	Vs	Wm
4.7	1 619	540
4.8	1 698	566
4.9	2 2	733
5 0	2.356	785
5 1	2.683	894

Ve	Vs	Wm
5.2	2 708	903
5.3	2.79	930
5.4	2.818	939
5 5	2.846	949
5.6	2 943	981

Ve	Vs	Wm
5.7	3 097	1032
5.8	3.252	1084
5.9	3.348	1116
6.0	3.728	1243
6 1	3.975	1325
6.2	4.02	1340
6.3	4.162	1387
6 4	4.21	1403
6.5	4.383	1461
6.6	4.474	1491
6 7	4 971	1657
6 8	5.079	1693
6 9	5.297	1766
7.0	5.539	1846
7.1	5 612	1871

Ve	Vs	Wm
7.2	5 886	1962
7.3	5.954	1985
7.4	6.266	2089
7.5	6.314	2105
7.6	6.567	2189
7.7	6.831	2277
7 8	6 952	2317
7 9	7.178	2393
8.0	7 302	2434
8.1	7.362	2454
8.2	7.422	2474
8 3	7.533	2511
8.4	7.616	2539
8 5	7.777	2592
8.6	7.939	2646

Ve	Vs	Wm
8.7	8.197	2732
8 8	8 341	2780
8.9	8.434	2811
9.0	8.776	2925
9.1	8.894	2965
9.2	9.212	3071
9.3	9.37	3123
9.4	9 569	3190
9.5	9.98	3327
9 6	10.106	3369
9.7	10.79	3597
9.8	10 96	3653
9 9	11.27	3757
10.0	11 73	3910

Al graficar velocidad angular del motor (Wm) contra voltaje de entrada (Ve) se obtiene la gráfica que se presenta en la figura 5 3.

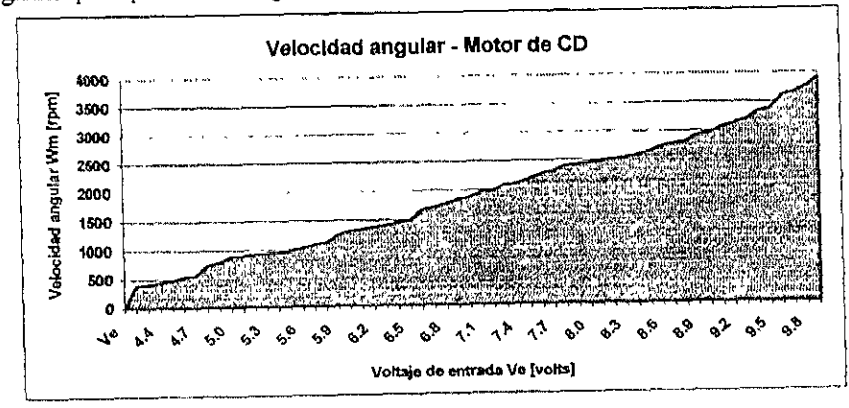


Figura 5.3 Gáfica de datos experimentales.

Como se puede observar el comportamiento de la gráfica no es lineal, aunque se aproxima en cierto grado Para fines de comparación se puede obtener una ecuación que

represente a este conjunto de datos ajustando la curva por medio de un método de regresión lineal, tal como la aproximación por mínimos cuadrados. La ecuación resultante es:

$$Vm = 580.79Ve - 2227.75$$

al graficar esta ecuación y compararla con la de los datos experimentales se puede observar con mayor claridad el echo de que la respuesta en la velocidad del motor al incremento del valor del voltaje en la entrada no es precisamente proporcional, ver la figura 5.4.

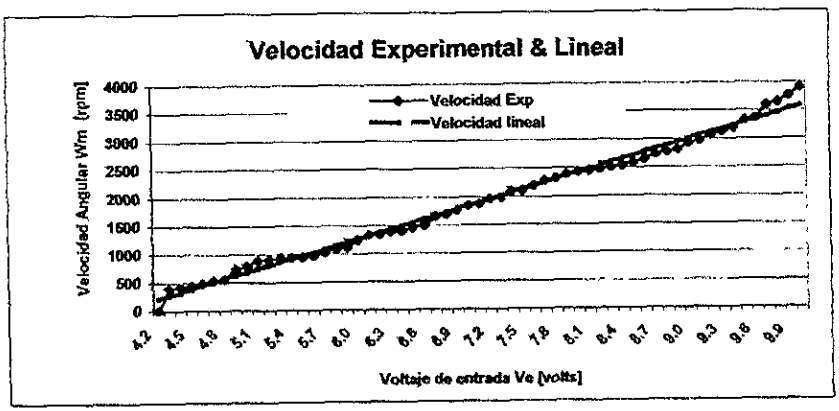


Figura 5.4 Velocidad Experimental & Velocidad Lineal

El hacer esta comparación toma importancia ya que lo que deseamos es encontrar una solución que sea lo más precisa que se pueda sobre el conjunto de datos experimentales El diseño de lógica difusa debe ser capaz de proporcionar, con el menor error posible, el comportamiento de este conjunto de datos. El siguiente apartado trata del entrenamiento de la red neuronal de estos datos.

5.2.2 Aprendizaje de datos

Una vez que se cuenta con el conjunto de datos hay que decidir que variables serán tomadas para describir el comportamiento deseado. En nuestro caso deseamos obtener el control de la velocidad del motor de corriente directa proporcionando el valor de voltaje

necesario. Por ejemplo, si se quiere que el motor tenga una velocidad angular (Wm) de 930 rpm se debe de proporcionar un voltaje de entrada de 5.3 volts, según las características obtenidas en los datos de prueba (ver la tabla de datos). Por lo tanto, el valor que daremos de entrada al diseño del controlador difuso será el valor de velocidad que deseamos que el motor presente (Wm) y el valor de salida esperado debe ser el voltaje necesario (Ve) para que el motor alcance dicha velocidad.

Después de decidir quien es la entrada y quien la salida, en nuestro diseño difuso, el paso siguiente es la creación de un archivo que contenga el conjunto de datos que deseamos aprenda la red neuronal. El archivo puede ser creado con cualquier editor de textos, teniendo la precaución de salvarlo sin ningún formato de texto enriquecido. El archivo puede tener cualquier nombre pero su extensión debe ser forzosamente "*.inp", en nuestro caso llamaremos al archivo "motor.inp". Tal y como se explico en el capítulo tres la estructura del archivo de entrada debe contar con la (o las) columna(s) de los datos de entrada y la columna del valor de salida deseado. Un extracto del archivo utilizado en esta aplicación es el siguiente.

```
0     4.2                    •
420   4.4                    •
445   4.5                 3071  9.2
483   4.6                 3123  9.3
540   4.7                 3190  9.4
733   4.9                 3369  9.6
785   5.0                 3597  9.7
903   5.2                 3653  9.8
930   5.3                 3757  9.9
939   5.4                 3910 10.0
  •
```

NOTA: Debido a que el programa NeuFuz que es proporcionado con el KIT de evaluación es demostrativo y no posee toda la capacidad de la versión profesional, de los 59 datos con los que se cuenta solo se usarán 50, que es el número máximo permitido en esta versión.

El archivo creado es colocado en un directorio, creado con anterioridad, llamado MOTOR y que cuelga del directorio principal de NeuFuz (C:\NeuFuz\MOTOR\motor.inp). Es en este directorio en donde se localizarán todos los archivos de nuestra aplicación usados por NeuFuz.

Una vez que se han preparado el directorio y el archivo se ejecuta NeuFuz y se carga el modelo (el archivo de datos), se especifica el directorio donde se localiza el modelo y se procede a fijar el valor de los parámetros de entrenamiento.

NOTA: El número máximo de funciones de membresía y de entradas, para este KIT de valuación, es de 3 funciones de membresía y dos entradas.

Primera corrida

Para esta primea corrida del programa se fija el valor de Epsilon en 0.01, este valor corresponde al grado de precisión que deseamos tenga nuestra solución. Como se explica en el capitulo tres una guía para la determinación de los otros dos parámetros es el observar la diferencia entre los valores de la variable de salida. La diferencia entre los valores de salida es pequeña por lo tanto el valor de los parámetros debe ser grande, tal como 0.5 ó 0.1, el porcentaje de aprendizaje se fija en 0.5 y el factor de aprendizaje en 0.5. Se abre la ventana de error para observar el comportamiento del Err/Eps Máximo y se inicia la corrida.

La siguiente lista muestra los resultados obtenidos durante los primeros ciclos:

Err/Eps Máx.	Ciclo
29.02	567
-30 63	1061
30.10	1566
-28 17	2080
25.92	2580
23.97	3030
30.73	3581
30.28	4006
27 27	4552
-21.72	5015

Para que exista convergencia se debe observar que el Err/Eps Máx. debe ir decreciendo y que no oscile alrededor de algún valor. Como se aprecia en la tabla no solamente no decrece Err/Eps Máx. sino que también oscila arriba del valor 20. Para tratar de evitar que la red aprenda demasiado rápido, lo que provoca que no se aproxime a la convergencia, se debe disminuir el valor de los parámetros de aprendizaje.

Segunda corrida

Después de reinicializar el modelo, se reajustan los parámetros de aprendizaje con los siguientes valores. el porcentaje de aprendizaje se pone a 0.1 y el factor de aprendizaje en 0.1. Se actualiza la ventana de errores y se inicia la corrida.

La siguiente lista muestra los resultados obtenidos durante los primeros ciclos:

Err/Eps Máx.	Ciclo
37.11	537
-23.93	1016
-17.44	1530
13.57	2017
-11.70	2539
11.73	3027
11.07	3501
-12.20	4030
-11.24	4516
11.11	5010

Se puede observar que la aproximación a la convergencia mejora (se redujo el valor de Err/Eps Máx.), se observa que comienza a decrecer el valor de Err/Eps Máx., pero aproximadamente al ciclo 3000 comienza a oscilar. Esto puede tratar de mejorarse disminuyendo el valor del factor de aprendizaje, tal y como se realiza en la tercera corrida.

Tercera corrida

Se reinicializa el modelo, se reajusta el factor de aprendizaje en 0.01. Se actualiza la ventana de errores y se inicia la corrida

La siguiente lista muestra los resultados obtenidos durante los primeros ciclos:

Err/Eps Máx.	Ciclo
-140.8	530
-155.1	1001
-151.6	1650
-124.2	2039
-127.7	2536
-136 6	3095
-123.1	3612

El valor Err/Eps Máx. comienza disminuyendo su valor, pero aproximadamente en el ciclo 3000 comienza a oscilar, además de que el valor de Err/Eps Máx. queda muy grande. Como las oscilaciones siguen presentándose y no se mejora la convergencia entonces hay que disminuir el valor del porcentaje de aprendizaje.

Cuarta corrida

Se reinicializa el modelo, se reajusta el porcentaje de aprendizaje en 0.01 y el factor de aprendizaje en 0.1. Se actualiza la ventana de errores y se inicia la corrida.

La siguiente lista muestra los resultados obtenidos durante los primeros ciclos:

Err/Eps Máx.	Ciclo
-54.98	531
-47 46	1046
-45.42	1527
-44.07	2034
—	—
—	—
-35.93	5042
-34.50	5602
-33.35	6062
-32 19	6542
—	—
—	—
25 60	10020
24 96	10519
24.15	11057

23.51	11570
—	—
—	—
13.98	20109
11.85	25077
11.14	30084
10.65	35187
—	—
—	—
-10 66	40048
-10.76	45098
-10 87	50011
-10.85	55035

Se puede observar que se mejora el acercamiento a la convergencia pero aproximadamente en el ciclo 40000 comienza a tener una oscilación infinita. Como ya se trato cambiando ambos parámetros, tanto el porcentaje de aprendizaje como el factor de aprendizaje, y ya no se considera como un valor viable disminuir aún más su valor, entonces lo recomendable es aumentar el valor de Epsilon, tal y como se hace en la quinta corrida.

Quinta corrida

Se reinicializa el modelo y se ajustan Epsilon en 0.1, el porcentaje de aprendizaje en 0.1 y el factor de aprendizaje en 0 1. Se actualiza la ventana de errores y se inicia la corrida.

La siguiente tabla muestra el comportamiento de Err/Eps Máx. para estos nuevos valores

Err/Eps Máx.	Ciclo
3.975	500
2 580	1005
2 003	1569
1 637	2066
1.256	3142
1.207	3623
1 154	4022
1.088	4589
1.099	5146
1 056	5534
1.048	6042
1.020	6562
-1.033	7040
-1 027	7524
Convergencia	8015

Con este nuevo valor de Epsilon el valor de Err/Eps Máx. disminuye constantemente no presenta oscilaciones y alcanza la convergencia en el ciclo 8015

Aunque ya se ha alcanzado la convergencia todavía se puede mejorar el resultado, hay que recordar la definición de Err/Eps:

$$\text{Err/Eps} = \frac{\text{valor de salida deseado} - \text{valor de salida calculado}}{\text{Epsilon}}$$

como lo indica la ecuación y sabiendo que Err/Eps debe ser 1 para que exista convergencia, entonces:

Epsilon = valor de salida deseado – valor de salida calculado

por lo que, Epsilon representa el grado de precisión de la solución. Entonces, ya que a un valor de Epsilon grande corresponde una menor precisión, se disminuirá Epsilon para tratar de mejorar la solución, tal y como se muestra en la corrida seis.

Sexta corrida

Se realizó el cambio de Epsilon a un valor de 0.09 pero no se alcanzó la convergencia y presento oscilaciones infinitas. Entonces se fijo el valor a 0.095, manteniendo al porcentaje de aprendizaje en 0 1 y al factor de aprendizaje en 0.1 alcanzándose la convergencia tal y como se muestra en la siguiente tabla:

Err/Eps Máx.	Ciclo
-3.6660	525
2 5550	1075
-1.7569	1626
1 5091	2078
1 3065	2541
1.3601	3066
—	—
—	—
-1 1559	5071
-1 0957	6116
-1 1644	7029
-1 1820	8135
—	—
—	—
-1.0991	10085
-1 0880	15008
-1.0499	20075
-1.0380	25046
-1 0971	30081
-1.0644	35060
-1.0494	40062
Convergencia	41450

Como se ve en la tabla la red comienza a converger lentamente y a pesar que en la parte final presenta oscilaciones pequeñas alcanza la convergencia en el ciclo 41450. En el siguiente apartado se trata el ajuste de esta solución y la evaluación del resultado.

5.2.3 Edición de funciones y prueba de la solución

Una forma de mejorar la solución es la de incrementar el número de funciones de membresía, pero debido a que en esta versión de NeuFuz el número máximo es de tres se descarta esta opción. Para una mejor identificación de las funciones de membresía de entrada se les asignara un nombre de la siguiente forma: *1ra. Función = velocidad BAJA (VB), 2da. Función = velocidad MEDIA (VM), 3ra. Función = velocidad ALTA (VA).* A la variable de entrada se le nombra "VelocidadAngular" y a la variable de salida "VoltajeEntrada" del sistema difuso. Ahora hay que editar las funciones de membresía, la figura 5.5 muestra a éstas todavía sin ser editadas. Las líneas rectas representan la aproximación a las funciones de membresía optimizadas (líneas curvas).

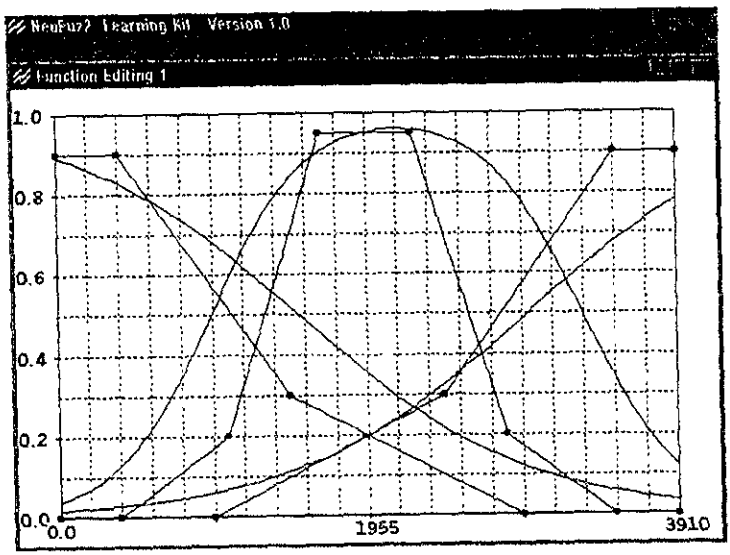


Figura 5. 5 Funciones membresía antes de ser editadas.

La edición se refiere a encontrar la mejor aproximación de las líneas rectas a las curvas, la edición de las funciones de membresía se describe en el capítulo 3. La figura 5.6 muestra las funciones de membresía una vez que han sido editadas

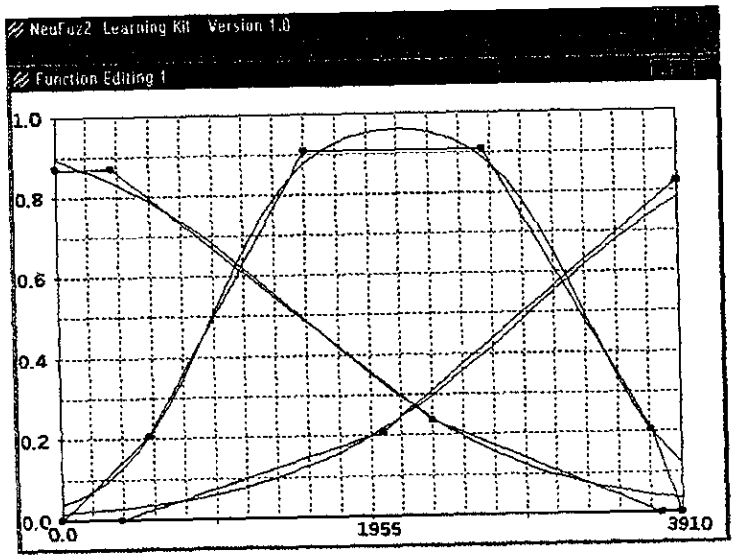


Figura 5. 6 Funciones de membresía después de ser editadas.

Ya que las funciones han sido editadas el siguiente paso es la evaluación de la solución. Para ello abrimos la ventana de verificación de reglas y se desliza la columna del conjunto de datos de entrada para determinar la respuesta a su conjunto de valores. Por ejemplo si la columna de los datos de entrada la colocamos al valor 1848.04 rpm los valores de las columnas Neural Net (red neuronal), Rules Bell MF (funciones de membresía de campana) y Appx MF (aproximación de las funciones de membresía) son 7.04, 7.04 y 7.03 volts respectivamente.

Si comparamos con el archivo de datos de entrada ("motor.inp") se ve que para la velocidad de 1846 rmp se necesita un voltaje de entrada de 7.0 volts, lo que quiere decir que el valor que se obtuvo en el diseño difuso es muy aceptable, ver la figura 5.7.

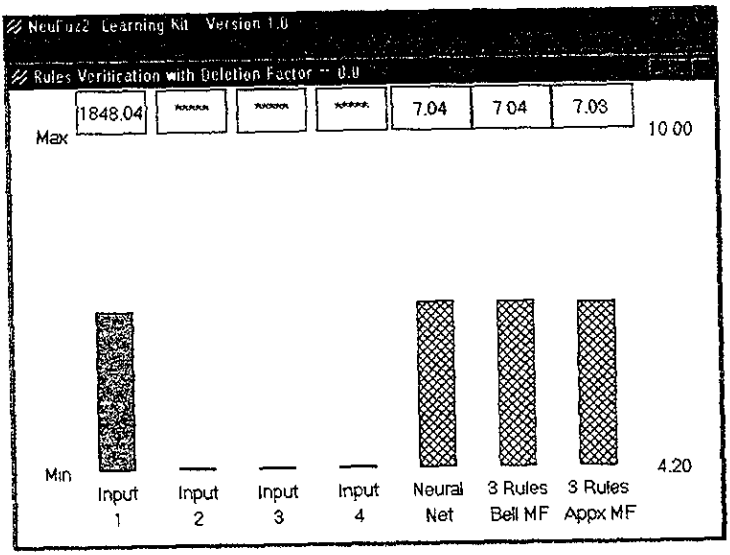


Figura 5.7 Verificación de la solución.

Para obtener una evaluación completa sobre todos los puntos deseados de nuestra aplicación se debe generar un archivo de salida ("motor.out") con ayuda de otro archivo llamado motor.rec, el cual contiene una columna con todos los valores de entrada que se deseen comprobar. La figura 5.8 muestra el archivo generado para esta aplicación.

```
# Rules Verification Results for Model 'motor'
# Deletion Factor 0.000

#Input    Neural   3 Rules   3 Rules
# 1       Net      Bell MF   Appx MF
#-------------------------------------

  0.000   4.292    4.292     3.861
396.000   4.453    4.453     4.311
420.000   4.474    4.474     4.343
445.000   4.498    4.498     4.376
483.000   4.537    4.537     4.427
540.000   4.606    4.606     4.503
566.000   4.642    4.642     4.548
733.000   4.923    4.923     4.957
```

785.000	5.030	5.030	5.085
894.000	5.274	5.274	5.352
903.000	5.295	5.295	5.374
930.000	5.359	5.359	5.440
939.000	5.381	5.381	5.462
949.000	5.405	5.405	5.486
981.000	5.482	5.482	5.565
1032.000	5.605	5.605	5.690
1084.000	5.730	5.730	5.817
1116.000	5.805	5.805	5.895
1243.000	6.088	6.088	6.206
1325.000	6.252	6.252	6.407
1340.000	6.281	6.281	6.444
1387.000	6.366	6.366	6.559
1403.000	6.394	6.394	6.598
1461.000	6.491	6.491	6.740
1491.000	6.539	6.539	6.814
1657.000	6.780	6.780	7.012
1693.000	6.829	6.829	7.015
1766.000	6.928	6.928	7.022
1846.000	7.039	7.039	7.030
1871.000	7.074	7.074	7.033
1962.000	7.207	7.207	7.041
1985.000	7.242	7.242	7.044
2089.000	7.410	7.410	7.177
2105.000	7.438	7.438	7.219
2189.000	7.587	7.587	7.436
2277.000	7.755	7.755	7.664
2317.000	7.834	7.834	7.767
2393.000	7.991	7.991	7.995
2434.000	8.078	8.078	8.129
2454.000	8.121	8.121	8.195
2474.000	8.165	8.165	8.260
2511.000	8.245	8.245	8.382
2539.000	8.306	8.306	8.474
2592.000	8.422	8.422	8.648
2646.000	8.538	8.538	8.825
2732.000	8.720	8.720	9.018
2780.000	8.816	8.816	9.066
2811.000	8.876	8.876	9.097
2925.000	9.079	9.079	9.210
2965.000	9.142	9.142	9.250

3071.000	9.286	9.286	9.355
3123.000	9.345	9.345	9.407
3190.000	9.410	9.410	9.473
3327.000	9.517	9.517	9.610
3369.000	9.546	9.546	9.651
3597.000	9.718	9.718	9.878
3653.000	9.771	9.771	9.933
3757.000	9.886	9.886	9.999
3910.000	10.088	10.088	9.991

Figura 5.8 Listado de archivo de salida "motor.out".

Con ayuda del archivo inicial ("motor.inp") se puede observar el grado de precisión que posee la solución obtenida. Con ayuda de una gráfica podemos darnos cuenta que el conjunto de datos ha sido aprendido por la red neuronal y que las funciones de membresía si proporcionan un resultado que sigue el comportamiento de las características de no-linealidad que presenta el motor de corriente directa. La figura 5.9 muestra una gráfica que ilustra lo anterior.

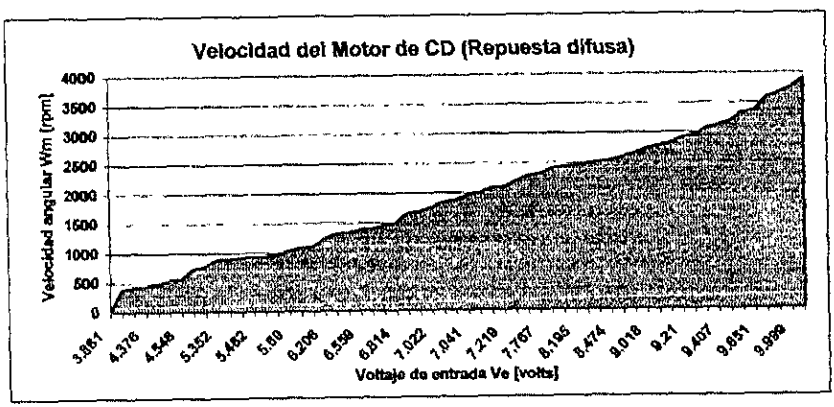


Figura 5.9 Respuesta Difusa.

Al comparar con la gráfica de la figura 5.3 se puede apreciar el gran parecido entre ambas. Para ver la diferencia a detalle obsérvese la figura 5.10, en donde se han trazado el valor del voltaje de entrada experimental y el valor del voltaje de entrada calculado por el diseño difuso. Uno de los ejes se a desplazado verticalmente para observar mejor. Como se puede ver las curvas no muestran una gran diferencia a lo largo del rango de valores, tal y como se deseaba.

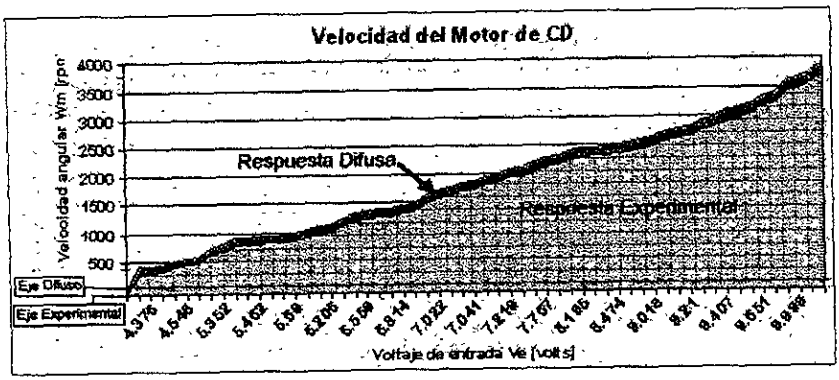


Figura 5. 10 Respuestas: Experimental Vs. Difusa

Tanto la tabla generada en el archivo "motor.out" como las gráficas anteriores nos muestran el desempeño de la solución obtenida. Y considerando los límites impuestos para esta aplicación, se determina que es lo suficientemente optima para cumplir con las expectativas planteadas. Si esto no hubiera sido de esta manera, entonces se deberá tratar con otra solución, es decir, ya sea aumentando el número de datos en el archivo de entrada o incrementado el número de las funciones de membresía.

5.3 Diseño del software

Para poder comprender con mayor claridad como trabaja el sistema difuso que se esta diseñando, antes de entrar por completo en el diseño del software que será implantado en el COP8, se explicará la forma en la cual se calcula la solución a partir de las funciones de membresía.

5.3.1 Reglas de membresía y solución

Todos los cambios realizados en las funciones de membresía son guardados en el archivo "motor.neu". En este archivo se guarda la información de las variables de entrada, la variable de salida, las coordenadas de las funciones de membresía y las reglas que determinan el grado de asociación de las funciones de membresía. El listado del archivo se muestra a continuación:

```
# Fuzzy Rules and Mem Func Approximations for Model "motor"
# Deletion Factor = 0.0000

.DEFUZ neural

.INPUTS VelocidadAngular

.OUTPUTS VoltajeEntrada

.RANGE VelocidadAngular 0.0000 3910.0000

.RANGE VoltajeEntrada 4.2000 10.0000

.LABELS VelocidadAngular Baja Mediana Alta

# Note: x-axis values are normalized to [-1.0,+1.0]
.FUNC VelocidadAngular Baja APPX6 -1.000 -1.000 -1.000 -0.820 0.201 0.939 0.000 0.871 0.234
.FUNC VelocidadAngular Mediana APPX6 -1.000 -0.716 -0.198 0.378 0.910 1.000 0.203 0.910 0.203
.FUNC VelocidadAngular Alta APPX6 -0.806 0.043 1.000 1.000 1.000 1.000 0.206 0.822 0.000

# Rule 1
If VelocidadAngular is Alta then VoltajeEntrada is 12.1618

# Rule 2
If VelocidadAngular is Mediana then VoltajeEntrada is 3.3696

# Rule 3
If VelocidadAngular is Baja then VoltajeEntrada is 4.4308
```

A partir de las coordenadas de las funciones de membresía se puede generar una gráfica para poder apreciar con mayor detalle la aportación de cada función para un punto en especifico del eje de velocidad Wm En la figura 5.11 se muestran las funciones de membresía, se puede observar como se traslapan sus valores entre si.

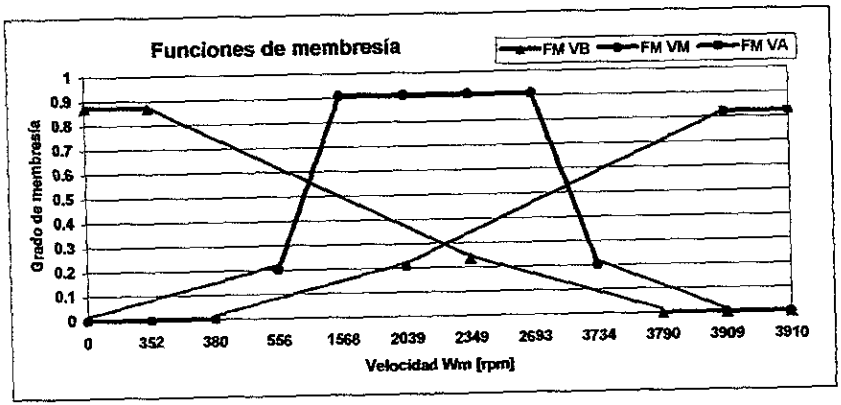


Figura 5. 11 Funciones de membresía VB, VM y VA.

La forma en que NeuFuz calcula el valor esperado de salida comienza con la selección del valor de entrada deseado, después se determina el grado de membresía de cada función afectada por este valor, el grado de membresía encontrado es multiplicado por el peso de la regla que le corresponde y por último se suman los productos encontrados obteniéndose el valor de salida buscado. Tanto las reglas de asociación como los pesos de éstas se localizan en el archivo "motor neu". Para aclarar lo anterior se desarrolla un ejemplo. Considérese el valor deseado Wm=2039 rpm, se traza una línea vertical sobre este punto y las intersecciones con las funciones de membresía son trasladadas en una línea horizontal al eje del grado de membresía obteniéndose los valores:

$$\mu_{WmVB}(2039)=0.333$$

$$\mu_{WmVM}(2039)=0.910$$

$$\mu_{WmVA}(2039)=0.206$$

para las funciones VB (Velocidad Baja), VM (Velocidad Media) y VA (Velocidad Alta) respectivamente. La figura 5.12 muestra el trazado y los puntos encontrados.

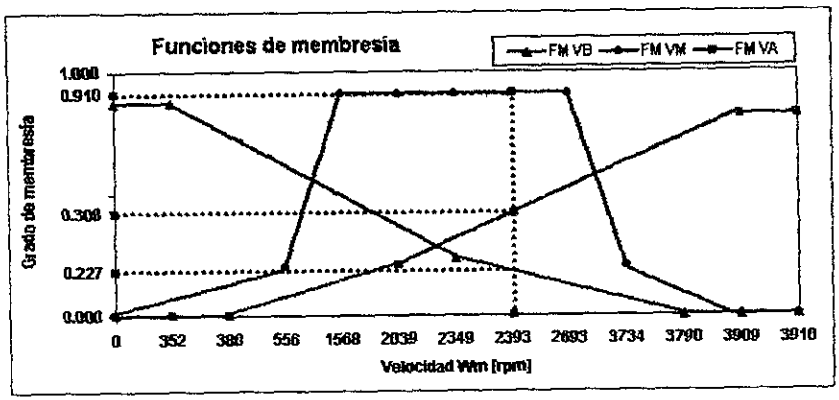


Figura 5. 12 Grados de membresía para el punto Wm=2039.

Haciendo uso de las reglas se tiene lo siguiente:

La regla # 3 dice: **Si** VelocidadAngular es Baja **entonces** VoltajeEntrada es 4.4308
por lo que, la aportación que realiza esta función es (0.227)(4.4308)=1.00579

La regla # 2 dice: **Si** VelocidadAngular es Mediana **entonces** VoltajeEntrada es 3.3696
por lo que, la aportación que realiza esta función es (0.910)(3.3696)=3.06634

La regla # 1 dice: **Si** VelocidadAngular es Alta **entonces** VoltajeEntrada es 12.1618
por lo que, la aportación que realiza esta función es (0.308)(12.1618)=3.74583

Por último la suma de estos productos da el resultado de salida esperado.

$$1.00579 + 3.06634 + 3.74583 = 7.81796$$

Al realizar una comparación con el archivo de entrada "motor.inp" se encuentra que para un valor de Wm = 2393 rpm se tiene un valor de voltaje de entrada de 7.9 Volts. Debido a que el diseño difuso se acerca al comportamiento de la velocidad del motor, en diferentes grados de aproximación, dependiendo del valor de entrada, el error proporcionado también cambia. Para el punto calculado en el ejemplo anterior el error es de 1.05%, al obtener el error para todos los datos del archivo de entrada se encuentra que el error más grande es de 5.250% y el error más pequeño de 0 085%.

5.3.2 Transformación a código ensamblador

El procedimiento de transformación del diseño difuso a código ensamblador es muy sencillo gracias al módulo generador de código incluido en NeuFuz. El generador utiliza el archivo "motor.neu" para producir dos archivos: el archivo "motor.log" que contiene información usada por el COP8, y el archivo "motor.asm" que contiene el código ensamblador que implementa el sistema difuso.

Archivo "motor.log"

El archivo "motor.log" se lista a continuación.

```
Rules    "motor.neu"
Log      "motor.log"
Code     "motor.asm"

Found 3 rules, 1 input, 1 output

 Variable name  VelocidadAngular
   Range min  0.000000 ( h'00 )
   Range max  3910.000000 ( h'FF )
 Mem Functions  Baja  Mediana  Alta
Asm code symbol  I1

 Variable name  VoltajeEntrada
   Range min  4.200000 ( b'0010CCCC )
   Range max  10.000000 ( h'00280000 )
 Weights factor  h'800
Asm code symbol  OUT1  ( low order byte )
          OUT2
          OUT3
          OUT4
          OUT5

ROM usage:
----------
      Code  518
MF Data  MFTBL  27
Rule Data RTBLn  9
```

Total 554 bytes

RAM usage:

BASE RAM 6
REG RAM 3
Stack 8
Other RAM 22
Total 39 bytes

En las primeras líneas se indica el nombre de los archivos que intervienen en el código generado. En las siguientes líneas se numeran la cantidad de reglas, entradas y salidas encontradas en el archivo "motor.neu". En el primer bloque se detalla el nombre de la variable de entrada, el valor máximo y mínimo del rango de valores(en decimal y hexadecimal) que son usados para acondicionar el valor de entrada, el nombre de las funciones de membresía y el símbolo usado en el código ensamblador. En el segundo bloque aparecen el nombre de la variable de salida, el máximo y mínimo de su rango de valores, el factor de peso que es usado para revertir el acondicionamiento realizado en la variable de entrada y por último los símbolos usados por el código ensamblador. El siguiente bloque, el tercero, muestra la cantidad de memoria ROM usada por el diseño difuso, tal como el código, las tablas de datos de las funciones de membresía y de las reglas difusas. El cuarto bloque, el último, dice que cantidad de memoria RAM es usada como base, en los registros, en la pila de memoria y algún otro uso adicional

El archivo "motor.asm"

El archivo "motor.asm" producido por el generador de código esta formado de tres partes: la tabla que contiene la descripción de las funciones de membresía, la tabla que contiene las reglas difusas y el código que permite acceder a las tablas, el proceso de difusificación y el proceso de evaluación de reglas(donde esta integrado el proceso de desdifusificación).

Adicionalmente el archivo "motor.asm" contiene un conjunto de declaraciones de constantes globales que son usadas en el código (en el apéndice B.1 se lista por completo el archivo).

Algunos módulos que intervienen en las distintas partes del código generado se describen a continuación, una tabla completa se da en el apartado 3.3.6.

Módulos de Difusificación

El proceso de difusificación (cálculo del grado de membresía) se realiza antes que el proceso de evaluación de reglas comience. Una tabla que contiene el resultado de éstos cálculos es almacenada en la memoria RAM para ser usada durante la evaluación. Las rutinas contenidas en los módulos FUZZ y MFCODE realizan la difusificación de las entradas del sistema. Los datos del módulo MFTABLE son usados por estas rutinas.

La rutina GOFUZZ llama a las rutinas MFPLUP y MFDCALC para cada función de membresía. La rutina MFPLUP accede a los datos que describen la aproximación de la función de membresía. La rutina MFDCALC calcula el grado de membresía de la variable de entrada basándose en el dato proporcionado por la rutina MFPLUP. El resultado de este cálculo es almacenado en la tabla RAM por la rutina GOFUZZ.

Al completar la difusificación la tabla de entrada difusa en la RAM contiene un byte por cada función de membresía. Cada byte representa el grado de membresía de una entrada en una de sus funciones de membresía. La tabla comienza en la localidad asignada a la variable I1MF1. El tamaño de la tabla es determinado por el número total de funciones de membresía de la solución.

El módulo MFTABLE contiene la tabla de datos que describen la aproximación de las funciones de membresía. La tabla es construida por el generador del código usando las funciones de membresía que fueron editadas previamente. Cada aproximación de las funciones de membresía es descrita por un conjunto de nueve puntos, donde cada punto es asignado a un byte. Todos los puntos (PA a PF) son acondicionados en un rango de 0 a 255 decimal Los puntos H1, H2 y H3 son acondicionados en un rango de 0 a 128 decimal. Los datos para la coordenada X, de PA hasta PF, dividen el rango de valores de entrada en cinco regiones. El cálculo del grado de membresía para un valor de entrada es determinado por la región en la cual el valor de entrada cae. Los datos son almacenados en la tabla de la manera siguiente:

```
.BYTE  PA,PB,PC,PD,PE,PF,H1,H2,H3  ;Input 1 Membership Function 1
.BYTE  PA,PB,PC,PD,PE,PF,H1,H2,H3  ;Input 1 Membership Function 2
.BYTE  PA,PB,PC,PD,PE,PF,H1,H2,H3  ;Input 1 Membership Function 3
```

Cada descripción de las funciones de membresia requiere de nueve bytes de almacenamiento en la memoria ROM. La tabla debe ser almacenada completamente en un bloque de 256-byte en la memoria de programa para facilitar el acceso de los datos vía la instrucción LAID. Por lo tanto, la tabla y la instrucción LAID están localizadas en una sección definida como INPAGE.

Módulos de Evaluación de reglas

Completado el proceso de difusificación de las entradas, el procedimiento de la evaluación de reglas inicia. Por cada regla se realizan las tareas siguientes:

- Se consulta la regla almacenada en memoria.
- Se determina el grado de membresía del antecedente consultando el dato almacenado en memoria RAM durante el proceso de difusificación
- Se determina si cualquier grado de membresia es cero. Si ninguno es cero, el proceso prosigue. De otra forma, se procede con la siguiente regla.
- Se multiplica los antecedentes de la regla (si hay mas de una entrada)
- Se multiplica la salida de la regla (peso) por el producto de los antecedentes
- Se suma la contribución de la regla al valor de salida final

La RULEVAL proporciona el control para el proceso de evaluación de reglas. La rutina RULELUP determina la localidad de almacenamiento, de la memoria de programa, de las reglas difusas. Si varias tablas de reglas son requeridas para el almacenamiento de los datos de las reglas, esto se logra manteniendo el número de página de la regla actual en la memoria de dato (RPAGE) El número RPAGE identifica el módulo RULEn que contiene la regla actual. Además la rutina RULELUP mantiene un número indexado de la regla actual en la memoria de dato (RINDEX) Este número corresponde a la localidad de una regla en la página de la regla actual

La rutina RULELUP llama a la rutina LUPRSn localizada en el módulo RULEn identificado por el número RPAGE. Si solamente una tabla de reglas es usada para almacenar las reglas, entonces RULELUP siempre llama a la rutina LUPRS0. La rutina LUPRSn recupera la regla de la memoria de programa y la almacena temporalmente en la memoria de dato. La rutina LUPRSn debe localizarse en el mismo bloque (256k-byte) de memoria como la tabla RTBLn para permitir el acceso a la información de la tabla de reglas. Por lo tanto, los módulos RULEn son creados como secciones de código INPAGE.

La rutina RDOM consulta la tabla de la entrada difusa en la memoria de datos y toma el grado de membresía de cada antecedente de la regla.

La rutina COUT multiplica el grado de membresía de los antecedentes y entonces multiplica el resultado por el peso de la regla. El resultado de la multiplicación por el peso es sumado al resultado final de salida.

Después que todas las reglas son procesadas, la salida final es almacenada en un número de complemento dos de 5-bytes en la localidad de memoria de datos asignada a los valores OUT1 a OUT5. El número de salida es reacondicionado por dos factores: un factor de 128 que es introducido en el acondicionamiento de las coordenadas de la función de membresía, y un factor específico para la aplicación que es usado para reacondicionar el peso de salida. El factor completo para la salida de la aplicación es calculado por la multiplicación del factor de escala mostrado en el archivo "motor.log" por 128.

Tablas de reglas

Los datos de las reglas difusas para el sistema están divididos en tablas de reglas para facilitar la consulta de los datos por la instrucción LAID. Un nuevo módulo es creado por el generador de código por cada tabla de reglas. Estos módulos son llamados RULE0, RULE1, RULE2,...RULEn Cada módulo contiene solo una tabla de reglas y una rutina de búsqueda, LUPRSn. El número de módulos RULEn es determinado por el número de reglas difusas en el archivo "motor.neu". El número de reglas por módulo RULEn es determinado por el número de antecedentes en cada regla.

Módulo de operaciones (MATH)

El módulo de matemáticas contiene un grupo de rutinas de multiplicación y división usadas en el proceso de evaluación de reglas. Algunas rutinas de multiplicación son proporcionadas principalmente para dismimuir el tiempo de ejecución requerido para la evaluación de las reglas. La memoria de uso reservada para estas rutinas esta cuidadosamente escogida y no debe cambiarse. Estas rutinas pueden ser usadas por el código de la aplicación tanto como por los procesos de difusificación y de evaluación de reglas.

Módulos asignados en el almacenamiento de datos(FUZDAT, FAST, TOP)

La memoria de datos del COP8 esta dividido en varias secciones. Cada sección tiene características especiales que están hechas específicamente para diferentes tipos de operación. Por lo tanto, las localidades de memoria utilizadas por el código difuso están distribuidas entre tres diferentes secciones basadas en el tipo de uso.

Las variables en la sección FUZDAT son localizadas en la sección de RAM común. Algunas de estas variables están localizadas en la misma localidad de memoria. Esto reduce la cantidad total de la memoria de datos usada por el código difuso. Las variables que comparten una localidad de memoria nunca son usadas simultáneamente por el código difuso.

Las variables FAST están localizadas en la sección BASE de la memoria RAM. Los datos almacenados en esta sección pueden ser consultados muy rápidamente con el puntero B. Por lo tanto, todos los registros para las rutinas de multiplicación y división están localizados en esta sección. Estos registros son usados frecuentemente y el tiempo de ejecución es reducido al colocar estos en la sección BASE. Las mismas localidades de memoria son asignadas para los registros usados para las diferentes rutinas de multiplicación y la rutina de división. La posición relativa de estos registros a cualquier otro es muy importante. No se deben alterar la asignación de estos registros.

Las variables TOP son asignadas a la sección REG de la RAM. Algunas instrucciones pueden operar solamente en operaciones localizadas en esta sección de la memoria de datos. Por lo que, las

variables asignadas a la sección REG no deben ser movidas a otra sección de la RAM. El mover estas variables puede causar errores al momento de ensamblar el código.

5.3.4 Complementación del código ensamblador

Para tener el código de la aplicación completo falta incluir el programa principal que tendrá la función de proporcionar el valor de las entradas y hacer el llamado a las rutinas del proceso de difusificación-desdifusificación. Además debe proporcionar los datos necesitados por el DAC0800, recibir los datos del ADC0804 y enviar la información necesitada por el programa encargado de monitorear el comportamiento de la solución. El diagrama de la figura 5.13 muestra el flujo seguido por el programa principal.

El programa principal empieza configurando los puertos que transmitirán y recibirán los distintos datos usados. El puerto C (bidireccional), es configurado en sus bits C0-C1 como de lectura, y los bits C2-C3 como de escritura, esto se logra colocando en el registro de puerto (PORTCC) un valor de F6 hex. El puerto D (de solo escritura), que esta encargado de controlar el DAC es configurado con un valor de 00 hex que inicializa el convertidor. El registro de configuración PORTLC es puesto a un valor de 00 hex, configurando al puerto L (bidireccional) por completo como de lectura.

Ya que se han configurado los puertos el programa entra en un ciclo esperando el dato del valor de velocidad angular deseada, que es proporcionado por el programa monitor Cuando se recibe el aviso de que el dato esta listo, el puerto F es configurado como de lectura, colocando en el registro PORTFC un valor de 00 hex y un valor de 00 hex en el registro PORTFD El valor del puerto de lectura (PORTFP) se lee y es pasado a 11, es solo entonces cuando la rutina FUZZ del código difuso es llamada para obtener el valor de salida.

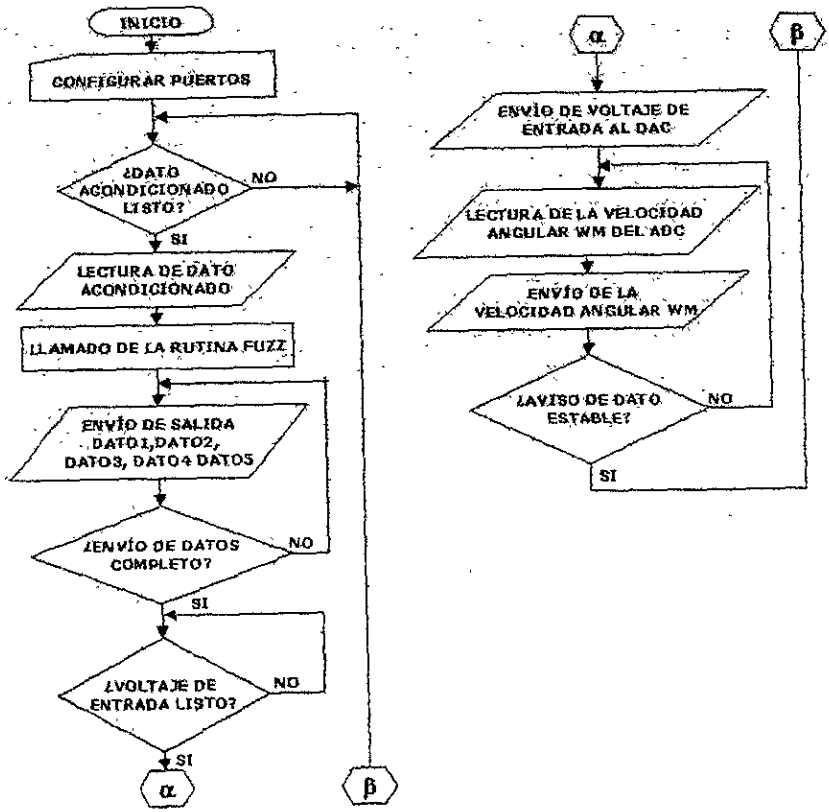


Figura 5.13 Diagrama de flujo programa principal.

Mientras la rutina FUZZ determina el valor de salida el programa monitor se queda esperando la respuesta. Cuando el valor de salida se encuentra listo envía un aviso al programa monitor para que reciba el resultado. El valor proporcionado por la rutina FUZZ esta compuesto por cinco bytes que envía uno a uno al programa monitor por medio del puerto paralelo de la computadora donde reside dicho programa. Mientras se calcula el valor del voltaje de entrada el COP8 queda en modo de espera.

El valor del voltaje de entrada es entonces regresado al COP8 para que sea pasado del DAC a la entrada del motor para, de esta forma, controlar la velocidad angular del motor. El programa

monitor aguarda el valor actual de la velocidad angular del motor, el cual es proporcionado por el ADC vía el COP8 Una rutina del programa monitor se ejecuta hasta que la velocidad angular del motor se estabilice para obtener el resultado final.Cuando se logra un valor estable el programa principal de la aplicación residente en el COP8 vuelve a comenzar su rutina en espera de otro dato.

5.3.5 Programa de monitoreo

El programa de monitoreo, realizado en C++, se encarga de enviar y recibir los datos que el COP8 utiliza para obtener la solución del problema difuso y el control de los demás periféricos. Esto se logra utilizando los pines del puerto paralelo: 2-9 para datos de entrada/salida, 10-11 como dos bit de lectura para recibir confirmación del COP8, 14 y 17 como dos bits de escritura para indicar al COP8 que puede leer el dato solicitado, el pin 16 para controlar el receptor/transmisor (74LS245) permitiendo el envío de datos del COP8 al puerto paralelo o viceversa y el pin 18 como conexión a tierra (GND). También en este proceso se hace uso de los registros de software: de datos(0x378=BASE), de estado (BASE+1)y de control (BASE+2). En el apéndice C.4 se detallan los pines y los registros de software La figura 5.14 muestra un diagrama de flujo del programa monitor.

El programa monitor solicita el valor de velocidad angular deseada y la envía hacia el COP8, mediante el puerto de datos. Este dato es acondicionado previamente por dicho programa de acuerdo a la siguiente expresión:

$$Dato\_Acondicionado = \frac{(Wm - min) \times 256}{(max - min)}$$

donde min y max son los extremos del rango del valor de entrada, que pueden consultarse en el archivo "motor.log".

Una vez que el programa monitor envía el dato acondicionado aguarda hasta que el COP8 le informa que ya tiene el resultado Entonces el programa monitor empieza la lectura de los cinco bytes que conforman a la respuesta esperada El programa monitor se encarga de reacondicionar el valor de salida para retirar el acondicionamiento, establecido previo a la rutina de difusificación mediante la expresión:

$$Voltaje\_entrada = \frac{Valor\_Salida\_Acondicionado}{128 \times FactorPeso}$$

donde el factor de peso puede ser consultado en el archivo "motor.log"

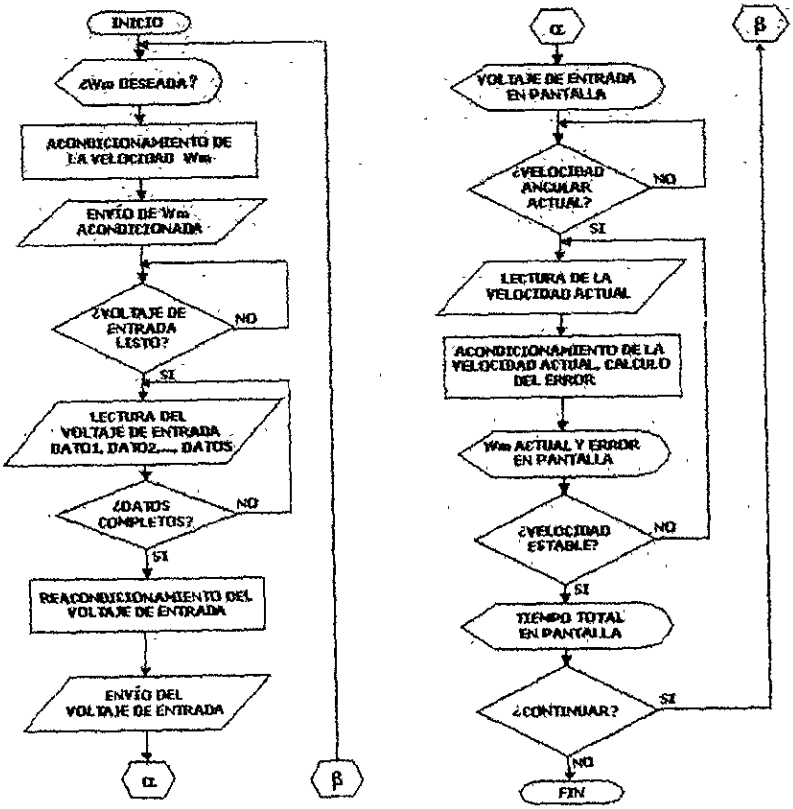


Figura 5.14 Diagrama de flujo del programa monitor.

Hasta este momento el programa monitor muestra en pantalla la velocidad angular deseada y el voltaje de entrada calculado. El voltaje de entrada es ahora acondicionado para ser enviado al COP8 para controlar el DAC, mientras tanto el programa espera la lectura de la velocidad actual en el motor debido al nuevo valor de voltaje de entrada. Cuando recibe el aviso de la velocidad

angular entra en una rutina que espera a que el motor estabilice su velocidad, le indica al COP8 que espere el envío de un nuevo dato y finalmente el programa monitor muestra en pantalla el error porcentual de la velocidad angular deseada con respecto al valor de velocidad actual y el tiempo ocupado en alcanzar dicha condición. El listado del programa se encuentra en el apéndice D.

5.4 Descripción del hardware

Para lograr el control de la velocidad angular del motor se hace uso de otros dispositivos. Además del COP8 se utilizan los circuitos integrados: DAC0800 (Convertidor Analógico Digital), un DAC0804 (Convertidor Digital Analógico), un Transmisor/Receptor de ocho bits 74LS245 y un circuito inversor TTL 74LS04.

5.4.1 DAC0800

El DAC0800 produce un voltaje analógico en su salida que corresponde directamente con el valor de entrada binario de ocho bits que presenta en su entrada. El circuito integrado contiene en su interior una red escalera y circuitos activos. En donde por cada bit se controla un conmutador que suministra corriente de referencia a la red escalera. La salida de la red escalera es aislada por medio de un amplificador que proporciona alta ganancia a la salida. El DAC0800 es de corriente de salida y de alta velocidad, ya que su tiempo de establecimiento es de 100nS. Gracias a una corriente de acoplamiento indicada a la escala total, se elimina la necesidad de potenciómetros de ajuste en la mayoría de las aplicaciones. Sus entradas son inmunes al ruido y aceptan niveles TTL colocando el pin 1 (V_{LC}), control de umbral, a tierra. Tanto el funcionamiento como las características del DAC se mantienen sin cambios significativos respecto al rango total del voltaje de alimentación ±4.5 a ±18 V. El error que presenta el dispositivo a escala total es de ±1 LSB. La figura 5.15 muestra el diagrama de conexiones usado en esta aplicación.

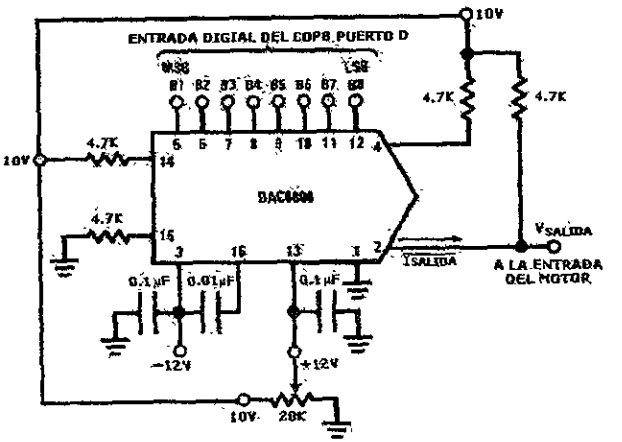


Figura 5.15 Diagrama de conexión del DAC0800.

5.4.2 ADC0804

El convertidor ADC0804 pertenece a la familia de convertidores de ocho bits, construidos sobre la base de tecnología CMOS, que utilizan el método de aproximaciones sucesivas para obtener la conversión. Una gran cualidad de estos convertidores es que parecen al microprocesador como localidades de memoria o puertos de E/S por lo que no es necesaria lógica de interfase La entrada de voltaje de referencia puede ajustarse para permitir codificar el tramo más pequeño de voltaje analógico a la resolución total de 8 bits. El rango de voltaje de entrada es de 0 a 5V y una alimentación única de 5V Además cuenta con un generador de reloj integrado, presenta un error total de ±1 LSB y ocupa un tiempo de conversión de 100µS.

Para que inicie la conversión el $\overline{CS}$ (Selector de Chip) y $\overline{WR}$ (Escritura) deben tener un nivel bajo en sus terminales. Después de terminar la conversión se transfiere el resultado a los biestables de salida y se reporta el termino de la conversión con un cambio de alto a bajo en $\overline{INTR}$ (Interrupción). Es entonces cuando puede efectuarse la lectura al habilitar a los biestables de salida por medio de un cero lógico en $\overline{RD}$. La figura 5 16 muestra el diagrama de conexiones usado en esta aplicación.

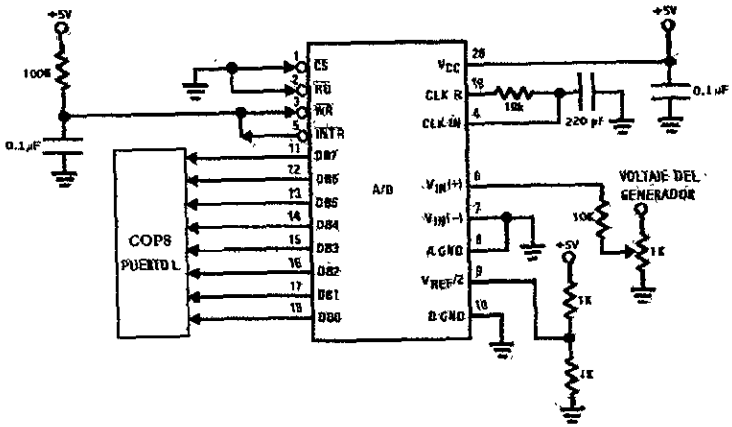


Figura 5.16 Diagrama de conexiones del ADC0804.

5.4.3 Trasnmisor/Receptor SN74LS245

El SN74LS245 es un Transmisor/Receptor de ocho bits, diseñado para permitir la comunicación entre buses de datos en dos direcciones. El pin DR controla la dirección de transmisión de los datos del bus A al bus B o viceversa, dependiendo su nivel lógico. La entrada E puede ser usada para aislar los buses. El voltaje de alimentación es el típico en circuitos lógicos (Vcc = 5V). El control de transmisión se rige por la siguiente tabla de verdad:

ENTRADAS		SALIDA
E	DIR	
L	L	Dato de bus B al bus A
L	H	Dato de bus A al bus B
H	X	DESHABILITAR

H – Nivel de voltaje alto

L – Nivel de voltaje bajo

X = No importa

El SN74LS245 se utiliza para permitir la conexión entre el puerto paralelo y el COP8, ya que es bidireccional puede enviar o recibir información entre ambos dispositivos. El pin DR dirige la

dirección del flujo de los datos y es controlado por la terminal 16 del puerto paralelo que corresponde al bit 2 del puerto de control de dicho puerto. La figura 5.17 muestra el diagrama de conexión utilizado.

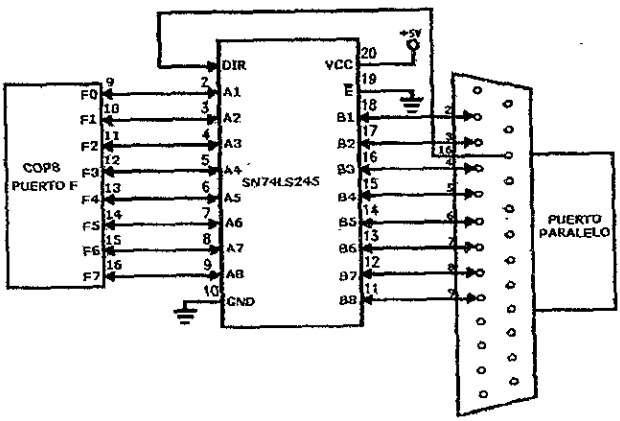


Figura 5.17 Diagrama de conexión del SN74LS245.

5.5 Ajuste y Evaluación del sistema

Ya con todos las partes listas: el COP8 y sus programas de operación, el Transmisor/Receptor, los convertidores D/A y A/D, la PC donde reside el programa de monitoreo y el puerto paralelo, el motor y sus fuentes de alimentación, se realiza la interconexión de todos ellos para realizar el trabajo de evaluación y depuración, para mejorar el desempeño, del sistema. Algunos ajustes adicionales se hacen en los potenciómetros preajustables de los módulos de los convertidores para obtener una respuesta lo más precisa que sea posible. El diagrama esquomático de la figura 5.18 muestran las conexiones utilizadas en este proceso.

Debido a que algunos pines del puerto paralelo presentan en sus terminales un valor invertido de hardware se ha incluido en el diseño el uso del inversor 74LS04, los pines que presentan esta característica son el 11, el 14 y el 17. Esta consideración resulta conveniente si no se desea que se presente confusión al momento de que se envía un determinado dato y el valor que realmente presenta en sus terminales.

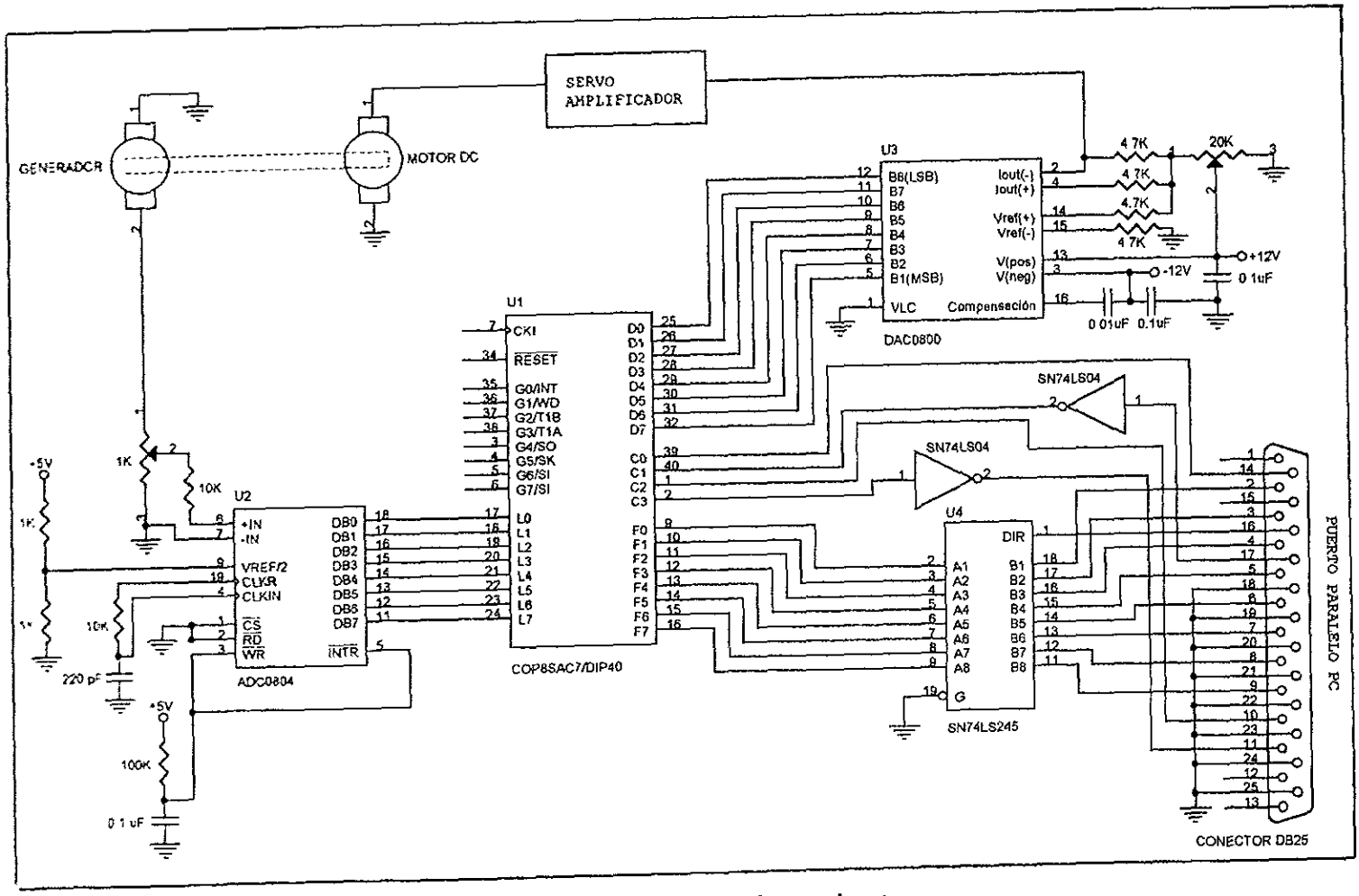


Figura 5.18 Diagrama completo de conexiones.

Para alimentar a los circuitos se utilizan tres fuentes de voltaje de CD. Una fuente de 5 Volts y una fuente bipolar con voltajes en sus terminales de ±12 Volts. Para obtener el voltaje de +10 en el circuito del DAC, se usa un preset de 20KΩ para formar un divisor de voltaje. Así mismo, para acondicionar el voltaje provemente del generador se utiliza otro potenciómetro preajustable a la entrada de voltaje del ADC. La figura 5.19 muestra las conexiones en la tableta de prueba y la figura 5 20 al sistema por completo.

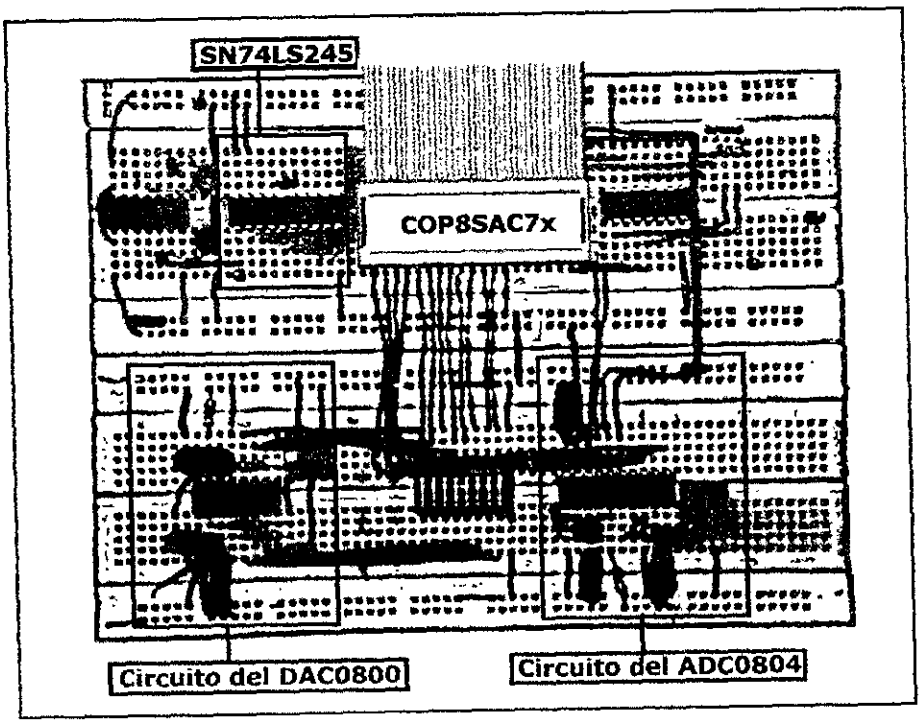


Figura 5.19 Tableta de conexiones.

Tanto el circuito del DAC como el ADC se ajustan para proporcionar el valor deseado en esta aplicación Es decir, el rango de valores de voltaje de salida del DAC debe ser el parecido al que se obtuvo de las caracteristicas del motor, de 4 3 a 10 Volts Esto significa que el valor más pequeño del rango de velocidad 396 rpm debe corresponder a 4 3 Volts, de forma similar el valor más grande 3910 rpm debe corresponder a 10 Volts En lo que toca al ADC el rango de voltaje que

recibe en la entrada del divisor de voltaje es de 0 a 11.73 Volts aproximadamente o lo que es lo mismo 0rpm a 3910. Lo que realmente proporciona el ADC es un conjunto de valores de 0 a 255, que al pasar a la PC son transformados en términos de la velocidad angular Wm. La fórmula utilizada para esto es:

$$Wm = \frac{valor\_puerto \times 11.7}{255 \times 0.003}$$

donde 0.003 es el factor de conversión que proporciona el generador.

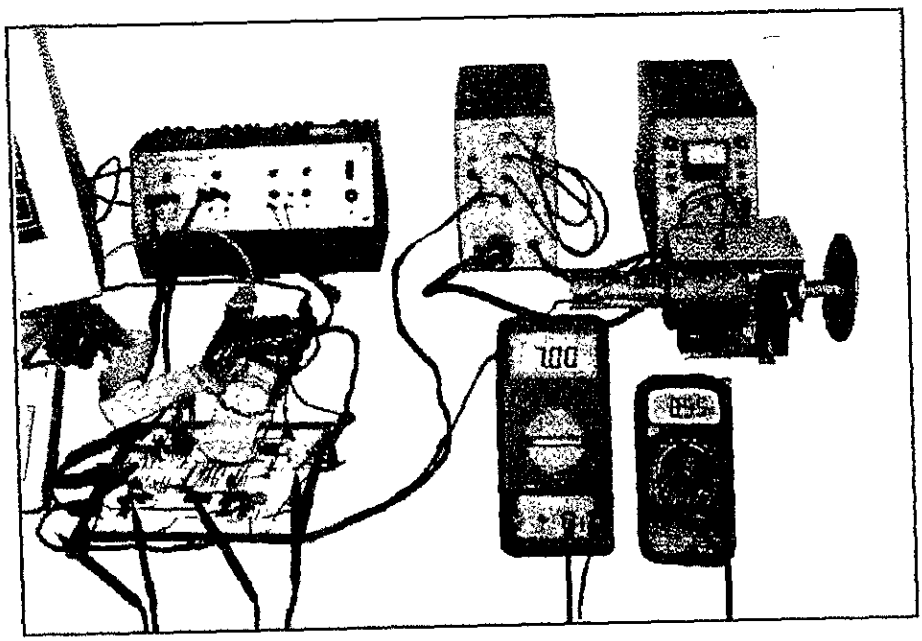


Figura 5.20 Diagrama completo del sistema.

Es importante anotar que los convertidores poseen un cierto grado de error, además del redondeo que se realiza al trabajar con una cierta resolución (ocho bits), y que el paso del valor a través de ellos no es tan exacto como se quisiera. Por lo anterior y con el objetivo de apreciar con mayor detalle el resultado se colocan dos voltímetros: uno a la salida del DAC (voltaje de entrada) y otro a la salida del generador.

Una vez que se a alimentado el circuito y el motor, el programa residente en el COP8 es puesto a andar y lo primero que hace es esperar que el programa de monitoreo le envíe el valor de la velocidad angular Wm propuesta. Se ejecuta el programa de monitoreo y pide un valor de Wm para operar el sistema.

Se realizaron un conjunto de pruebas, dentro del rango de valores conocidos, de 396 rpm a 3910rpm, tal y como lo muestra la siguiente tabla:

	Valor deseado	Valor obtenido	%error
1	396	381	3.78
2	787	792	-0.63
3	1178	1102	6.45
4	1569	1630	-3.88
5	1960	1840	6.12
6	2351	2387	-1.53
7	2351	1840	0.98
8	2742	2715	-2.14
9	3324	3397	-2.19
10	3910	3960	-1.28

Como puede observarse el error que se presenta va de 0.63% a 6 45% dentro de este conjunto de valores El rango de error que presento la solución en el archivo "motor.out" fue [0.085%-5 25%] Como se puede apreciar el error aumento un poco, debemos anotar, también, que los valores de los voltímetros sí muestran los valores esperados a la salida del DAC y del generador. Esto quiere decir que el motor si se esta comportando de acuerdo al diseño difuso y que el incremento del error se encuentra en el paso del ADC al COP8 Esto puede corregirse aumentando la resolución del convertidor ADC y adecuando la programación del COP8 para procesar este valor hacia la PC. A pesar de esto, el sistema si responde al diseño difuso tal y como se había planteado en el diseño inicial.

5.6 Conclusiones

En esta aplicación se comprueba que la lógica difusa es una buena alternativa. La naturalidad con la cual se puede describir un sistema, no importando lo complejo que sea éste, representa una gran ventaja. El tratar con relaciones lingüísticas y no con expresiones complejas es algo a destacar. La obtención de las funciones de membresía, del diseño difuso, realmente no fue ningún problema. La evaluación de la solución es clara y sin complicaciones, y las adecuaciones al diseño son realizadas muy rápidamente. Y si añadimos que la implantación en el microcontrolador es casi inmediata, entonces, las prestaciones que da el diseño difuso representan una buena opción a considerar cuando se este diseñando un sistema de control.

Sin embargo, la lógica difusa presenta algunos inconvenientes, que dependiendo la aplicación deberán ser tomados muy en cuenta, como la necesidad de contar con un operador del sistema de control que conozca las características propias del sistema, ¿Qué tanto debe conocer un experto del sistema?. La respuesta dependiendo el sistema puede ser ambigua. La exigencia de contar con un conjunto de datos, para de ahí partir hacia la obtención de la solución. ¿Y si la obtención del conjunto de datos resulta más complicada que la solución de control misma?. Y quizás lo más importante la falta de garantía en cuanto si el sistema es o no lo suficientemente robusto, ya que situaciones no contempladas en el conjunto de datos pueden ocasionar que el sistema se vuelva simplemente inestable.

Entonces, ¿Cuándo y donde se debe usar la lógica difusa?. Creo que la respuesta es: En aquellos problemas que no han encontrado solución con los métodos tradicionales y donde no se haya realizado aún un diseño con las técnicas clásicas. Es decir, si ya existe un diseño de control clásico y cumple de manera optima y funcional con los requerimientos exigidos, entonces no hay necesidad de aplicar un diseño de lógica difusa. Creo que la pauta la da el mismo sistema y los resultados que se hayan tenido al momento de tratar de resolver el diseño con control clásico. Generalizando se puede decir que "Mientras más grande y complejo sea un sistema, entonces, un diseño con lógica difusa es lo más recomendable".

Por otro lado el COP8 es un dispositivo excelente, como se puede apreciar en este diseño los circuitos de los convertidores usan más elementos que el mismo microcontrolador. El fácil manejo de operación y programación hacen de él una muy buena opción si lo que se desea es optimizar el tamaño del circuito final. La variedad que presenta de opciones de configuración de temporización, de interrupciones y de los monitores de operación, y de reloj lo proveen de una gran flexibilidad para una gran cantidad de aplicaciones.

Adicionalmente una de las ventajas es la posibilidad de crear código ensamblador a partir de herramientas de software existentes en el mercado, esto permite un gran ahorro de trabajo y de tiempo al momento de la, a veces, pesada tarea de programación. Otra, es la existencia de módulos de evaluación, que los hay muy completos, y que ayudan enormemente a la depuración y evaluación, del trabajo en cuestión.

Por último, el hacer uso de la técnica de la lógica difusa por medio de un microcontrolador es una idea con mucho sentido, ya que la capacidad del dispositivo (memoria) no esta peleada con la capacidad que necesita una solución de lógica difusa. Es más esta mancuerna (microcontrolador-lógica difusa) puede ser explotada en muchos y variados caminos, dependiendo solo del ingenio del diseñador.

Apéndice A Conjunto de Instrucciones

ADD — Suma

Sintaxis: a) ADD A,[B]
b) ADD A,MD
c) ADD A,#

Descripción: El contenido de la localidad de memoria indicada por

a) el puntero B
b) la dirección en el segundo byte de la instrucción
c) el valor inmediato del segundo byte de la instrucción

es sumado al contenido del acumulador, y el resultado es puesto en el acumulador.

Operación. A ←A + VALOR

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
ADD A,[B]	Registro Indirecto(B Puntero)	1	1	84
ADD A,#	Inmediato	2	2	94/Imm #
ADD A,MD	Memoria Directa	4	3	BD/MA/84

ADC— Suma con acarreo

Sintaxis: a) ADC A,[B]
b) ADC A,#
c) ADC A,MD

Descripción: El contenido de

a) la localidad del dato de memoria indicada por el puntero B
b) el valor inmediato en el segundo byte de la instrucción

c) la localidad del dato de memoria indicada por el segundo byte de la instrucción.

Son sumados al contenido del acumulador, y el resultado es simultáneamente incrementado si la bandera de acarreo se encuentra establecida.

Operación: A ← A + VALOR + C

C ← CARRY; HC ←HALF CARRY

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
ADC A,[B]	Registro Indirecto(B Puntero)	1	1	80
ADC A,#	Inmediato	2	2	90/Imm #
ADC A,MD	Memoria Directa	4	3	BD/MA/

AND — Y

Sintaxis: a) AND A,[B]
b) AND A,#
c) AND A,MD

Descripción: Una operación AND es realizada en los bits correspondientes del acumulador y

a) el contenido de la localidad de memoria indicada por el puntero B
b) el valor inmediato encontrado en el segundo byte de la instrucción
c) el contenido de la localidad de memoria es indicada por la dirección en el segundo byte de la instrucción.

El resultado reemplaza el contenido del acumulador.

Operación: A←A AND VALOR

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
AND A,[B]	Registro Indirecto(B Puntero)	1	1	85
AND A,#	Inmediato	2	2	95/Imm.#
AND A MD	Memoria Directa	4	3	BD/MA/85

ANDSZ – Y lógico, Saltar si Cero

Sintaxis: ANDSZ A,#

Descripción: Una operación AND es realizada en los bits correspondientes del acumulador y el valor inmediato que se encuentra en el segundo byte de la instrucción. Si el resultado es cero, la siguiente instrucción es omitida. El acumulador permanece sin cambios. Esta instrucción es usada para comprobar algún bit seleccionado en el acumulador. La máscara en el segundo byte es usada para seleccionar cual bite es probado.

Operación: SI (A Y #) = 0, ENTONCES BRINCA LA SIGUIENTE INSTRUCCIÓN

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
AND A,#	Inmediato	2	2	60/Imm.#

CLR – Limpiar el Acumulador

Sintaxis: CLR A

Descripción: El acumulador es puesto a cero.

Operación: A←0

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
CLR A	Implícito	1	1	64

DCOR – Ajustar decimal

Sintaxis: DCOR A

Descripción. Esta instrucción cuando es usada después de una instrucción ADC (suma con acarreo) o una instrucción SUBC (substracción con acarreo) ajusta el decimal resultante de la suma o resta binaria. Nótese que la instrucción ADC debe de ser precedida con una instrucción ADD A, #066 (sumar el hexadecimal 66) para la corrección de la suma decimal. Esta instrucción asume que los dos operandos se

encuentran en el formato BCD y produce el resultado en el mismo formato BCD. Las banderas de Acarreo y Medio Acarreo permanecen sin cambio.

Operación: A (FORMATO BCD) ← A(FORMATO BINARIO)

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
DECOR A	Implícito	1	1	66

DEC – Disminuir el acumulador

Sintaxis: DEC A

Descripción. Esta instrucción disminuye el contenido del acumulador y coloca el resultado de regreso al acumulador. Las banderas de acarreo y medio acarreo permanecen sin cambio.

Operación: A ← A-1

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
DEC A	Implícito	1	1	8B

DRSZ REG# - Disminuir registro y saltar si el resultado es cero

Sintaxis: DRSZ REG#

Descripción: Esta instrucción disminuye el contenido del registro de memoria seleccionado (seleccionado por #, donde #= 0 a F) y coloca el resultado en el mismo registro. Si el resultado es cero, la siguiente instrucción es saltada. Esta instrucción es utilizada cuando se necesita repetir una secuencia varias veces. El número de veces que se

desea repetir una secuencia se encuentra en el registro, y una instrucción DRSZ con ese registro es codificada al final de la secuencia, seguida de una instrucción de salto (JP salto relativo) que regresa al inicio de la secuencia. La instrucción de salto JP es ejecutada cada vez que se pasa por la secuencia hasta que el registro de cuenta decrece a cero; en ese momento la instrucción de salto JP es omitida y el programa termina la secuencia.

Operación: REG ← REG-1
SI (REG – 1)= 0 , ENTONCES SALTAR LA SIGUIENTE INSTRUCCION

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
DRSZ REG#	Registro Directo (Implícito)	3	1	C(REG#)

IFBIT – Comprobar BIT

Sintaxis: a) IFBIT #,[B]
b) IFBIT #,MD
c) IFBIT #,A

Descripción: El bit seleccionado (# = 0 a 7) de

a) la localidad de memoria indicada por el puntero B es comprobado.
b) la localidad de memoria indicada por la dirección en el segundo byte de la instrucción es comprobado.
c) el acumulador es comprobado

Si el bit escogido esta en alto (=1), entonces la siguiente instrucción es ejecutada. De otra forma, la siguiente instrucción es saltada

Operación: SI EL BIT (#) SELECCIONADO ES IGUAL A CERO,
ENTONCES SALTAR LA SIGUIENTE INSTRUCCIÓN.

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
IFBIT #,[B]	Registro Indirecto (Puntero B)	1	1	7#
IFBIT #,MD	Memoria Directa	4	3	BD/MA/7#
IFBIT #,A	Inmediato	2	2	60/2#

NOTA: La instrucción IFBIT #,A es un subconjunto mas generalizado que la instrucción ANDSZ y comparte el mismo código (60). Esta instrucción se desensambla dentro de la instrucción ANDSZ.

Instrucciones equivalentes:

IFBIT 0,A = ANDSZ A,#1
IFBIT 1,A = ANDSZ A,#2
IFBIT 2,A = ANDSZ A,#4
IFBIT 3,A = ANDSZ A,#8
IFBIT 4,A = ANDSZ A,#16
IFBIT 5,A = ANDSZ A,#32
IFBIT 6,A = ANDSZ A,#64
IFBIT 7,A = ANDSZ A,#128

IFBNE # - Si el puntero B no es igual

Sintaxis: IFBNE #

Descripción: Si el nibble bajo del puntero B no es igual a # (donde # = 0 a F), entonces la siguiente instrucción es ejecutada. De otra manera, la siguiente instrucción es saltada. Esta instrucción es utilizada cuando el puntero B es enviado a través del campo de datos como una parte de una secuencia cíclica. La instrucción IFBNE es

codificada al termino de la secuencia, seguida por una instrucción JP que envía de regreso al comienzo de la secuencia. El código # con la instrucción IFBNE representan la siguiente instrucción fuera del campo de datos. La instrucción del puntero B con post-incrementos o decrementos del puntero puede ser usada en el paso a través del campo de datos en cualquier dirección. La secuencia se repite hasta que el nibble bajo del puntero B es igual al # (representando la siguiente dirección fuera del campo de datos), al mismo tiempo que la instrucción JP salta fuera del ciclo.

Operación: SI EL NIBBLE BAJO DEL PUNTERO B ES IGUAL A #,
ENTONCES SALTAR LA SIGUIENTE INSTRUCCIÓN

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
IFBNE #	Implícito	4	1	4#

IFC – Comprobar si hay acarreo

Sintaxis: IFC

Descripción: La siguiente instrucción es ejecutada si la bandera de acarreo esta establecida. De otra forma la siguiente instrucción es saltada. La bandera de acarreo queda sin cambio.

Operación: SI NO HAY ACARREO (C=0),
ENTONCES SALTAR LA SIGUIENTE INSTRUCCIÓN

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
IFC #	Implícito	1	1	88

IFEQ – Comprobar igualdad

Sintaxis: a) IFEQ A,[B]
b) IFEQ A,#
c) IFEQ A,MD
d) IFEQ MD,#

a) El contenido de la localidad de memoria de dato indicada por el puntero B es comparado con el contenido del acumulador.

b) El valor inmediato encontrado en el segundo byte de la instrucción es comparado con el contenido del acumulador.

c) El contenido de la localidad de memoria de dato indicada por la dirección en el segundo byte de la instrucción es comparada con el contenido del acumulador.

d) El contenido de la localidad de memoria indicada por la dirección en el segundo byte de la instrucción es comparada con el valor inmediato encontrado en el tercer byte de la instrucción.

Una comparación con éxito de igualdad provoca la ejecución de la siguiente instrucción. De otra manera, la siguiente instrucción es omitida.

Operación: SI EL CONTENIDO DE LA LOCALIDAD ESPECIFICADA VALOR #,
ENTONCES LA SIGUIENTE INSTRUCCIÓN ES SALTADA

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
IFEQ A,[B]	Registro Indirecto	1	1	82
IFEQ A,#	Inmediato	2	2	92/Imm,#
IFEQ A,MD	Memoria Directa	4	3	BD/MA/82
IFEQ MD,#	Memoria Directa, Inmediato	3	3	A9/MA/Imm,#

IFGT – Comprobar si el acumulador es más grande que..

Sintaxis: a) IFGT A,[B]
b) IFGT A,#
c) IFGT A,MD

Descripción: El contenido del acumulador es probado para saber si es más grande que

a) el contenido de la localidad de memoria indicada por el puntero B.
b) el valor inmediato encontrado en el segundo byte de la instrucción.
c) el contenido de la localidad de memoria indicada por la dirección en el segundo byte de la instrucción.

Si el acumulador resulta mayor, se produce la ejecución de la siguiente instrucción. De otra manera, la siguiente instrucción es brincada.

Operación. SI A ≤ VALOR,
ENTONCES SALTAR LA SIGUIENTE INSTRUCCIÓN

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
IFGT A,[B]	Registro Indirecto(Puntero B)	1	1	83
IFGT A,#	Inmediato	2	2	93/Imm,#
IFGT A,MD	Memoria Directa	4	3	BD/MA/83

IFNC – Comprobar si no existe acarreo

Sintaxis: IFNC

Descripción. La siguiente instrucción es ejecutada si la bandera de acarreo se encuentra establecida.. De otra forma, la siguiente instrucción es saltada. La bandera de acarreo permanece sin cambio.

Operación: SI EL ACARREO (C=1),
ENTONCES SALTAR LA SIGUIENTE INSTRUCCIÓN.

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
IFNC	Implícito	1	1	89

IFNE – Comprobar la desigualdad

Sintaxis: a) IFNE A,[B]
b) IFNE A,#
c) IFNE A,MD

Descripción: a) El contenido de la localidad de memoria de dato indicada por el puntero B es comparada para desigualdad con el contenido del acumulador.
b) El valor inmediato que se encuentra en el segundo byte de la instrucción es comparado para la desigualdad con el contenido del acumulador.
c) El contenido de la localidad de memoria de dato indicada por la dirección en el segundo byte de la instrucción es comparada para desigualdad con el contenido del acumulador.

Una comparación de desigualdad exitosa resulta en la ejecución de la siguiente instrucción; de otra forma, la siguiente instrucción es saltada.

Operación: SI A = VALOR,
ENTONCES SALTAR LA SIGUIENTE INSTRUCCIÓN.

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
IFNE A,[B]	Registro Indirecto(Puntero B)	1	1	B9
IFNE A,#	Inmediato	2	2	99/Imm,#
IFNE A,MD	Memoria Directa	4	3	BD/MA/B9

INC – Incremento del Acumulador

Sintaxis: INC A

Descripción: Esta instrucción incrementa el contenido del acumulador y regresa el resultado al acumulador. La bandera de acarreo y medio acarreo permanecen son cambio.

Operación: A ← A + 1

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
INC A	Implícito	1	1	8A

INTR – Interrupción (Trampa de Software)

Sintaxis: INTR

Descripción Esta instrucción de interrupción de software almacena su dirección de retorno en la pila de memoria de software y entonces se bifurca a la localidad de memoria de programa 00FF. Esta localidad de memoria es el punto de conmutación normal para todas las interrupciones del COP8, tanto para hardware como software. El programa inicia en la localidad de memoria 00FF manejando la salida por prioridad de varias interrupciones y entonces dirige correctamente la rutina de servicio de interrupción

A fin de salvar la dirección de retorno, el contenido de PCL (los 8 bit bajos de PC) son transferidos a la localidad de memoria de dato, indicada por SP(puntero de pila), el SP es entonces decrementado. El contenido de PCU (los 7 bits superiores del PC) son transferidos a la nueva localidad de memoria indicada por SP. Entonces SP es de nueva cuenta decrementado para establecer la pila de software para la siguiente interrupción o subrutina.

La instrucción INTR no significa que sea explícitamente programada, pero más que sea automáticamente invocada cuando cierta condición de error ocurre. La lectura de memoria de programa no definida (no existente) produce ceros, lo que provoca el llamado de la instrucción INTR. De forma similar una trampa de software puede establecerse si la subrutina SP es inicializada para la localidad de memoria de dato en la parte alta del espacio de memoria RAM. Entonces si la pila del software esta siempre saturada (hay más regresos de subrutinas o interrupciones que llamadas), todas regresaran de la RAM indefinida. Esto causa que le programa regrese a la dirección del programa HF Hex, lo que provoca una lectura de ceros y de nueva cuenta invocar la instrucción INTR de software de trampa.

Operación: [SP] ← PCL
[SP -1] ← PCU
[SP -2] : ESTABLECER PARA LA SIGUIENTE PILA DE REFERENCIA
PC ← 0FF

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
INTR	Implícito	7	1	00

JID – Salto indirecto

Sintaxis: JID

Descripción: La instrucción JID usa el contenido del acumulador para apuntar a una tabla de vector indirecto de direcciones de programa. El contenido del acumulador es transferido al PCL(los 8 bits bajos de PC), después con el dato obtenido de la localidad de memoria especificado por PC es transferido a PCL. El programa entonces salta a la localidad de memoria indicada por PC. Se puede observar que PCU (los 7 bits superiores de PC) nunca cambia durante la instrucción JID, así que la instrucción JID saltará a la localidad en la página de direcciones de memoria de programa. Sin embargo, si la instrucción JID esta localizada en la última dirección de la página, el contador PC se incrementará sobre el límite de la página, y ambos accesos a la memoria de programa pueden ser llamados desde la siguiente página de 256 bytes.

Operación: PCL ← A
PCL ← Memoria de Programa (PCU,A)

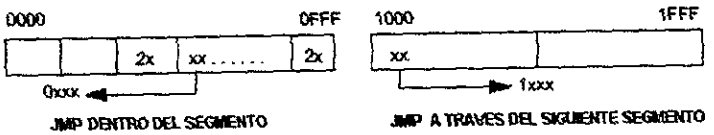
Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
JID	Indirecto	3	1	A5

JMP – Salto absoluto

Sintaxis: JMP ADDR

Descripción: Esta instrucción salta a la dirección de memoria programada. El valor encontrado en el nibble bajo (4 bits) del prime byte de la instrucción es transferido al nibble bajo de PCU (los 7 bits superiores de PC), y entonces el valor encontrado en el segundo byte de la instrucción es transferido al PCL(los 8 bits de PC). El programa entonces salta a la localidad de memoria de programa indicada por PC.

El rango de direcciones es de 15 bits. El contenido de la instrucción JMP es de solamente 12 bytes. La resolución de los 3 bits de la dirección es para los 4K del segmento de memoria conteniendo el byte bajo de la instrucción. El siguiente diagrama ilustra lo anterior:



Si la instrucción del byte 2 esta en el segmento 0 de 4K, el salto de dirección es en el segmento 0.

Si la instrucción del byte 2 esta en el segmento 1 de 4K, el salto de dirección esta en el segmento 1.

Operación: PC 11-8 ←HIADDR (NIBBLE ALTO DEL SEGUNDO BYTE DE INSTRUCCIÓN, NIBBLE BAJO DEL PRIMER BYTE DE INSTRUCCIÓN).

PC 7-0 ← LOADDR (SEGUNDO BYTE DE LA INSTRUCCIÓN)

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
JMP	Absoluto	3	2	2HIADDR/LOADDR

JMPL – Salto absoluto largo

Sintaxis: La instrucción JMPL permite bifurcar a cualquier parte de los 32-Kbyte de memoria de programa. El valor encontrado, en el segundo y tercer bytes de la instrucción, es transferido al PCU (los 7 bites superiores de PC) y PCL (los 8 bites bajos de PC)

respectivamente. El programa entonces salta a la localidad de memoria de programa accedida por PC.

Operación: PC 14-8 ← HIADDR (SEGUNDO BYTE DE LA INSTRUCCIÓN)

PC 7-0 ← LOADDR (TERCER BYTE DE LA INSTRUCCIÓN)

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
JMPL	Absoluto	4	3	AC/HIADDR/LOADDR

JP – Salto relativo

Sintaxis: JP DISP

Descripción: El valor de desplazamiento relativo encontrado en el código de la instrucción (los 8 bits completos) es sumado al contador de programa (PC). El incremento normal en el PC también es llevado a cabo. El valor de desplazamiento permite un camino de regreso de 0 a 31 lugares (el 0 representa un lazo cerrado infinito) y un camino hacia delante de 2 a 32 lugares. Una bifurcación hacia delante de 1 no esta permitida, puesto que el opcode cero tiene conflicto con la instrucción INTR (trampa de software).

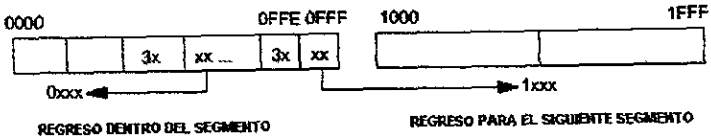
Operación: PC ← PC + DISP + 1 (DISP ≠ 0)

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
JP DISP	Relativo	3	1	0, 1, E, F + DISP #

JSR – Salto a subrutina

Sintaxis: Esta instrucción coloca la dirección de regreso dentro de la pila de software en el dato de memoria y salta a la dirección de la subrutina. El contenido e PCL (los 8 bites bajos de PC) son transferidos a la localidad de memoria de dato indicada por SP (apuntador de la pila) SP es entonces disminuido, entonces el contenido de PCU (los 7 bites superiores de PC) comienzan a transferirse a la nueva localidad de memoria de dato señalada por SP. La dirección de retorno es entonces guardada en la pila de software en el dato de memoria RAM. Luego el SP es otra vez disminuido para establecer la referencia de la pila de software para la siguiente subrutina.

El rango de dirección es de 15 bits. El contenido de la dirección de la instrucción JSR es de solamente 12 bites. La resolución de los 3 bits altos de la dirección es para los 4K del segmento de memoria que contienen el byte alto de la dirección de retorno. Un JSR contenida en los últimos 2 bytes de un segmento de 4K saltará al siguiente segmento. El siguiente diagrama ilustra lo anterior:



Si la dirección de retorno esta en el segmento 0 de los 4K, la dirección de salto esta en el segmento 0.
Si la dirección de retorno esta en el segmento 1 de los 4K, la dirección de salto esta en segmento 1.

Operación: [SP] ← PCL

[SP - 1] ← PCU

[SP - 2]; ESTABLECIMIENTO PARA LA SIGUIENTE REFERENCIA DE LA PILA.

PC 11-8 ← HIADDR (NIBBLE ALTO DE LA DIRECCIÓN DE RETORNO, NIBBLE BAJO DEL PRIMER BYTE DE LA INSTRUCCIÓN)

PC 7-0 ← LOADDR (SEGUNDO BYTE DE LA INSTRUCCIÓN)

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
JSR DISP	Absoluto	5	2	3HIADDR/LOADDR

JSRL – Salto largo a subrutina

Sintaxis: JSRL ADDR

Descripción: La instrucción JSRL permite que la subrutina este localiza da en cualquier parte de los 32 Kbytes del espacio de memoria. La instrucción coloca la dirección de retorno

dentro de la pila de software en la memoria de dato y entonces salta a la dirección de la subrutina.

El contenido de PCL (los 8 bites bajos de PC) es transferido a la localidad del dato de memoria indicada por SP (puntero de pila). SP es entonces disminuido, entonces el contenido de PCU (los 7 bits superiores de PC) comienza a transferirse hacia la nueva localidad de memoria de dato señalada por SP. La dirección de retorno es ahora guardado en la pila de software en la memoria de dato RAM. Entonces SP es otra vez decrementado para establecer la referencia de la pila de software para la siguiente subrutina.

A continuación, los valores encontrados en el segundo y tercer byte de la instrucción son transferidos a PCU y a PCL respectivamente. El programa entonces salta a la localidad de memoria de programa accedida por PC.

Operación: [SP] ← PCL

[SP -1] ← PCU

[SP - 2]: ESTABLECER LA REFERENCIA PARA LA SIGUIENTE PILA

PC 14-8 ← HIADDR (SEGUNDO BYTE DE LA INSTRUCCIÓN)

PC 7-0 ← LOADDR (TERCER BYTE DE LA INSTRUCCIÓN)

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
JSRL ADDR	Absoluto	5	3	AD/HIADDR/LOADDR

LAID – Carga del acumulador indirecta

Modo de direccionamiento: INDIRECTO

Descripción: La instrucción LAID usa el contenido del acumulador para apuntar a una tabla de datos fija almacenada en la memoria de programa. La tabla de datos normalmente representa una matriz de traducción, tal como la entrada de un teclado o la salida a una pantalla.

El contenido del acumulador es intercambiado con el contenido del PCL(los 8 bites bajos de PC). El dato accedido de la localidad de memoria de programa especificada por PC es entonces transferida al acumulador. Simultáneamente, el contenido original de PCL es regresado de vuelta del acumulador. Puede observarse que PCU (los 7 bites superiores de PC) no es modificado durante la instrucción LAID, a sí que la carga indirecta del acumulador junto con la tabla de datos fija asociada deben estar localizados en la página actual de memoria de 256 bytes. Sin embargo, si la instrucción LAID esta localizada en la última dirección de la página, el contador PC tendrá listo el incremento sobre el límite de la página resultando en el comienzo del llamado del operando de la siguiente página. En consecuencia, en este instante, la tabla de datos fija deberá residir en la siguiente página de 256 bytes en la memoria de programa.

Operación: A ← Memoria de Programa (PCU, A)

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
LAID	Indirecto	3	1	A4

LD – Carga del acumulador

Sintaxis: a) LD A,[B]

b) LD A,[B+]

c) LD A,[B-]

d) LD A,#

e) LD A,MD

f) LD A,[X]

g) LD A,[X+]

h) LD A,[X-]

Descripción: a) El contenido de la localidad de la memoria de dato indicada por el puntero B es almacenado en el acumulador.

b) El contenido de la localidad de la memoria de dato indicada por el puntero es almacenado en el acumulador, y entonces el puntero B es post-incrementado.

c) El contenido de la localidad de memoria de dato indicada por el puntero B es almacenado en el acumulador, y entonces el puntero B es post-decrementado.

d) El valor inmediato encontrado en el segundo byte de la instrucción es almacenado en el acumulador.

e) El contenido de la localidad de memoria de dato indicada por la dirección en el segundo byte de la instrucción es almacenado en el acumulador.

f) El contenido de la localidad de memoria de dato indicada por el puntero X es almacenado en el acumulador.

g) El contenido de la localidad de memoria de dato indicada por el puntero X es almacenado en el acumulador.

h) El contenido de la localidad de memoria de dato indicada por el puntero X es almacenado en el acumulador, y entonces el puntero X es post-decrementado

Operación: A ← VALOR

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
LD A,[B]	Registro Indirecto (Puntero B)	1	1	AE
LD A,[B+]	Registro Indirecto con post incremento del puntero B	2	1	AA
LD A,[B-]	Registro Indirecto con post decremento del puntero B	2	1	AB
LD A,#	Inmediato	2	2	98/Imm,#
LD A,MD	Memoria Directa	3	2	9D/MA
LD A,[X]	Registro Indirecto (Puntero X)	3	1	BE
LD A,[X+]	Registro Indirecto con post incremento del puntero X	3	1	BA
LD A,[X-]	Registro Indirecto con post decremento del puntero X	3	1	BB

LD – Carga del puntero B

Sintaxis: a) LD B,# (# < 16)

b) LD B,# (# > 15)

Descripción: a) El complemento a uno del valor encontrado en el nibble bajo (4 bits) de la instrucción es transferido a la posición del nibble bajo del registro B, con la posición del nibble superior comenzado a ser puesto a cero.

b) El valor inmediato en el segundo byte de la instrucción es transferido al registro B.

Operación: a) B3-B0 ← # y B7-B4 ← 0

b) B←#

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
LD B,#	Inmediato corto	1	1	5(15-#)
LD B,#	Inmediato	2	2	9F/Imm,#

LD – Carga en memoria

Sintaxis:
a) LD [B],#
b) LD [B+],#
c) LD [B-],#
d) LD MD,#

Descripción:
a) El valor inmediato encontrado en el segundo byte de la instrucción es almacenado en la localidad de memoria de dato indicada por el puntero B.

b) El valor inmediato encontrado en el segundo byte de la instrucción es almacenado en la localidad de memoria de dato indicada por el puntero B, y entonces el puntero B es post incrementado.

c) El valor inmediato encontrado en el segundo byte de la instrucción es almacenado en la localidad de memoria de dato indicada por el puntero B, y entonces el punte B es post decrementado.

d) El valor inmediato encontrado en el tercer byte de la instrucción es almacenado en la localidad de memoria de dato indicada por la dirección en el segundo byte de la instrucción.

Operación: a) [B] ← #

b) [B] ←#; B ← B + 1

c) [B] ←#; B ← B - 1

d) MD[B] ← #

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
LD [B],#	Registro Inditecto/Inmediato	2	2	9E/Imm,#
LD [B+],#	Registro Indirecto con post incremento / Inmediato	2	2	9A/Imm,#
LD [B-],#	Registro Indirecto con post decremento / Inmediato	2	2	9B/Imm,#
LD MD,#	Memoria directa / Inmediata	3	3	BC/MA/Imm,#

LD – Cargar registro

Sintaxis: LD REG,#

Descripción: El valor inmediato encontrado en el segundo byte de la instrucción es almacenado en el registro de memoria de dato indicada por el nibble bajo del primer byte de la instrucción.

Operación: REG ← #

0

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
LD REG],#	Implicito / Inmediato	3	2	D(REG,#) / Imm,#

NOP – Ninguna operación

Sintaxis: NOP

Descripción: Ninguna operación es desarrollada por esta instrucción, por lo que el resultado es un retardo de un ciclo de instrucción.

Operación: SIN OPERACIÓN

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
NOP	Implícito	1	1	B8

OR – O lógica

Sintaxis: a) OR A,[B]
b) OR A,#
c) OR A,MD

Descripción: Una operación OR es desarrollada en los bits correspondientes del acumulador con:

a) el contenido de la localidad de memoria de dato indicada por el puntero B.

b) el valor inmediato en el segundo byte de la instrucción.

c) el contenido de la localidad de memoria de datos indicada por la dirección en el segundo byte de la instrucción.
El resultado es colocado de nuevo en el acumulador.

Operación: A ← A OR VALOR

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
OR A,[B]	Registro Indirecto (puntero B)	1	1	87
OR A,#	Inmediato	2	2	97 / Imm,#
OR A,MD	Memoria Directa	4	3	BD / MA / 87

POP – Pop de la pila

Sintaxis: POP A

Descripción: El puntero de la pila (SP) es incrementado, y entonces el contenido de la localidad de memoria de dato indicada por el SP es transferido hacia el acumulador.

Operación: SP ← SP + 1

A [SP]

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
POP	Implícito	3	1	8C

PUSH – PUSH de la pila

Sintaxis: PUSH A

Descripción: El contenido del acumulador es transferido a la localidad de memoria de dato indicada por el puntero de pila (SP), y entonces el SP es decrementado.

Operación: [SP] ← A

SP ← SP -1

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
PUSH	Implícito	3	1	67

RBIT – Puesta a cero de bit de memoria

Sintaxis: a) RBIT #,[B]

b) RBIT #,MD

Descripción: El bit seleccionado (# = 0 a 7, siendo 7 el bit de orden mayor) de la localidad de memoria indicada por el (la):

a) puntero B es puesto a 0.

b) dirección en el segundo byte de la instrucción es puesta a 0.

Operación: [Dirección:#] ← 0

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
RBIT #,[B]	Registro indirecto (puntero B)	1	1	6(8 + #)
RBIT #,[B]	Memoria directa	4	3	BD/MA/6(8+#)

RC – Reiniciar el acarreo

Sintaxis: RC

Descripción: Las banderas de acarreo y medio acarreo son puestas a cero.

Operación: $C \leftarrow 0$

$HC \leftarrow 0$

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
RC	Implícito	1	1	A0

RET – Retorno de una subrutina

Sintaxis: RET

Descripción: El puntero de la pila (SP) es primero incrementado El contenido de la localidad de memoria de dato indicada por SP es transferido a PCU (los 7 bites superiores de PC), después SP se incrementa otra vez. A continuación, el contenido de la localidad de memoria de dato indicada por SP es transferido a PCL (los 8 bites bajos de PC). La dirección de retorno es ahora recuperada de la pila de software en la memoria de dato RAM. El programa entonces salta a la localidad de memoria accedida por PC.

Operación: $PCU \leftarrow [SP + 1]$

$PCL \leftarrow [SP + 2]$

[SP + 2] : ESTABLECER PARA LA SIGUIENTE REFERENCIA DE PILA

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
RET	Implícito	5	1	8E

RETI – Retorno de una interrupción

Sintaxis: RETI

Descripción: El puntero de pila (SP) es primero incrementado. El contenido de la localidad de memoria de dato indicada por SP es transferido a PCU (los 7 bits superiores de PC), y SP es incrementado otra vez. Enseguida, el contenido de la localidad de memoria de dato indicada por SP es transferida a PCL (los 8 bits bajos de PC). La dirección de retorno es ahora recuperada de la pila de software en la memoria de dato RAM. El programa ahora salta a la localidad de memoria de programa accedida por PC. La bandera de habilitación de interrupción global (GIE) es puesta a 1.

Operación: PCU ← [SP + 1]

PCL ← [SP + 2]

[SP +2] : ESTABLECER PARA LA SIGUIENTE REFERENCIA DE PILA

GIE ← 1

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex

RETI	Implícito	5	1	8F

RETSK – Regresar y saltar

Sintaxis: RETSK

Descripción: El puntero de pila (SP) es primero incrementado. El contenido de la localidad de memoria de dato indicada por PC es transferido a PCU (los 7 bits superiores de PC), y SP es incrementado de nuevo. A continuación, el contenido de la localidad de memoria de dato indicada por SP es transferido a PCL (los 8 bits bajos de PC). La dirección de retorno es ahora recuperada de la pila de software en la memoria de dato RAM. El programa ahora regresa y salta la instrucción en la localidad de memoria de programa accedida por PC.

Operación: PCU ← [SP + 1]

PCL ← [SP + 2]

[SP + 2] : ESTABLECER PARA LA SIGUIENTE REFERENCIA DE PILA

SALTAR LA SIGUIENTE INSTRUCCIÓN

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
RETSK	Implícito	5	1	8D

RLC – Rotación del acumulador a la izquierda por medio del acarreo

Modo de direccionamiento: RLC A

Descripción: El contenido del acumulador y la bandera de acarreo son rotados a la izquierda un bit, la bandera de acarreo sirve como un noveno bit ligando el final de los 8 bits del acumulador. El acarreo previo es transferido al bit más bajo del acumulador. El bit más alto del acumulador (A7) es transferido a la bandera de acarreo. El A3 (bit alto del nibble inferior) del acumulador es transferido a la bandera de medio acarreo (HC) a si como el bit A4.

Operación: $C \leftarrow A7 \leftarrow A6 \leftarrow A5 \leftarrow A4 \leftarrow A3 \leftarrow A2 \leftarrow A1 \leftarrow A0 \leftarrow C$

$HC \leftarrow A3$

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
RLC A	Implícito	1	1	A8

RPND – Restablecimiento pendiente

Sintaxis: RPND

Descripción: La instrucción RPND reestablece la bandera pendiente de interrupción no enmascarada (NMIPND) siempre que la interrupción NMI este reconocida y la bandera pendiente de trampa de software (STPND) no se encuentre establecida. Además, RPND incondicionalmente restablece la bandera pendiente de trampa de software.

Operación: SI LA INTERRUPCIÓN NMI ES RECONOCIDA Y STPND = 0

ENTONCES NMPND ← 0 Y STPND ← 0

SI NO STPND ← 0

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
RPND	Implícito	1	1	B5

RRC – Rotar el acumulador a la derecha por medio de acarreo

Modo de direccionamiento: RRC A

Descripción: El contenido del acumulador y la bandera de acarreo son rotados a la derecha un bit, la bandera de acarreo sirve cono un noveno bit liga el final de los 8 bits del acumulador El acarreo previo es transferido al bit más alto del acumulador. El bit bajo del acumulador (A0) es transferido a la bandera de acarreo y la bandera de acarreo medio

Operación: C ← A7 ← A6 ← A5 ← A4 ← A3 ← A2 ← A1 ← A0 ← C

A0 ← HC

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
RRC A	Implícito	1	1	B0

SBIT – Establecer bit de memoria

Sintaxis: a) SBIT #,[B]
b) SBIT #,MD

Descripción: El bit seleccionado (# = 0 a 7, el bit 7 es el de mayor orden) de la localidad del dato de memoria indicada por el (la):

a) puntero B esta establecido como 1.
b) dirección en el segundo byte de la instrucción esta puesto como 1.

Operación: [Dirección:#] ← 1

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
SBIT #,[B]	Registro indirecto (puntero B)	1	1	7(8 + #)
SBIT #,[B]	Memoria directa	4	3	BD/MA/7(8 + #)

SC – Puesta a 1 del acarreo

Sintaxis: SC

Descripción. Tanto como el acarreo y el medio acarreo son puestos a 1.

Operación: C ← 1

HC ← 1

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
SC	Implícito	1	1	A1

SUBC – Substracción con acarreo

Sintaxis: a) SUBC A,[B]
b) SUBC A,#
c) SUBC A,MD

Descripción: a) El contenido de la localidad de memoria de dato indicada por el puntero B es substraído del contenido del acumulador, y el resultado es simultáneamente decrementado si la bandera de acarreo es encontrada previamente reiniciada

b) El valor inmediato encontrado en el segundo byte de la instrucción es substraído del contenido del acumulador, y el resultado es simultáneamente decrementado si la bandera de acarreo es encontrada previamente reiniciada.

c) El contenido de la localidad de memoria de dato indicada por la dirección en el segundo byte de la instrucción es substraída del contenido del acumulador, y el resultado es simultáneamente decrementado si la bandera de acarreo es encontrada previamente reiniciada.

El resultado colocado de regreso en el acumulador, y la bandera de acarreo es establecida o reiniciada, dependiendo de la presencia o ausencia de un préstamo del resultado. Similarmente, la bandera de medio acarreo es establecida o reiniciada, dependiendo de la presencia o ausencia de un préstamo del nibble inferior.

Esta instrucción es implementada por la suma del complemento a uno del substraendo al acumulador y entonces incrementar el resultado. Por consiguiente, el préstamo es el equivalente de la ausencia de acarreo y viceversa. Similarmente el medio acarreo es el equivalente de la ausencia del medio préstamo y viceversa. Un préstamo previo (ausencia de acarreo previo) inhibirá el incremento del resultado.

Operación: $A \leftarrow A + VALOR + C$

$C \leftarrow$ AUSENCIA DE BYTE DE PRÉSTAMO

$HC \leftarrow$ AUSENCIA DE PRESTAMO MEDIO DE NIBBLE INFERIOR

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
SUBC A,[B]	Registro indirecto (puntero B)	1	1	81
SUBC A,#	Inmediato	2	2	91/Imm,#
SUBC A,MD	Memoria directa	4	3	BD/MA/81

SWAP – Intercambio de nibbles del acumulador

Sintaxis. SWAP A

Descripción: Los nibbles superior e inferior del acumulador son intercambiados.

Operación: $A(7\text{-}4) \leftrightarrow A(3\text{-}0)$

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
SWAP A	Implícito	1	1	65

VIS – Selección de vector de interrupción

Sintaxis: VIS

Descripción: El propósito de la instrucción VIS es para dirigir a la rutina del servicio de interrupción para la interrupción con la prioridad más alta y orden arbitrario que esta actualmente establecida y demandada. La instrucción VIS agiliza este procedimiento de dirigir la rutina del servicio de interrupción.

Todas las interrupciones se dirigen a la localidad de memoria 00FF Hex una vez que una interrupción es reconocida. Así, cualquier cambio de entorno deseado (tal como el almacenamiento fuera del contenido del acumulador B o del puntero X) es generalmente programado para comenzar en al dirección 00FF Hex, seguido por la instrucción VIS. La instrucción VIS puede ser programada en la localidad de memoria 00FF Hex si no se desea el cambio de entorno.

La instrucción VIS primero salta a un vector de doble byte en una tabla de memoria de programa de vector de interrupción que esta localizada en la parte superior de un bloque de memoria de programa de direcciones xyE0 a xyFF Hex. Nótese que xy es el número de bloque (normalmente 01) donde la instrucción VIS esta localizada (cada bloque de memoria de programa contiene 256 bytes). Este vector de doble byte es transferido a PC (primer byte de orden alto), y el programa salta a la rutina del servicio de interrupción asociada indicada por el vector. Esta rutina del servicio de interrupción puede estar en cualquier espacio de memoria de programa en los 32 Kbytes.

Deberá la instrucción VIS es programada en la parte superior de un bloque de memoria (tal y como la dirección 00FF Hex), la tabla de vector asociada de 32 byte

esta residente en la parte alta del siguiente bloque alto (localidades 01E0 a 01FF Hex) con la instrucción VIS en 00FF Hex).

Operación: PCL ← VA (Vector arbitrario de interrupción generado por hardware)

PCU ← Memoria de programa (PCU,VA)

PCL ← Memoria de programa (PCU,VA+1)

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
VIS	Implícito	5	1	B4

X – Intercambio de memoria con el acumulador

Sintaxis:
a) X A,[B]
b) X A,[B+]
c) X A,[B-]
d) X A,MD
e) X A,[X]
f) X A,[X+]
g) X A,[X-]

Descripción: a) El contenido de la localidad de memoria de dato indicada por el puntero B es intercambiado con el contenido del acumulador.

b) El contenido de la localidad de memoria de dato indicada por el puntero B es intercambiado con el contenido del acumulador, y entonces el puntero B es post incrementado

c) El contenido de la localidad de memoria de dato indicada por el puntero B es intercambiado con el contenido del acumulador, y entonces el puntero B es post decrementado.

d) El contenido de la localidad de memoria de dato indicada por la dirección en el segundo byte de la instrucción es intercambiado con el contenido del acumulador.

e) El contenido de la localidad de memoria de dato indicada por el puntero X es intercambiado con el contenido del acumulador.

f) El contenido de la localidad de memoria de dato indicada por el puntero X es intercambiado con el contenido del acumulador, y entonces el puntero X es post decrementado.

c) El contenido de la localidad de memoria de dato indicada por el puntero X es intercambiado con el contenido del acumulador, y entonces el puntero X es post decrementado.

Operación. a) $A \leftrightarrow [B]$

b) $A \leftrightarrow B$; $B \leftarrow B + 1$

c) $A \leftrightarrow B$; $B \leftarrow B - 1$

d) $A \leftrightarrow MD$

e) $A \leftrightarrow X$

f) $A \leftrightarrow X$; $X \leftarrow X + 1$

g) A ↔ X; X ← X - 1

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
X A, [B]	Registro Indirecto (puntero B)	1	1	A6
X A,[B+]	Registro Indirecto con post incremento puntero B	2	1	A2
X A,[B-]	Registro Indirecto con post decremento puntero B	2	1	A3
X A,MD	Memoria directa	3	2	9C/MA
X A,[X]	Registro indirecto (puntero X)	3	1	B6
X A,[X+]	Registro Indirecto con post incremento puntero X	3	1	B2
X A,[X-]	Registro Indirecto con post decremento puntero X	3	1	B3

XOR – O exclusiva

Sintaxis: a) XOR A,[B]
b) XOR A,#
c) XOR A,MD

Descripción: Una operación XOR (OR exclusiva) es ejecutada en los bits correspondientes del acumulador con:

a) el contenido de la localidad de memoria de dato indicada por el puntero B

b) el valor inmediato encontrado en el segundo byte de la instrucción.

c) el contenido de la localidad de memoria indicada por la dirección en el segundo byte de la instrucción.

El resultado es colocado en el acumulador.

Operación: A ← A XOR VALOR

Instrucción	Modo de direccionamiento	Ciclo de Instr.	Bytes	Código Hex
XOR A, [B]	Registro Indirecto (puntero B)	1	1	86
XOR A,#	Inmediato	2	2	96/Imm,#
XOR A,MD	Memoria directa	4	3	BD/MA/86

Apéndice B Listados de código ensamblador para el COP8

B.1 Listado del archivo MOTOR.ASM

```
.INCLD cop8sac.inc necesidad
;Declaración de constantes usadas en la difusificación,
;evaluación de reglas y rutinas de  de-difusificación.

NUMI = 1              ; número de variables de entrada
NMFI1 = 3             ; número de MFs pata entrada 1
NMFI2 = 0             ; número de MFs pata entrada 2
NMFI3 = 0             ; número de MFs pata entrada 3
NMFI4 = 0             ; número de MFs pata entrada 4
NUMRULES = 3          ; número de reglas (decimal)
RULECNT1 = 003              ; número de reglas(byte bajo)
RULECNT2 = 000              ; número de reglas(byte alto)
RULELENGTH = 3              ; número de bytes de almacenamiento para una regla
RULEPERBLOCK = 71   ; número de reglas que caben en un bloque de 256 bytes
SB = 7              ; bit índice

;***************** ASIGNACIÓN DEL ALMACENAMIENTO DE VARIABLES  ********************
;Estos módulos de datos declaran variables usadas en la difusificación
;y en las rutinas de evaluación. No se deben alterar estas declaraciones.

.PUBLIC I1,I2
.PUBLIC OUT1,OUT2,OUT3,OUT4,OUT5
.PUBLIC MUL1,MULT1,MULT2,PROD1,PROD2,PROD3,PROD4,PROD5
.PUBLIC DIVR1,DIV1,DIV2,QUO1,QUO2,REM1,REM2

;*** MÓDULO FUZDAT ***
.SECT FUZDAT,RAM
PARAM: .DSB 10          ;MF parámetro durante  la difusificación
       IN=PARAM:BYTE    ;Argumentos de reglas durante evaluación de reglas
       PA=IN+1:BYTE
       PB=IN+2:BYTE
       PC=IN+3:BYTE
       PD=IN+4:BYTE
       PE=IN+5:BYTE
       PF=IN+6:BYTE
       H1=IN+7:BYTE
```

```
        H2=IN+8:BYTE
        H3=IN+9:BYTE
        PWU=IN+5
        PWL=IN+6
IMFD:   .DSB    3               ;Tamaño total del número de funciones de membresia
        I1MF1=IMFD:BYTE     ;funciones para todas las entradas (max 28 decimal)
INPUT:  .DSB 2    ;Entradas del sistema acondicionadas(parámetro transitorio)
        I1=INPUT:BYTE
        I2=I1+1:BYTE
        RCNT1=I1:BYTE               ;Contador de reglas
        RCNT2=I2:BYTE
OUT:    .DSB 5                      ;Salida difusa (parámetro transitorio)
        OUT1=OUT                    ;byte menos significante
        OUT2=OUT+1
        OUT3=OUT+2
        OUT4=OUT+3
        OUT5=OUT+4                  ;byte mas significante
TEMPS:  .DSB 2                      ;Almacenamiento temporal de datos
        MFN=TEMPS:BYTE              ;Número de funciones de membresia
                                ;       0 <= MFN <= 127
        RPAGE=MFN                   ;Página de reglas actual
        TEMP=MFN+1:BYTE             ;En borrador

;*** FAST MODULE ***
.SECT FAST,BASE
MATH:   .DSB 6          ;Registros de las rutinas de Multiplicación y división
        MUL1=MATH:BYTE              ;Multiplicador
        MULT1=MUL1+1                ;Multiplicando (byte bajo)
        MULT2=MUL1+2                ;Multiplicando (2nd byte)
        PROD1=MULT1                 ;Producto (byte bajo)
        PROD2=MULT2                 ;Producto (2nd byte)
        PROD3=MUL1+3                ;Producto (3rd byte)
        PROD4=MUL1+4                ;Producto (4th byte)
        PROD5=MUL1+5                ;Producto (byte alto)
        DIVR1=MUL1                  ;Divisor
        DIV1=MULT1                  ;Dividendo (byte bajo)
        DIV2=MULT2                  ;Dividendo (byte alto)
        QUO1=MULT1                  ;Cociente (byte bajo)
        QUO2=MULT2                  ;Cociente (byte alto)
        REM1=PROD3                  ;Resto (byte bajo)
        REM2=PROD4                  ;Resto (byte alto)
```

```
;*** MÓDULO TOP ***
.SECT TOP,REG                    ;Contador de almacenamiento
COUNTER: .DSB 3
       CNT=COUNTER               ;Contador
       MFCNT=CNT+1               ;Contador de funciones de memebresia
       RINDEX=MFCNT              ;Indice de regla actual
       SGNFLG=CNT+2              ;Bandera del Factor de paso
.ENDSECT
;********************** MÓDULO FUZZ *********************************
;Este módulo contiene la rutina principal del código difuso. También realiza la
difusificación de las entradas I1 a I2
;
;ENTRADAS:    I1,I2,I3,I4
;ALTERDOS : A,B,X,HC,C,CNT,MFCNT,MFN,PA thru H1,I1,I2,IN,TEMP,MUL/DIV
;           REGs, I1MF1-I4MF7 (DoM for all inputs),SGNFLG
;SALIDAS:   OUT1,OUT2,OUT3,OUT4,OUT5
;
.PUBLIC FUZZ
.SECT FUZZ,ROM
FUZZ:
        LD      A,I1             ;Inicializa parámetros para difusificar I1
        X       A,IN
        LD      MFCNT,#NMFI1        ;Cargar el contador MF con el número de MFs
                                    ; para I1
        LD      MFN,#00             ;Inicializar el número global MF
        JSR     GOFUZ               ;Calculo de DoM para cada MF de I1
        JSR     RULEVAL             ;Evaluación de reglas y calculo de la salida
        RET                         ;Regreso al código de la aplicación

; *** GOFUZ ***
; Llamado de la rutina de búsqueda de parámetros y de la rutina de       ; calculo para
todas las funciones de membresia de la entrada IN
GOFUZ:
        JSR     MFPLUP          ;Búsqueda de parámetros MF
        JSR     MEDCALC         ;Calculo del  grado de membresia
        X       A,B             ;Almacenar el grado de membresia en B
        LD      A,MFN           ;Calcular la localidad de almacenamiento
        ADD     A,#I1MF1
        X       A,B             ;Cargar A con DoM, Cargar B con la dirección
        X       A,[B]           ;Almacenar DoM
        LD      A,MFN           ;Incrementar el número de la MF
        INC     A
```

```
        X       A,MFN           ;Guardar el número de la MF
        DRSZ    MFCNT           ;Disminuir el contador de la rutina de MF
        JP      GOFUZ           ;Si no es cero entonces procesar la siguiente MF
        RET                     ;en caso contrario regresar


;*** MFDCALC ***
;Calculo del grado de membresía para una entrada en una FM particular
;IN:    IN,PA,PB,PC,PD,PE,PF,H1,H2,H3
;ALTER: A,B,X,CNT,MUL/DIV REGs
;OUT:   A=Degree-of-membership
MFDCALC:
        LD      B,#IN                   ;Cargar B con la localidad de INPUT
        LD      A,[B]                   ;Cargar A con INPUT
        IFGT    A,PF                    ;Si INPUT > PF
        JP      REG6                    ;entonces INPUT esta en la región 6
        LD      A,PA                    ;}Cargar A con PA
        IFEQ    A,[B]                   ;Si PA == INPUT
        JP      TEST1                   ;entonces prueba todas las regiones
        IFGT    A,[B]                   ;Si (PA > INPUT)
        JP      REG6                    ;entonces INPUT esta en la región 6
TEST1:  LD      A,[B]                   ;Cargar A con INPUT
        LD      B,#PC                   ;Cargar B con la localidad de PC
        IFEQ    A,[B]                   ;Si INPUT == PC
        JP      REG1                    ;entonces INPUT esta en la región 1
        IFGT    A,[B]                   ;Si INPUT > PC
        JP      CTEST1                  ;entonces continuar probando la región 1
        JP      TEST2                   ;si no ir a al prueba de los límites de 2
REG6:   LD      A,#00                   ;Región 6 regresar a cero
        RET                             ;regresar
REG1:   LD      A,H2                    ;Región 1 regresar a altura H2
        RET                             ;regresar
CTEST1: LD      B,#IN                   ;Cargar B con la localidad de INPUT
        LD      A,PD                    ;Cargar A con PD
        IFEQ    A,[B]                   ;Si PD == INPUT
        JP      REG1                    ;entonces INPUT esta en la Región 1
        IFGT    A,[B]                   ;Si (PD > INPUT)
        JP      REG1                    ;entonces INPUT esta en la Región 1
TEST2:  LD      A,IN                    ;Cargar A con INPUT
        LD      B,#PB                   ;Cargar B con PB
        IFEQ    A,[B]                   ;Si (INPUT == PB)
        JP      REG2                    ;entonces INPUT esta en la Región 2
```

```
        IFGT    A,[B]           ;Si (INPUT > PB)
        JP      CTEST2          ;entonces continuar probando la Región 2
        JP      TEST3           ;si no probar los límites de la Región 3
REG2:   LD      A,IN            ;Calcular MFD para la Región 2
        SC
        SUBC    A,PB            ;A = (INPUT - PB)
        LD      B,#MUL1
        X       A,[B+]          ;Store result of subr in MUL1
        LD      A,H2
        SC
        SUBC    A,H1            ;A = (H2 - H1)
        X       A,[B]           ;Guardar el resultado de en MULT1
        JSR     MUL08           ;PROD2:1 = (INPUT-PB)*(H2-H1)
        LD      B,#PC           ;Cargar B con la localidad PC
        JSR     TDIV            ;Dividir PROD2:1 by (PC - PB)
        ADD     A,H1
        RET
CTEST2: LD      A,PC            ;Cargar A con PC
        IFGT    A,IN            ;Si PC > INPUT
        JP      REG2            ;entonces la entrada esta en la Región 2
TEST3:  LD      B,#IN           ;Cargar B con la localidad INPUT
        LD      A,PE            ;Cargar A con PE
        IFEQ    A,[B]           ;Si (PE == INPUT)
        JP      REG3            ;entonces la entrada esta en la Región 3
        IFGT    A,[B]           ;Si (PE > INPUT)
        JP      CTEST3          ;entonces continuar probando la  Región 3
        JP      TEST4           ;si no probar los límites de la  Región 4
REG3:   LD      A,PE            ;Calculo de  MFD para la Región 3
        SC
        SUBC    A,IN            ;A = (PE - INPUT)
        LD      B,#MUL1
        X       A,[B+]          ;Almacenar el resultado en MUL1
        LD      A,H2
        SC
        SUBC    A,H3            ;A = (H2 - H3)
        X       A,[B]           ;Almacenar el resultado en MULT1
        JSR     MUL08           ;PROD2:1 = (PE-INPUT)*(H2-H3)
        LD      B,#PE           ;Cargar B con la localidad de PE
        JSR     TDIV            ;Dividir PROD2:1 por (PE - PD)
        ADD     A,H3
        RET
CTEST3: LD      A,IN            ;Cargar el acumulador con INPUT
```

```
        IFGT    A,PD            ;Si INPUT > PD
        JP      REG3            ;entonces la entrada esta en la Región 3
TEST4:  LD      B,#PA           ;Cargar B con la localidad de PA
        LD      A,IN            ;Cargar el acumulador con INPUT
        IFEQ    A,[B]           ;Si (INPUT == PA)
        JP      REG4            ;La entrada esta en la región 4
        IFGT    A,[B]           ;Si (INPUT > PA)
        JP      CTEST4          ;entonces continuar probando la región 4
        JP      REG5            ;si no la entrada esta en la Región 5
REG4:   LD      A,IN            ;Calcular MFD para la Región 4
        SC
        SUBC    A,PA            ;A = (INPUT - PA)
        LD      B,#MUL1
        X       A,[B+]          ;Almacenar el resultado en MUL1
        LD      A,H1
        X       A,[B]           ;almacenar el resultado en MULT1
        JSR     MUL08           ;PROD2:1 = (INPUT-PA)*(H1)
        LD      B,#PB           ;Cargar B con la localidad de PE
        JP      TDIV            ;Dividir PROD2:1 por(PB - PA)
CTEST4: LD      A,PB            ;Cargar A con PB
        IFGT    A,IN            ;Si PB > INPUT
        JP      REG4            ;si no la entrada esta en la Región 4
REG5:   LD      A,PF            ;Calcular MFD para la Región 5
        SC
        SUBC    A,IN            ;A = (PF - INPUT)
        LD      B,#MUL1
        X       A,[B+]          ;Almacenar el resultado en MUL1
        LD      A,H3
        X       A,[B]           ;Almacenar el resultado en MULT1
        JSR     MUL08           ;PROD2:1 = (PF-INPUT)*(H3)
        LD      B,#PF           ;Cargar B con la localidad de PF
                                ;Dividir PROD2:1 por (PF - PE)
TDIV:   LD      A,[B-]          ;Región 2: Divisor = PC-PB
        SC                      ;Región 3: Divisor = PE-PD
        SUBC    A,[B]           ;Región 4: Divisor = PB-PA
        LD      B,#DIVR1        ;Región 5: Divisor = PF-PE
        X       A,[B]
        JSR     DIV             ;Quol = MULT2:1 dividir por Divisor
        LD      A,DIVR1
        RC
        RRC     A               ;Dividir divisor por 2
        IFGT    A,REM1          .Si divisor/2 > residuo
```

```
        JP      NINCR                   ;entonces ignorar el residuo
        LD      A,QUO1                  ;si no redondear hacia arriba
        INC     A
        RET                             ;Regresar A = DM = Quo1 + 1
NINCR:  LD      A,QUO1
        RET                             ;Regresar A = DM = Quo1
.ENDSECT
;****************************** MÓDULO MFCODE ******************************
;Este módulo contiene el código para acceder a los valores de la tabla de la ;función
de membresía en ROM. Un llamado a esta rutina busca los parámetros para ;una función de
membresía.
;ENTRADAS:      IN,MFN
;ALTERADAS:     A,B,X,CNT,TEMP
;SALIDAS:       IN,PA,PB,PC,PD,PE,PF,H1,H2,H3
.SECT MFCODE,ROM
MFPLUP:
        LD      X,#PA                   ;Fijar X = RAM guardar la localidad de PA
        LD      A,MFN                   ;Cargar A con la MFN
        LD      CNT,#08                 ;Fijar el Contador a 8
MUL9:   ADD     A,MFN                   ;Multiplicar la MFN por 9
        DRSZ    CNT                     ;conseguir el índice dantro de la tabla
        JP      MUL9
        ADD     A,#L(IMFTBL)            ;Sumar el inicio de la dirección de la
                                        ;tabla a el índice
        LD      CNT,#09                 ;Fijar el Contador a 9(buscar parámetros)
        LD      B,#TEMP                 ;Fijar B a la localidad de almacenamiento
                                        ;para la dirección de la tabla
UP1:    X       A,[B]                   ;Guardar la dirección de la tabla
        LD      A,[B]                   ;Recuperar la dirección de la tabla en A
        JSR     LUP                     ;Buscar dato
        X       A,[X+]                  ;Guardar dato, incrementar el puntero a
                                        ;la siguiente localidad de almacenamiento
        LD      A,[B]                   ;Recuperar la dirección de la tabla en A
        INC     A                       ;Incrementar la dirección de la tabla al
                                        ;siguiente dato
        DRSZ    CNT                     ;Disminuir el registro del contador
        JP      UP1                     ;Si no es cero entonces buscar el dato
                                        ;siguiente
        RET                             ;si no regresar
.ENDSECT

****************************** MÓDULO MFTABLE ******************************
```

```
;Este módulo contiene la descripción de la tabla de la función de membresía y ;busca la
instrucción LAID. Este módulo esta designado a INPAGE, en consecuencia ;el código
completo y los datos de este módulo están contenidos en un bloque de  :256-byte de la
memoria de programa del COP8.
.SECT MFTABLE,ROM,REL,INPAGE
LUP:
        LAID                        ;Buscar dato
        RET                         ;Regresar (ACC = data)
                                    ;Entrada de la tabla de función de membresía


IMFTBL:
       .BYTE  000, 000, 000, 016, 099, 0f7, 000, 06f, 01e
       .BYTE  000, 023, 066, 0af, 0f3, 0ff, 01a, 074, 01a
       .BYTE  018, 085, 0ff, 0ff, 0ff, 0ff, 01a, 069, 000
                                   ;Tamaño máximo = 28 x 9


.ENDSECT
;****************************** MÓDULO RULECODE  ******************************
;Este módulo evalúa las reglas difusas y calcula la salida difusa.
;ENTRADAS:    I1MF1-I#MF# (fuzzy input table)
;ALTERADAS: A,B,X,HC,C,MULT Regs,RCNT1,RCNT2,RINDEX,RPAGE,CNT,SGNFLG,TEMP,PA-PWL
;SALIDAS:   OUT1,OUT2,OUT3,OUT4,OUT5
;
.SECT RULECODE,ROM
RULEVAL:
        LD      B,#OUT1             ;Puntero a el LSB de la salida final
        LD      [B+],#00            ;Establecer la salida a cero
        LD      [B+],#00
        LD      [B+],#00
        LD      [B+],#00
        LD      [B],#00
        LD      RCNT1,#00           ;Inicializar el contador de reglas a cero
EVAL:   JSR     RULELUP             ;Buscar regla
        JSR     RDOM                ;Encontrar DoM para cada antecedente
        ;Check if any DoM is zero
        LD      B,#PA               ;Conprobar si algun DoM es cero
        LD      A,[B+]
        IFEQ    A,#00               ;Si DoM de I1 = cero
        JP      IRULE               ;entonces ir a la segunda regla
        JSR     COUT                ;Calcular la salida de la regla y
                                    ;sumarla a la salida final
IRULE:  LD      B,#RCNT1            ;Incrementar el contador de la regla
```

```
        LD      A,[B]
        INC     A
        X       A,[B]                   ;Guardar el incremento del contador
        LD      A,[B]
        IFEQ    A,#RULECNT1             ;Si el contador = número de reglas
        RET                             ;entonces regresar (evaluzación de reglas
                                        ; completada)
        JP      EVAL                    ;si no evaluar la siguiente regla

;*** RDOM ***
;Encontrar el grado de membrsia para el antecedente de la regla
;
RDOM:                                   ;Encontrar DoM para cada antecedente
        LD      A,PA                    ;Cargar A con el antecedente de I1 del
                                        ;número de MF
        ADD     A,#I1MF1                ;Sumar la dirección de inicio del número
                                        ;de MF al índice de MF
        X       A,B                     ;Almacenar la dirección en B
        LD      A,[B]                   ;Buscar DoM del antecedante de I1
        X       A,PA                    ;Guardar DoM del antecedante de I1 en PA
        RET

;*** COUT***
;Calcular la salidaa para una regla y sumar el resultado a la salida final
COUT:
        LD      SGNFLG,#00
        LD      B,#MUL1
        LD      A,PA
        X       A,[B]           ;Almacenar el resultado del antecedante en MUL1
MWGT:   LD      B,#PWL          ;Multiplicar el resultado del antecedente por el
                                ;peso de la salida
        LD      A,[B-]
        X       A,MULT1         ;Cargar el multiplicador byte-bajo del peso
                                ;de la salida
                ;Comprobzar el signo de PWU (byte-alto del peso de salida)
        IFBIT   SB,[B]          ;Si SB = 1
        SBIT    0,SGNFLG        ;entonces establecer la bandera de signo
        LD      A,[B]           ;Remover el bit de signo de PWU
        AND     A,#07F
        X       A,MULT2         ;Cargar el multiplicador byte-alto del peso de
                                ;la salida
        JSR     MUL16           ;Multiplicar (Resultado = PROD3:PROD2:PROD1)
```

```
SUMM:   LD      B,#PROD4        ;Limpiar los 2 bytes superiores de la salida
        LD      [B+],#00        ; (resultado máximo es en 3 bytes)
        LD      [B],#00
        IFBIT   0,SGNFLG        ;Comprobar la bandera de signo (1=negativo)
        JP      CMPL            ;entonces encontrar el complement0 a  2's de la
                                ;regla de salida
SADD:   RC                      ;Sumar la contribución de la regla a la salida
        LD      B,#OUT1         ;Apuntar al 1er byte de la salida final
        LD      X,#PROD1        ;Apuntar al 1er byte de la regla de salida
        LD      A,[X+]          ;A = PROD1
        ADC     A,[B]           ;A = PROD1 + OUT1 + C
        X       A,[B+]          ;Almacenar resultado en OUT1
        LD      A,[X+]          ;A = PROD2
        ADC     A,[B]           ;A = PROD2 + OUT2 + C
        X       A,[B+]          ; Almacenar resultado en OUT2
        LD      A,[X+]          ;A = PROD3
        ADC     A,[B]           ;A = PROD3 + OUT3 + C
        X       A,[B+]          ; Almacenar resultado en OUT3
        LD      A,[X+]          ;A = PROD4
        ADC     A,[B]           ;A = PROD4 + OUT4 + C
        X       A,[B+]          ; Almacenar resultado en OUT4
        LD      A,[X+]          ;A = PROD5
        ADC     A,[B]           ;A = PROD5 + OUT5 + C
        X       A,[B]           ; Almacenar resultado en OUT5
        RET
CMPL:   LD      B,#PROD1        ;El formato es complementado
        LD      A,[B]
        XOR     A,#0FF
        X       A,[B+]
        LD      A,[B]
        XOR     A,#0FF
        X       A,[B+]
        LD      A,[B]
        XOR     A,#0FF
        X       A,[B+]
        LD      A,[B]
        XOR     A,#0FF
        X       A,[B]
        SC                      ;Sumar uno al formato del complemento 2's
```

```
        LD      B,#PROD1
        CLRA
        ADC     A,[B]           ;Producto 1
        X       A,[B+]
        CLRA
        ADC     A,[B]           ;Producto 2
        X       A,[B+]
        CLRA
        ADC     A,[B]           ;Producto 3
        X       A,[B+]
        CLRA
        ADC     A,[B]           ;Producto 4
        X       A,[B+]
        CLRA
        ADC     A,[B]           ;Producto 5
        X       A,[B]
        JMP     SADD        ;Ir a la suma del complemento 2's para la salida final
.ENDSECT

;******************************** MÓDULO RULE *********************************
.SECT RULE,ROM,REL,INPAGE
;Determina la localidad de la tabla de reglas para una regla dada
RULELUP:                        ;Todas las reglas en un  bloque de 256 byte
        LD      X,#PA           ;Apuntar a la localidad de almacenamiento para el
                                ;primer antecedente
        LD      B,#RCNT1
        LD      A,[B]           ;entrada-1 ó entrada-2 -> 3 bytes por regla
        ADD     A,[B]           ;Multiplicar Rulecnt indexado por 3
        ADD     A,[B]
       LD     B,#TEMP             ;Cargar B con la dirección de localidad temporal
       X      A,[B]        ;Almacenar el índice de la regla en la localidad temporal
       JMP    LUPRS0              ;Buscar regla

.ENDSECT
;******************************** MÓDULOS RULEn *******************************
;Las rutinas LUPRSn (n = 0,1, ...) buscaen el antecedente y el consecuente
;para una regla. La regla es almacenada de PA hasta PWL.
;ENTRADS:X     = dirección de la localidad de almacenamiento para el 1er
;                antecedente
;         Temp = índice de regla
;         B    = dirección de Temp
;ALTERADAS: X, CNT, A, B, Temp
```

```
;SALIDA:  PA   = 1st antecedente MF#
;         PB   = 2nd antecedente MF# (si esta presente)
;         PC   = 3rd antecedente MF# (si esta presente)
;         PD   = 4th antecedente MF# (si esta presente)
;         PWU  = bit de signo y byte superior del peso de la salida
;         PWL  = byte bajo del peso de la salida


; Este módulo contiene la Tabla de Regla 0

;*** LUPRS0 ***
.SECT RULE0,ROM,REL,INPAGE
LUPRS0:
      LD      A,[B]                 ;Cargar el A el índice de la regla
      ADD     A,#L(RTBL0)           ;Calcula la dirección del 1er antecedente
      X       A,[B]                 ;Guarda la dirección del antecedente
      LD      A,[B]                 ;Carga A con la dirección del antecedente
      LAID                          ;Buscar antecedente
      AND     A,#0F0                ;Mascara del nibble bajo de salida
      SWAP    A                     ;Intercambiar el antecedente al nibble bajo
      X       A,PA                  ;Guardar antecedente
      LD      A,[B]                 ;Cargar A con la dirección del antecedente
      INC     A                     ;Incrementar dirección al puntero de PWU
      X       A,[B]                 ;Guardar dirección
      LD      A,[B]                 ;Recuperar dirección a A
      LAID                          ;Buscar PWU
      X       A,PWU                 ;Guardar PWU
      LD      A,[B]                 ;Cargar A con dirección de PWU
      INC     A                     ;Incrementar dirección a el puntero de PWL
      LAID                          ;Buscar PWL
      X       A,PWL                 ;Guardar PWL
      RET

RTBL0:

      .BYTE  020, 061, 04b
      .BYTE  010, 01a, 0f4
      .BYTE  000, 023, 072

.ENDSECT
;******************************** MODULO MATH *********************************
```

```
;Este módulo contiene las rutinas de multiplicación y división usadas en la
;difusificación y el proceso de evaluación de reglas.
;
.PUBLIC DIV,MUL16,MUL08              ;Declaración global para el uso por el código
                                     ;de la aplicación
.SECT MATH,ROM

;*** DIV ***
;DIVIDE (16-bit by 8-bit)
;Divisor    =  DIVR1
;Dividend   =  DIV2:DIV1
;Quotient   =  QUO2:QUO1  OR DIV2:DIV1
;Remainder  =  REM1
;ALTER: A,B,X,CNT,DIV2,DIV1,REM1,REM2
.LOCAL
DIV:    LD      CNT,#16
        LD      B,#REM1
        LD      [B+],#00
        LD      [B],#00
        LD      X,#REM1
LSHFT:  RC
        LD      B,#DIV1
        LD      A,[B]
        ADC     A,[B]
        X       A,[B+]
        LD      A,[B]
        ADC     A,[B]
        X       A,[B+]
        LD      A,[B]
        ADC     A,[B]
        X       A,[B+]
        LD      A,[B]
        ADC     A,[B]
        X       A,[B]
TSUBT:  SC
        LD      B,#DIVR1
        LD      A,[X+]
        SUBC    A,[B]
        LD      A,[X-]
        SUBC    A,#00
        IFNC
        JP      $TEST
```

```
SUBT:   LD      A,[X]
        SUBC    A,[B]
        X       A,[X+]
        LD      A,[X]
        SUBC    A,#00
        X       A,[X-]
        LD      B,#DIV1
        SBIT    0,[B]
$TEST:  DRSZ    CNT
        JMP     LSHFT
        RET


;*** MUL16 ***
;MULTIPLICACION (16-bit by 8-bit)
;Multiplicando =   MUL1
;Multiplicador =   MULT2:MULT1
;Productos     =   PROD3:PROD2:PROD1 6 PROD3:MULT2:MULT1
;ALTERADOS: A,B,X,MULT1,MULT2,CNT
.LOCAL
MUL16:
        LD      CNT,#17
        LD      B,#PROD3
        LD      [B],#00
        LD      X,#MUL1
        RC
$MLOOP: LD      A,[B]
        RRC     A
        X       A,[B-]
        LD      A,[B]
        RRC     A
        X       A,[B-]
        LD      A,[B]
        RRC     A
        X       A,[B]
        LD      B,#PROD3
        IFNC
        JP      $TEST
        RC
        LD      A,[X]
        ADC     A,[B]
        X       A,[B]
```

```
$TEST:  DRSZ    CNT
        JP      $MLOOP
        RET

;*** MUL08 ***
;MULTIPLICACION (8-bit by 8-bit)
;Multiplicando =   MUL1
;Multiplicador =   MULT1
;Producto      =   PROD2:PROD1 or MULT2:MULT1
;ALTERADOS: A,B,X,MULT1,MULT2,CNT
.LOCAL
MUL08:
        LD      CNT,#9
        LD      B,#PROD2
        LD      [B],#00
        LD      X,#MUL1
        RC
$MLOOP: LD      A,[B]
        RRC     A
        X       A,[B-]
        LD      A,[B]
        RRC     A
        X       A,[B]
        LD      B,#PROD2
        IFNC
        JP      $TEST
        RC
        LD      A,[X]
        ADC     A,[B]
        X       A,[B]
$TEST:  DRSZ    CNT
        JP      $MLOOP
        RET
.ENDSECT
.END
```

B.2 Listado del archivo PRINCIPAL.ASM

```
.INCLD cop8sac.inc
;***************** Declaración De Variables Difusas Externas ********************

.SECT FUZDAT,RAM
.EXTRN I1
.EXTRN OUT1, OUT2, OUT3, OUT4, OUT5
.SECT FUZZ,ROM
.EXTRN FUZZ
.ENDSECT

********************* SECCIÓN PRINCIPAL DEL PROGRAMA ***************************

.SECT PRINCIPAL,ROM

;********************* Configuración inicial

inicio:   CLR   A
          LD    PORTCC,#0FC   ;PORTCC =0xF6;
                              ;Escritura(L3=L2=1),Lectura(L1=L0=0)
          LD    PORTD,#00     ;PORTD  = 0x00. Inicializando el DAC
          LD    PORTLC,#00    ;PORTLC = 0x00. Puerto L solo de lectura

;********************* Bucle Principal

repetir:  LD    PORTCD,#0F0   ;PORTCD = 0xF0. Puerto C configuración de lectura

;********************* Dato Acondicionado

avisoda:  LD    A,PORTCP      ;Esperando aviso
          AND   A,#003
          IFNE  A,#003
          JP    avisoda

          LD    PORTFC,#00    ;PORTFC = 0x00. Configuración de lectura Puerto F
          LD    PORTFD,#00    ;PORTFD = 0x00
          LD    A,PORTFP      ,I1 = PORTFP. Leyendo Dato acondicionado

          X     A,I1          ;Llamada del código difuso
```

```
          JSR     FUZZ

;******************** Envío De Dato Difuso

          LD      PORTFC,#0FF    ;PORTFC = 0xFF. Puerto F como salida

          LD      A,OUT1         ;PORTFD = OUT1
          X       A,PORTFD
          LD      PORTCD,#0FC    ;PORTCD = 0xFC
confird1: LD      A,PORTCP       ;Esperando confirmación de dato 1
          AND     A,#003
          IFNE    A,#002
          JP      confird1

          LD      A,OUT2         ;PORTFD = OUT2
          X       A,PORTFD
          LD      PORTCD,#0F8    ;PORTCD = 0xF8
confird2: LD      A,PORTCP       ;Esperando confirmación de dato 2
          AND     A,#003
          IFNE    A,#001
          JP      confird2

          LD      A,OUT3         ;PORTFD = OUT3
          X       A,PORTFD
          LD      PORTCD,#0F4    ;PORTCD = 0xF4
confird3: LD      A,PORTCP       ;Esperando confirmación de dato 3
          AND     A,#003
          IFNE    A,#003
          JP      confird3

          LD      A,OUT4         ;PORTFD = OUT4
          X       A,PORTFD
          LD      PORTCD,#0FC    ;PORTCD = 0xFC
confird4: LD      A,PORTCP       ;Esperando confirmación de dato 4
          AND     A,#003
          IFNE    A,#002
          JP      confird4

          LD      A,OUT5         ;PORTFD = OUT5
          X       A,PORTFD
          LD      PORTCD,#0F8    ;PORTCD = 0xF8
confird5: LD      A,PORTCP       ;Esperando confirmación de dato 5
```

```
            AND     A,#003
            IFNE    A,#001
            JP      confird5

;******************** Recibiendo el voltaje de entrada

recibeve:   LD      A,PORTCP        ;Esperando aviso
            AND     A,#003
            IFNE    A,#002
            JP      recibeve

            LD      PORTFC,#00      ;PORTFC = 0x00. Configuración lectura Puerto F
            LD      PORTFD,#00      ;PORTFD = 0x00
            LD      A,PORTFP        ;PORTD = PORTFP. Enviando voltaje al DAC
            X       A,PORTD

;******************** Monitoreo

            LD      PORTFC,#0FF     ;PORTFC = 0xFF. Configuración escritura Puerto F
monitor:    LD      A,PORTLP        ;PORTFD = PORTLP. Leyendo velocidad del ADC
            X       A,PORTFD        ;PORTCD = 0x04. Enviando velocidad a la PC
            LD      PORTCD,#0FC     ;PORTCD = 0xFC

            LD      A,PORTCP        ;Continuar hasta aviso
            AND     A,#003
            IFEQ    A,#001
            JP      regresar
            JP      monitor

regresar:   JMP     repetir         ;Fin del Bucle Principal

.ENDSECT
.END inicio
```

Apéndice C Hojas de especificaciones

C.1 DAC0800 (Convertidor Digital – Analógico)

C.2 ADC0804 (Convertidor Analógico –Digital)

C.3 Transmisor / Receptor SN74LS245

C.4 Configuración del Puerto Paralelo

National Semiconductor

A a D, D a A

Convertidores digitales—analógicos de 8 bits DAC0800, DAC0801, DAC0802

Descripción general

Las series DAC0800 son convertidores digitales analógicos monolíticos de 8 bits de corriente de salida y alta velocidad, que tienen como característica tiempos de asentamiento de 100 nS. Cuando se utiliza como un CAD multiplicador, es posible un funcionamiento monotónico superior a 40 a 1 sobre un rango de corriente de referencia. Las series DAC0800 también tienen como característica salidas de corriente complementarias de alta compliancia para permitir voltajes de salida diferencial de 20 Vp-p con resistores de carga simples como se muestra en la figura 1. Una corriente de acoplamiento referida a escala total mayor de ± 1 LSB elimina la necesidad de potenciómetros de ajuste en la mayoría de las aplicaciones mientras que las no linealidades mejores del ±0.1% con la temperatura minimizan los acumulaciones de error del sistema. Las entradas inmunes a ruido de las series DAC0800 aceptarán niveles de LTT con la terminal de umbral lógico, V_{LC}, terminal 1, a tierra. Ajustes simples del potencial V_{LC} permiten interfase con todas las familias lógicas. El funcionamiento y las características del dispositivo se mantienen esencialmente sin cambio sobre el rango total de voltaje de alimentación ± 4.5 V_a ± 18V, la disipación de potencia es únicamente de 33 mW con alimentaciones ±5V y es independiente de los estados de entrada lógicos.

Los DAC0800, DAC0802, DAC0800C, DAC0801C y DAC0802C son un reemplazo directo respectivamente, para los DAC-08, DAC-08A, DAC-08C, DAC-08E y DAC-08H.

Características

- Corriente de salida de asentamiento rápida 100 ns
- Error a escala total ±1 LSB
- No linealidad con la temperatura ±0.1%
- Corrimiento de corriente a escala total ±10 ppm/°C
- Compliancia de salida alta −10V a +18V
- Salidas de corrientes complementarias
- Interfase directa con LTT, CMOS, PMOS y otros
- Capacidad de multiplicación en un amplio rango en 2 cuadrantes
- Amplio rango de voltaje de alimentación ±4.5 a ±18V
- Bajo consumo de potencia 33 mW a ±5V
- Bajo costo

Aplicaciones típicas

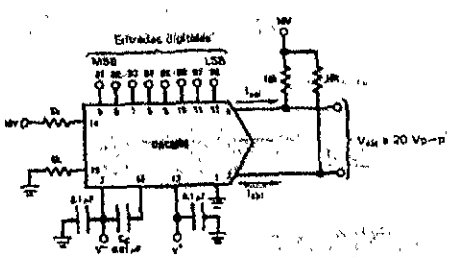
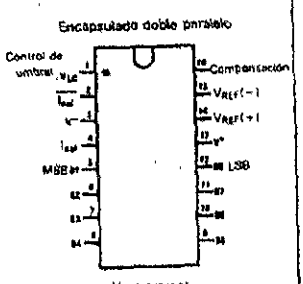


Figura 1 Convertidor digital analógico con salida de 20 Vp-p

Diagrama de conexiones



Información de órdenes

No linealidad	Rango de temperatura	Número de orden*					
		Encapsulado D (D16C)		Encapsulado J (J16A)		Encapsulado N (N16A)	
±0.1% ET	−55°C ≤ T_A ≤ +125°C	DAC0802LD	DAC-08AQ	DAC0802LCJ	DAC-08HQ	DAC0802LCN	DAC-08HP
±0.1% ET	0°C ≤ T_A ≤ +70°C						
±0.19% ET	−55°C ≤ T_A ≤ +125°C	DAC0800LD	DAC-08Q				
±0.19% ET	0°C ≤ T_A ≤ +70°C			DAC0800LCJ	DAC-08EQ	DAC0800LCN	DAC-08EP
±0.39% ET	0°C ≤ T_A ≤ +70°C			DAC0801LCJ	DAC-08CQ	DAC0801LCN	DAC-08CP

*Nota Los dispositivos pueden ordenarse utilizando uno u otro número de orden

Capacidades Máximas Absolutas

Voltaje de Alimentación	±18V a 36V
Disipación de Potencia	500 mW
Referencia de Voltaje de Entrada Diferencial (V14 a V15)	V^{--} a V^{+}
Referencia de Rango de Entrada de Modo Común (V14, V15)	V^{--} a V^{+}
Referencia de Corriente de Entrada	5 mA
Entradas Lógicas	V^{--} a V^{--} plus 36V
Salidas Corriente Analógica	*Figura 2a*
Temperaturas de Almacenamiento	−65°C a + 150°C
Temperatura de Terminal (Soldando, 10 segundos)	300°C

Condiciones de Operación

	MIN	MÁX	UNIDADES
Temperatura (T_A)			
DAC0802L	−55	+125	°C
DAC0800L	−55	+125	°C
DAC0800LC	0	+70	°C
DAC0801LC	0	+70	°C
DAC0802LC	0	+70	°C

Características Eléctricas (V_A = + 15V, I_{REF} = 2 mA, $T_{MIN} \le T_A \le T_{MAX}$ a menos que se especifique otra cosa. Las características de salida se refieren a ambas I_{OUT} e I_{OUT}.

PARÁMETRO	CONDICIONES	DAC0802L/ DAC0802LC			DAC0800L/ DAC0800LC			DAC0801LC			UNIDADES
		MIN	TYP	MÁX	MIN	TYP	MÁX	MIN	TYP	MÁX	
	Resolución	8	8	8	8	8	8	8	8	8	Bits
	Monotonicidad	8	8	8	8	8	8	8	8	8	Bits
	No linealidad			±0.1			±0.19			±0.39	%FS
T_A	Tiempo de asentamiento	A ± 1/2 LSB, todos los Bits Conmutados "ENC" o "APAG" T_A = 25°C		100	135				100	150	ns
		DAC0800L				100	135				ns
		DAC0800LC				100	150				ns
T_{PHL}, T_{PLH}	Retraso de propagación	T_A = 25°C									
	Cada bit			35	60		35	60		35	60	ns
	Todos los Bits Conmutados			35	60		35	60		35	60	ns
TC/T_D	Tempco a escala total			±10	±50		±10	±10		±10	±80	ppm/°C
V_{OC}	Complimiento de voltaje de salida	Cambio de corriente a escala total ≤ 1/2 LSB, R_{out} TYP > 20 MΩ	−10		18	−10		18	−10		18	V
I_{FS4}	Corriente a escala total	V_{REF} = 10.000V, R14 = 5.000 kΩ, R15 = 5.000 kΩ, T_A = 25°C	1.984	1.992	2.000	1.94	1.99	2.04	1.94	1.99	2.04	mA
I_{OFS}	Simetría a escala Total	$I_{FS4} - I_{FS2}$		±0.5	±4.0		±1	±8.0		±2	±16	μA
I_{ZS}	Corriente a escala cero			0.1	1.0		0.2	2.0		0.2	4.0	μA
I_{OBR}	Rango de corriente de salida	V^{--} = −5V	0	2.0	2.1	0	2.0	2.1	0	2.0	2.1	mA
		V^{--} = −8V a −18V	0	2.0	4.2	0	2.0	4.2	0	2.0	4.2	mA
	Niveles de entrada Lógica											
V_{IL}	"0" Lógico	V_{CL} = 0V			0.8			0.8			0.8	V
V_{IH}	"1" Lógico		2.0			2.0			2.0			V
	Corriente entrada Lógica	V_{CL} = 0V										
I_{IL}	"0" Lógico	−10V ≤ V_{ENT} ≤ +0.8V	2.0	−2.0	−10		−2.0	−10		−2.0	−10	μA
I_{IH}	"1" Lógico	2V ≤ V_{ENT} ≤ +18V		0.002	10		0.002	10		0.002	10	μA
V_{CS}	Excursión Entrada Lógica	V^{--} = −18V	−10		18	−10		18	−10		18	V
V_{THR}	Rango de Umbral Lógico	V_S = ±15V	−10		13.5	−10		13.5	−10		13.5	V
I_{15}	Corriente de Polarización de Referencia			−1.0	−3.0		−1.0	−3.0		−1.0	−3.0	μA
SR	Velocidad de Barrido de Retroinicio	*(Figura 3d)*	4.0	8.0		4.0	8.0		4.0	8.0		mA/μs
$PSSI_{FS}$	Sensitividad a la Alimentación	4.5V ≤ V^{+} ≤ 18V		0.0001	0.01		0.0001	0.01		0.0001	0.01	% %FS
$PSSI_{ZS}$		−4.5V ≥ V^{--} ≥ −18V		0.0001	0.01		0.0001	0.01		0.0001	0.01	%/%
	Corriente de Alimentación											
I^{+}		I_{REF} = 1mA		2.3	3.8		2.3	3.8		2.3	3.8	mA
I^{--}		V_A = ±5V, I_{REF} = 1mA		−4.3	−5.8		−4.3	−5.8		−4.3	−5.8	mA
		V_A = 5V, −15V, I_{REF} = 2mA										
I^{+}				2.4	3.6		2.4	3.8		2.4	3.8	mA
I^{--}				−6.4	−7.6		−6.4	−7.8		−6.4	−7.8	mA
		V_A = ±15V, I_{REF} = 2mA										
I^{+}				2.6	3.8		2.6	3.8		2.6	3.8	mA
I^{--}				−6.6	−7.8		−6.5	−7.8		−6.5	−7.8	mA
I^{--}		+ 5V, I_{REF} = 1mA		33	48		33	48		33	48	mW
P_D	Disipación de Potencia	5V, −15V, I_{REF} = 2mA		108	138		108	138		108	135	mW
		±15V, I_{REF} = 2mA		138	174		138	174		136	174	mW

Nota 1: La máxima temperatura de unión de los DAC0800, DAC0801 y DAC0802 es 175°C. Para operar a temperaturas más elevadas a dispositivos en el encapsulado doble cerámico J o D debe considerarse una pérdida de disipación de potencia sobre la base de una resistencia térmica de 100°C/W, de la unión al ambiente. 175°C/W para el encapsulado doble paralelo moldeado N.

Características de funcionamiento típicas

Corriente a escala total vs. corriente de referencia

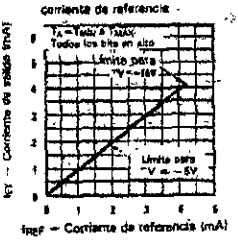


Figura 3

Retraso de propagación del LSB vs. I_{RT}

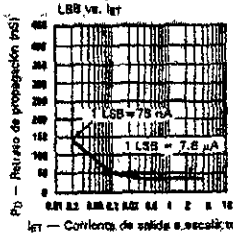
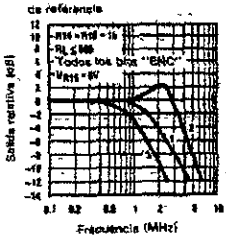


Figura 4

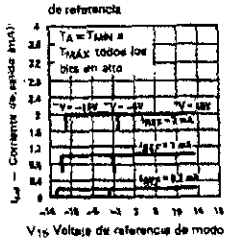
Respuesta en frecuencia entrada de referencia



Curva 1. $C_C = 15$ pF, $V_{ent} = 2\ V_{p-p}$ centrado en 1 V.
Curva 2. $C_C = 15$ pF, $V_{ent} = 50\ mV_{p-p}$ centrado en 200 mV
Curva 3. $C_C = 0$ pF, $V_{ent} = 100\ mV_{p-p}$ en 0 volts y a través de 50Ω conectados a la terminal 14, 2V aplicados en R14

Figura 5

Rango de modo común del ampli. de referencia



Corriente de entrada lógica vs. voltaje de entrada

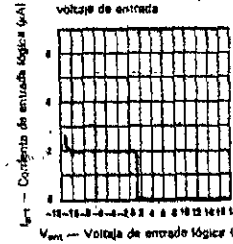


Figura 7

$V_{BIAS} - V_{LC}$ vs. temperatura

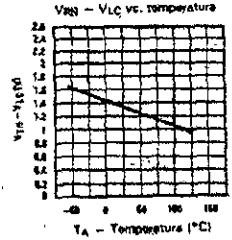


Figura 8

Nota: El rango positivo de modo común siempre es $(V_- - 1.5\ V)$

Figura 6

Corriente de salida vs. voltaje de salida (compliancia del voltaje de salida)

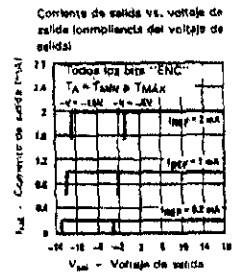


Figura 9

Compliancia del voltaje de salida vs. temperatura

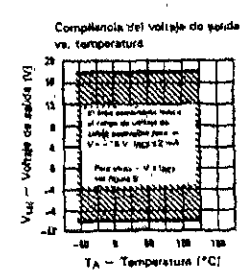
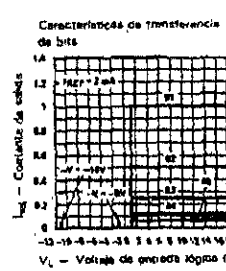


Figura 10

Características de transferencia de bits



Nota: B1 – B8 Tienen características de transferencia idénticas. Los bits están rastreados conmutados con menos de 1/2 LSB de error, a menos de ± 100 mV del umbral efectivo. Se garantiza que estos puntos de conmutación se mantienen entre 0.8 y 2 V sobre el rango total de temperatura ($V_{LC} = 0V$)

Figura 11

Características de funcionamiento típicas (Continuación)

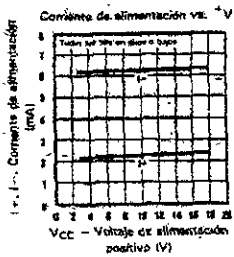


Figura 12

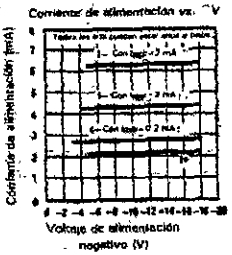


Figura 13

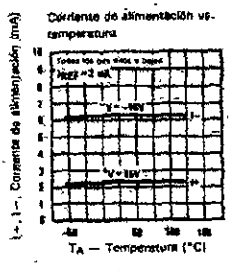
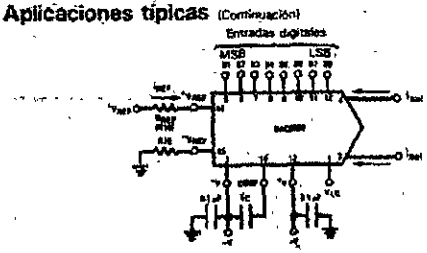


Figura 14

Aplicaciones típicas (Continuación)



$$I_{ET} = \frac{+V_{REF}}{R_{REF}} \times \frac{255}{256}$$

$I_{on} + I_{out} = I_{ET}$ para todos los estados lógicos

Los valores típicos para operación, LTT, Referencia fija, son:
$V_{REF} = 10.000V$
$R_{REF} = 5.000k$
$R15 = R_{REF}$
$C_C = 0.01\ \mu F$
$V_{LC} = 0V$ (tierra)

Figura 15. Operación referencia positiva básica.

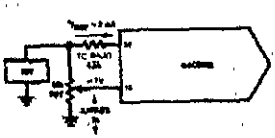
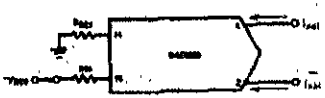


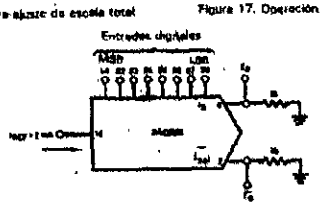
Figura 16. Circuito recomendado para ajuste de escala total



$$I_{ET} = \frac{-V_{REF}}{R_{REF}} \times \frac{255}{256}$$

Nota. R_{REF} ajusta I_{ET}; R 15 es para cancelación de la corriente de polarización.

Figura 17. Operación referencia negativa básica



	B1	B2	B3	B4	B5	B6	B7	B8	I_{out} mA	$\overline{I}_{out}$ mA	E_{out}	$\overline{E}_{out}$
Escala total	1	1	1	1	1	1	1	1	1.992	0.000	−9.960	0.000
Escala total − LSB	1	1	1	1	1	1	1	0	1.984	0.008	−9.920	−0.040
Media escala + LSB	1	0	0	0	0	0	0	1	1.008	0.984	−5.040	−4.920
Media escala	1	0	0	0	0	0	0	0	1.000	0.992	−5.000	−4.960
Media escala − LSB	0	1	1	1	1	1	1	1	0.992	1.000	−4.960	−5.000
Escala cero + LSB	0	0	0	0	0	0	0	1	0.008	1.984	−0.040	−9.920
Escala Cero	0	0	0	0	0	0	0	0	0.000	1.992	0.000	−9.960

Figura 18. Operación unipolar negativa básica

National Semiconductor

A a D, D a A

Convertidores A/D compatibles con μP de 8 bits ADC0801, ADC0802, ADC0803, ADC0804, ADC0805

Descripción general

El ADC0801, ADC0802, ADC0803, ADC0804 y ADC0805 son convertidores A/D de aproximaciones sucesivas de 8 bits CMOS los cuales utilizan red potenciométrica diferencial similar a los productos 256R. Estos convertidores están diseñados para permitir la operación con los canales de control derivativo NSC800 y INS8080A y enganchadores de salida de tres estados TRI-STATE® que manejan directamente el canal de datos. Estos convertidores A/Ds se parecen al microprocesador como localidades de memoria o puertos de E/S y no es necesaria lógica de interfase.

Un nuevo voltaje de entrada diferencial analógico permite incrementar el rechazo de modo común y ocorrer el valor analógico de voltaje de entrada cero. Además, la entrada de voltaje de referencia puede ajustarse para permitir la codificación del tramo más pequeño de voltaje analógico a la resolución total de 8 bits.

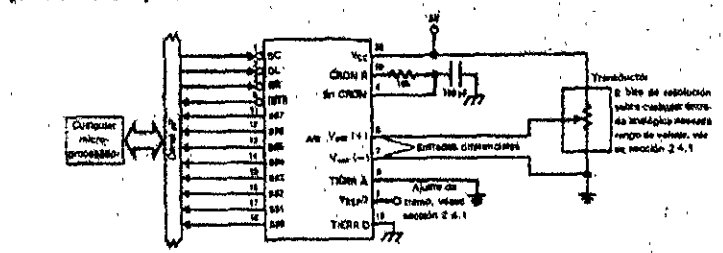
Características

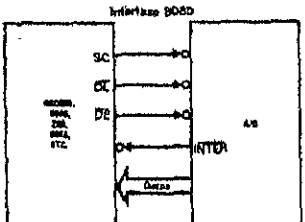
- Compatible con los derivados del μP 8080 — no es necesaria lógica de interfase — tiempo de acceso 135 ns
- Interfase fácil con todos los microprocesadores, o opera en forma autónoma.
- Voltajes analógicos de entrada diferenciales.
- Las entradas y salidas lógicas cumplen ambas especificaciones de nivel de voltaje MOS y LT²
- Trabaja con un voltaje de referencia de 2.5V (LM336)
- Generador de reloj en la misma pastilla
- Rango de voltaje de entrada analógico de 0 a 5V con alimentación única de 5V.
- No requiere ajuste de cero
- Encapsulado normal EDP con 20 terminales de 0.3"
- Opera radiométricamente o con 6 V_{CC}, 2.5 V_{CC}, o un tramo analógico ajustado como referencia de voltaje

Especificaciones clave

- Resolución: 8 bits
- Error Total: ±1/4 LSB, ±1/2 LSB y ±1 LSB
- Tiempo de conversión: 100 μs

Aplicaciones típicas





Especificación de error (incluye escala total, error de cero y no linealidad)			
Número de parte	Escala total ajustada	$V_{REF}/2 = 2.500V_{CD}$ (SIN AJUSTES)	$V_{REF}/2$ = sin conexión (SIN AJUSTES)
ADC0801	± 1/4 LSB	± 1/2 LSB	± 1 LSB
ADC0802	± 1/2 LSB	± 1 LSB	
ADC0803			
ADC0804			
ADC0805			

® TRI-STATE es una marca registrada de National Semiconductor

Capacidades máximas absolutas (Notas 1 y 2)

Voltaje de Alimentación (V_{CC}) (Nota 3)	6 5V
Voltaje	
Entradas de Control Lógicas	−0.3V a +18V
En Otras Entradas y Salidas	−0.3V a (V_{CC} + 0.3V)
Rango de Temperatura de Operación	−65°C a +150°C
Disipación de Encapsulado en T_A = 25°C	875 mW
Temperatura de Terminal (Soldando, 10 Segundos)	300°C

Capacidades de operación (Notas 1 y 2)

Rango de Temperatura	$T_{MIN} \le T_A \le T_{MAX}$
ADC0801/02LD	−55°C ≤ T_A ≤ +125°C
ADC0801/02/03/04/CD	−40°C ≤ T_A ≤ +85°C
ADC0801/02/03/05LCN	−40°C ≤ T_A ≤ +85°C
ADC0804LCN	0°C ≤ T_A ≤ +70°C
Rango de V_{CC}	4.5 V_{CD} a 6.3 V_{CD}

Características Eléctricas

Los siguientes especificaciones se aplican para V_{CC} = 5 V_{CD}, $T_{MIN} \le T_A \le T_{MAX}$ y fclk = 640 kHz a menos que se especifique otra cosa

PARÁMETRO	CONDICIONES	MÍN	TYP	MÁX	UNIDADES
ADC0801: Error ajustado total (Nota 6)	Con ajuste de escala total (Véase sección 2.5 2)			±1/4	LSB
ADC0802: Error sin ajustar total (Nota 6)	$V_{REF}/2$ = 2.500V_{CD}			±1/2	LSB
ADC0803: Error ajustado total (Nota 6)	Con ajuste de escala total (Véase sección 2.5.2)			±1/2	LSB
ADC0804: Error sin ajustar total (Nota 6)	$V_{REF}/2$ = 2.500V_{CD}			±1	LSB
ADC0805: Error sin ajustar total (Nota 6)	$V_{REF}/2$ = No conecta				LSB
Resistencia de entrada $V_{REF}/2$ (Terminal 9)	ADC0801/02/03/05 ADC0804 (Nota 8)	2.5 1.0	8.0 1.3		kΩ kΩ
Rango del voltaje de entrada analógica	(Nota 4) V (+) o V (−)	Tierra −0.05		V_{CC} + 0.05	V_{CD}
Error de modo común CD	Sobre el rango del voltaje de entrada analógica		±1/16	±1/8	LSB
Sensitividad a la fuente de alimentación	V_{CC} = 5V_{CD} ± sobre 10% V_{REF} (+) ≤ V_{IN} (−) permitido Rango de voltaje (Nota 4)		±1/16	±1/8	LSB

Características de CA

Los especificaciones siguientes se aplican para V_{CC} = 4V_{CD} y T_A = 25°C a menos que se especifique otra cosa

	PARÁMETRO	CONDICIONES	MÍN	TYP	MÁX	UNIDADES
T_C	Tiempo de conversión	f_{CRON} = 640 kHz (Nota 9)	103		114	μs
T_C	Tiempo de conversión	(Notas 5, 6)	66		73	$1/f_{CRON}$
f_{CRON}	Frecuencia de reloj	V_{CC} = 5V, (Nota 5)	100	640	640	kHz
	Ciclo de trabajo del reloj	(Nota 5)	40		60	%
VC	Velocidad de conversión en modo repetitivo	INTER unida a OL con SC = 0 V_{CD}, f_{CRON} = 640 kHz			8770	conv/s
t_W (OE)	Ancho de entrada de (Ancho del pulso de lectura)	SC = 0 V_{CD} (Nota 7)	100			ns
t_{ACC}	Tiempo de acceso (Retraso del flip de caída de OL a la salida de dato válido)	C_L = 100 pF		135	200	ns
t_{1H}, t_{0H}	Control de tres estados TRI STATE® (Retraso del flip de subida de OL al estado de alta Z)	C_L = 10 pF, R_L = 10kΩ (Véase circuitos de prueba de tres estados TRI STATE®)		125	200	ns
$t_{1,0}$, t_{H1}	Retraso del flip de caída de OE o OL al reinicio de INTR			300	450	ns
C_{IN}	Entradas de control del capacitador de entrada lógicas			5	7.5	pF
C_{OUT}	Salida tres estados TRI-STATE® Capacitancia (Acopladores de datos)			5	7.5	pF

Características Eléctricas

Las especificaciones siguientes se aplican para V_{CC}=5V$_{CD}$ y $T_{MIN} \leq T_A \leq T_{MÁX}$, a menos que se especifique otra cosa

PARÁMETRO		CONDICIONES	MÍN	TYP.	MÁX	UNIDADES
Entradas de control (Nota: (TERMINAL 4) Es la entrada de un circuito disparador Schmitt y, por tanto, se especifica separadamente)						
V_{enc} (1)	Voltaje de entrada "1" Lógico (Excepto terminal 4-en cron)	V_{CC}=5.25 V_{CD}	2.0		15	V_{CD}
V_{enc} (0)	Voltaje de entrada "0" Lógico (Excepto Terminal 4-en cron)	V_{CC}=4.75 V_{CD}			0.8	V_{CD}
I_{enc} (1)	Corriente de entrada "1" Lógico (todas las entradas)	V_{enc}=5V V_{CD}		0.005	1	μA_{CD}
I_{enc} (0)	Corriente de entrada "0" Lógico (todas las entradas)	V_{enc}=0 V_{CD}	−1	-0.005		μA_{CD}
Cronómetro de entrada y cronómetro R						
V_T^+	En cron (TERMINAL 4) Voltaje de umbral subiendo		2.7	3.1	3.5	V_{CD}
V_T^-	En cron (TERMINAL 4) Voltaje de umbral bajando		1.5	1.8	2.1	V_{CD}
V_H	En cron (TERMINAL 4) Histéresis ($V_T^+ - V_T^-$)		0.6	1.3	2.0	V_{CD}
V_{out} (0)	"0" Lógico del voltaje de salida cron (R)	I_S=360 μA V_{CC}=4.75 V_{CD}			0.4	V_{CD}
V_{out} (1)	"1" Lógico del voltaje de salida cron R	I_S=−360 μA V_{CC}=4.75 V_{CD}	2.4			V_{CD}
Salida de datos e INTER						
V_{out} (0)	"0" Lógico de voltaje de salida Salida de datos Salida de datos INTER	I_{DR}=1.6 mA, V_{CC}=4.75 V_{CD}			0.4	V_{CD}
		I_{out}=1.0 mA, V_{CC}=4.75 V_{CD}			0.4	V_{CD}
V_{out} (1)	"1" Lógico de voltaje de salida	I_S=−360μA, V_{CC}=4.75 V_{CD}	2.4			V_{CD}
V_{out} (1)	"1" Lógico de voltaje de salida	I_S=−10μA, V_{CC}=4.75 V_{CD}	4.5			V_{CD}
I_{out}	Fuga de salida (deshabilitada de tres estados-(todas las ópcopladores de datos)	V_{out}=0 V_{CD}	−3			μA_{CD}
		V_{out}=5 V_{CD}			3	μA_{CD}
$I_{aumento}$		V_{out} corto a tierra, T_A=25°C	4.5	6		mA_{CD}
$I_{contiene}$		V_{out} corto a V_{CC}, T_A=25°C	9.0	16		mA_{CD}
Suministro de energía						
I_{CC}	Corriente de alimentación (incluye corriente de la red potenciométrica)	f_{cron}=640 kHz $V_{REF}/2$=NC, T_A=25°C y $\overline{SC}$="1"				
		ADC0801/02/03/05		1.1	1.8	mA
		ADC0804 (Nota 9)		1.9	2.6	mA

Nota 1. Las capacidades máximas absolutas son aquellos valores debajo de los cuales la vida del dispositivo puede deteriorarse

Nota 2. Todos los voltajes se miden con respecto a tierra, a menos que se especifique otra cosa. El punto de tierra "A" separada siempre se debe alambrar al punto de tierra "D".

Nota 3. Internamente, existe un diodo Zener de V_{CC} a tierra y tiene un voltaje de ruptura típico de 7 V_{CD}

Nota 4. Para $V_{IN}(-) \geq V_{IN}(+)$ el código digital de salida será 0000 0000. Dos diodos en la pastilla del circuito serán unidos a cada una de las entradas analógicas (véase diagrama de bloques) los cuales conducirán directamente para voltajes de entrada analógicos a una caída de diodo por abajo de tierra o una caída de diodo mayor que el voltaje de alimentación V_{CC}. Sea precavido, durante pruebas a niveles más bajos de V_{CC}(4.5), como entradas analógicas de alto nivel (5V) puede ocasionar que este diodo de entrada conduzca especialmente a temperaturas elevadas, y ocasionar errores para entradas analógicas cercanos a la escala total. Es posible especular con que se permiten 50 mV de polarización directa de uno u otro diodo. Esto significa que V_{IN} no exceda el voltaje de alimentación por más de 50 mV, el código de salida será correcto. Por lo tanto, para lograr un rango de entrada absoluta de 0 V_{CD} a 5 V_{CD} se requiere un voltaje de alimentación mínimo de 4.950 V_{CD} sobre variaciones de temperatura, tolerancia y carga inicial.

Nota 5. Se garantiza la precisión en f_{cron}=640 kHz. A frecuencias de cronómetro mayores la precisión puede degradarse. A frecuencias de cronómetro menores, los límites del ciclo de trabajo pueden extenderse tanto como el intervalo mínimo del cronómetro en alto o en mínimo intervalo de tiempo del cronómetro en bajo no es mayor de 275 ns.

Nota 6. Con un pulso de inicio asincrono, pueden requerirse más de 8 ciclos de cronómetro antes de que los flases del cronómetro interno sean las adecuadas para iniciar el proceso de conversión. La solicitud de inicio se engancha internamente, ver la figura y la sección 2.0.

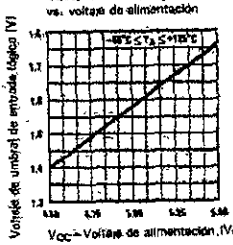
Nota 7. Se supone que la entrada de $\overline{SC}$ soporta la entrada de selección de $\overline{CS}$ y, por tanto, el tiempo depende del ancho del pulso $\overline{SC}$. Un ancho de pulso arbitrariamente amplio mantendrá el convertidor en el modo de reinicio y el inicio de conversión empezará en la transición de bajo a alto del pulso $\overline{SC}$ (Ver diagramas de tiempo)

Nota 8. Ninguno de estos convertidores A/D requiere un ajuste de cero (véase sección 2.5.1). Para obtener un código cero en otros voltajes de entrada analógicos ver la sección 2.5 y la figura 5.

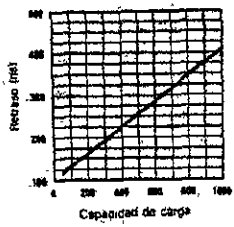
Nota 9. Para el ADC0804LCD el valor típico de resistencia de entrada de $V_{REF}/2$ es de 8kΩ y de I_{CC} es 1.1 mA

Características de funcionamiento típicas

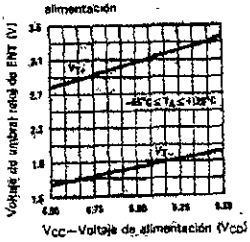
Voltaje de umbral de entrada lógica vs. voltaje de alimentación



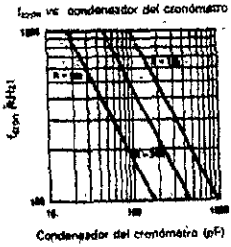
Retraso del flip de bajada de $\overline{INTR}$ a salida, dato válido vs. capacitancia de carga



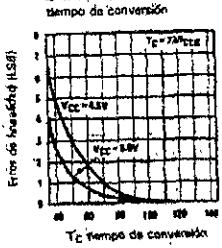
Niveles de excursión cronómetro de ENT SCHMITT vs. voltaje de alimentación



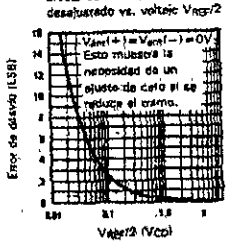
f_{error} vs. condensador del cronómetro



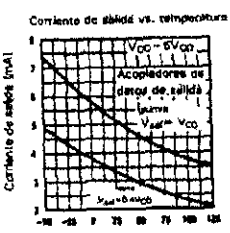
Error de escala total vs tiempo de conversión



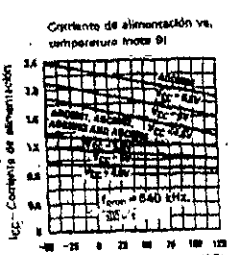
Efecto del error de desvío desajustado vs. voltaje $V_{REF}/2$



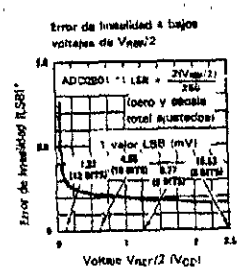
Corriente de salida vs. temperatura



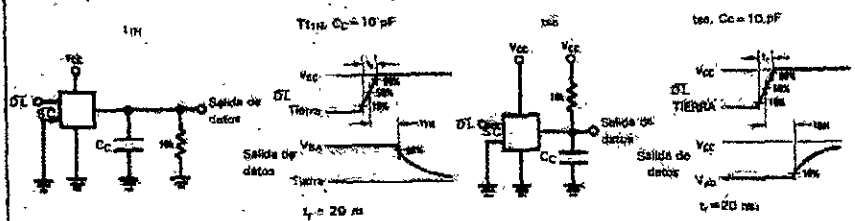
Corriente de alimentación vs. temperatura (nota 9)



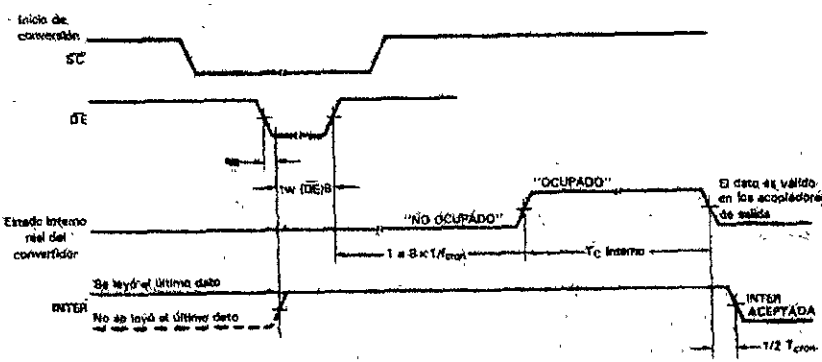
Error de linealidad a bajos voltajes de $V_{REF}/2$



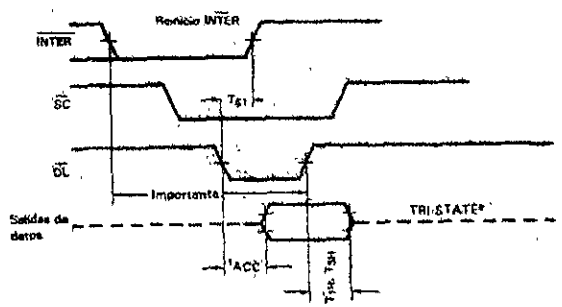
Formas de onda y circuitos prueba de tres estados



Diagramas de tiempo (todos los tiempos se miden desde los puntos de voltaje de 50%)



Salida habilitada y reinicio de $\overline{INTER}$



Nota: La selección de lectura debe ocurrir 8 pulsos ($8/f_{CLK}$) después de la afirmación de interrupción para garantizar el reinicio de $\overline{INTER}$

MOTOROLA

SN54/74LS245

OCTAL BUS TRANSCEIVER

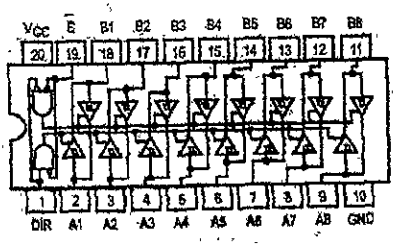
The SN54/74LS245 is an Octal Bus Transmitter/Receiver designed for 3-line asynchronous 2-way data communication between data buses. Direction input (DIR) controls transmission of Data from bus A to bus B or bus B to bus A depending upon its logic level. The Enable input (E) can be used to isolate the buses.

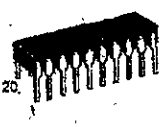
- Hysteresis Inputs to Improve Noise Immunity
- 2-Way Asynchronous Data Bus Communication
- Input Diodes Limit High-Speed Termination Effects
- ESD > 3500 Volts

OCTAL BUS TRANSCEIVER

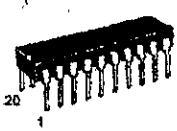
LOW POWER SCHOTTKY

LOGIC AND CONNECTION DIAGRAMS DIP (TOP VIEW)





J SUFFIX
CERAMIC
CASE 732-03



N SUFFIX
PLASTIC
CASE 738-03

DW SUFFIX
SOIC
CASE 751D-03

TRUTH TABLE

INPUTS		OUTPUT
E	DIR	
L	L	Bus B Data to Bus A
L	H	Bus A Data to Bus B
H	X	Isolation

H = HIGH Voltage Level
L = LOW Voltage Level
X = Immaterial

ORDERING INFORMATION

SN54LSXXXJ	Ceramic
SN74LSXXXN	Plastic
SN74LSXXXDW	SOIC

GUARANTEED OPERATING RANGES

Symbol	Parameter		Min	Typ	Max	Unit
V_{CC}	Supply Voltage	54 74	4.5 4.75	5.0 5.0	5.5 5.25	V
T_A	Operating Ambient Temperature Range	54 74	−55 0	25 25	125 70	°C
I_{OH}	Output Current — High	54, 74			−3.0	mA
		54 74			−12 −15	mA
I_{OL}	Output Current — Low	54 74			12 24	mA

DC CHARACTERISTICS OVER OPERATING TEMPERATURE RANGE (unless otherwise specified)

Symbol	Parameter		Limits Min	Typ	Max	Unit	Test Conditions	
V_{IH}	Input HIGH Voltage		2.0			V	Guaranteed input HIGH Voltage for All Inputs	
V_{IL}	Input LOW Voltage	54			0.7	V	Guaranteed input LOW Voltage for All Inputs	
		74			0.8			
$V_{T+}-V_{T-}$	Hysteresis		0.2	0.4		V	V_{CC} = MIN	
V_{IK}	Input Clamp Diode Voltage			−0.65	−1.5	V	V_{CC} = MIN, I_{IN} = −18 mA	
V_{OH}	Output HIGH Voltage	54, 74	2.4	3.4		V	V_{CC} = MIN, I_{OH} = −3.0 mA	
		54, 74	2.0			V	V_{CC} = MIN, I_{OH} = MAX	
V_{OL}	Output LOW Voltage	54, 74		0.25	0.4	V	I_{OL} = 12 mA	V_{CC} = V_{CC} MIN, V_{IN} = V_{IL} or V_{IH} per Truth Table
		74		0.35	0.5	V	I_{OL} = 24 mA	
I_{OZH}	Output Off Current HIGH				20	µA	V_{CC} = MAX, V_{OUT} = 2.7 V	
I_{OZL}	Output Off Current LOW				−200	µA	V_{CC} = MAX, V_{OUT} = 0.4 V	
I_{IH}	Input HIGH Current	A or B, DR or E			20	µA	V_{CC} = MAX, V_{IN} = 2.7 V	
		DR or E			0.1	mA	V_{CC} = MAX, V_{IN} = 7.0 V	
		A or B			0.1	mA	V_{CC} = MAX, V_{IN} = 5.5 V	
I_{IL}	Input LOW Current				−0.2	mA	V_{CC} = MAX, V_{IN} = 0.4 V	
I_{OS}	Output Short Circuit Current (Note 1)		−40		−225	mA	V_{CC} = MAX	
I_{CC}	Power Supply Current Total, Output HIGH				70	mA	V_{CC} = MAX	
	Total, Output LOW				90			
	Total at HIGH Z				85			

Note 1: Not more than one output should be shorted at a time, nor for more than 1 second.

AC CHARACTERISTICS (T_A = 25°C, V_{CC} = 5.0 V, T_{RISE}/T_{FALL} ≤ 5.0 ns)

Symbol	Parameter	Limits Min	Typ	Max	Unit	Test Conditions
t_{PLH} t_{PHL}	Propagation Delay, Data to Output		8.0 8.0	12 12	ns	C_L = 45 pF, R_L = 667 Ω
t_{PZH}	Output Enable Time to HIGH Level		25	40	ns	
t_{PZL}	Output Enable Time to LOW Level		27	40	ns	
t_{PLZ}	Output Disable Time from LOW Level		15	25	ns	C_L = 5.0 pF, R_L = 667 Ω
t_{PHZ}	Output Disable Time from HIGH Level		15	25	ns	

C.4 Configuración de puerto paralelo.

- Asignación de pines del conector de Puerto paralelo tipo D (25 pines) y del conector Centronic

Pin No (D-Type 25)	Pin No (Centronics)	SPP Signal	Direction in/out	Register	Hardware Inverted
1	1	nStrobe	In/Out	Control	Yes
2	2	Data 0	Out	Data	
3	3	Data 1	Out	Data	
4	4	Data 2	Out	Data	
5	5	Data 3	Out	Data	
6	6	Data 4	Out	Data	
7	7	Data 5	Out	Data	
8	8	Data 6	Out	Data	
9	9	Data 7	Out	Data	
10	10	nAck	In	Status	
11	11	Busy	In	Status	Yes
12	12	Paper-Out / Paper-End	In	Status	
13	13	Select	In	Status	
14	14	nAuto-Linefeed	In/Out	Control	Yes
15	32	nError / nFault	In	Status	
16	31	nInitialize	In/Out	Control	
17	36	nSelect-Printer / nSelect-In	In/Out	Control	Yes
18 - 25	19-30	Ground	Gnd		

Table 1. Pin Assignments of the D-Type 25 pin Parallel Port Connector.

Registros de software del puerto paralelo

- Base 0 – Datos de puerto

Offset	Name	Read/Write	Bit No.	Properties
Base + 0	Data Port	Write (Note-1)	Bit 7	Data 7
			Bit 6	Data 6
			Bit 5	Data 5
			Bit 4	Data 4
			Bit 3	Data 3
			Bit 2	Data 2
			Bit 1	Data 1
			Bit 0	Data 0

- Base+1 – Estado de puerto.

Offset	Name	Read/Write	Bit No.	Properties
Base + 1	Status Port	Read Only	Bit 7	Busy
			Bit 6	Ack
			Bit 5	Paper Out
			Bit 4	Select In
			Bit 3	Error
			Bit 2	IRQ (Not)
			Bit 1	Reserved
			Bit 0	Reserved

- Base+2 – Control de puerto.

Offset	Name	Read/Write	Bit No.	Properties
Base + 2	Control Port	Read/Write	Bit 7	Unused
			Bit 6	Unused
			Bit 5	Enable Bi-Directional Port
			Bit 4	Enable IRQ Via Ack Line
			Bit 3	Select Printer
			Bit 2	Initialize Printer (Reset)
			Bit 1	Auto Linefeed
			Bit 0	Strobe

Apéndice D Programa de monitoreo

```
/*************************************************************
* Nombre Prog: MONITOR.CCP                                   *
* Programador: Oscar Cervantes Martínez                      *
*                                                            *
* Descripción: Este programa además de monitorear el com-    *
*              portamiento de la velocidad de un motor,      *
*              (controlado por el microcontrolador COP8)     *
*              prepara el valor de velocidad (Wm) deseado    *
*              antes de ser enviado al COP8. Cuando el COP8  *
*              regresa el resultado, vuelve a adecuarlo para *
*              que finalmente al reenviarlo al COP8 ,este lo *
*              mande al DAC0832. A fin de monitorear el      *
*              proceso, el COP8 reporta el estado del        *
*              ADC0804, para observar la salida entregada.   *
*                                                            *
*************************************************************/
/******* Archivos Incluidos ***********************************/
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <dos.h>
#include <math.h>
#include <time.h>
#include <ctype.h>

void border(int iniciox, int inicioy, int finx, int finy, int fcolor);
void dato_difuso(double dato, int pot1, int pot2);
void sonido(int beep);
void mensaje(int msj);
void ventanas();
void limpiar();
void continuar_terminar();
/******* Declaración de variables *****************************/

float Wm=0, Wm_minima=0, Wm_maxima=0;
```

```
float Wm_ajustada=0,Wm_acondicionada=0;
int   leer;
int dato1=0, dato2=0, dato3=0, dato4=0, dato5=0;
int pot1, pot2;
double dig1, dig2,datodifuso=0, dato;
double Voltaje_entrada=0, factor_peso,Voltaje_entrada_DAC;
double velocidad_angular_ADC, error_velocidad;
int beep;
char continuar;
int  ciclo,i;
#define puerto 0x378

/******* Programa principal ************************************/
main()
{
// Inicializando registro de puertos.
    outportb(puerto+2,2);

    ventanas();

/***************** Bucle principal ****************************/
for (;;){


    textcolor(15);
    window(3,7,41,20);   // Ventana de datos.
    border(3,7,41,20,0);
    gotoxy(3,3);
    textbackground(1);
    cprintf(" Velocidad Angular [Wm]=       rpm ");
    textcolor(14);
    gotoxy(28,3);
    cscanf("%f", &Wm);
    gotoxy(3,3);
    textcolor(7);
    cprintf(" Velocidad Angular [Wm]= %4.0f rpm ",Wm);
    textcolor(15);
```

```
// Acondicionamiento de Wm.
    Wm_minima=0;
    Wm_maxima=3910;
    Wm_acondicionada=(((Wm-Wm_minima)*256)/(Wm_maxima-Wm_minima))-1;
    textbackground(1);
    for (i=2; i<38; i++){ gotoxy(1,5); cprintf("Í"); }

// Tiempo de ejecución.
    time_t  inicio, fin;
    inicio = time(NULL);

// Envio de Velocidad angular.
    mensaje(2);
    sonido(1000);
    delay(500);
    outportb(puerto,Wm_acondicionada);

    outportb(puerto+2,10); // Aviso de envío de dato acondicionado.
                    // Pin 17=1 , pin 14=1 (en hardware son inversos)
    delay(500);

// Habilitar el puerto bidireccional, espera de respuesta y lectura del dato.
    int datoprueba=0;

    datodifuso = 0;

    outportb(puerto+2,46);    // Habilitar puerto bidireccional y
                        // habilitar pines 14 y 17 del micro.
    delay(500);
/*-------------------------------------------------------------------*/
     mensaje(1);
     do{                                        // Esperar
      datoprueba=inportb(puerto+1);
       }while ((datoprueba & 0xC0) != 0xC0);
      dato1=inportb(puerto);                    // Dato1
      mensaje(4);
```

```
outportb(puerto+2,0x2C);                    // Confirmación
delay(500);
sonido(2000);

do{                                         // Esperar
datoprueba=inportb(puerto+1);
 }while ((datoprueba & 0xC0) != 0x80);
dato2=inportb(puerto);                      // Dato2
mensaje(5);
outportb(puerto+2,0x26);                    // Confirmación
delay(500);
sonido(2000);

do{                                         // Esperar
datoprueba=inportb(puerto+1);
 }while ((datoprueba & 0xC0) != 0x40);
dato3=inportb(puerto);                      // Dato3
mensaje(6);
outportb(puerto+2,0x2E);                    // Confirmación
delay(500);
sonido(2000);

do{                                         // Esperar
datoprueba=inportb(puerto+1);
 }while ((datoprueba & 0xC0) != 0xC0);
dato4=inportb(puerto);                      // Dato4
mensaje(7);
outportb(puerto+2,0x2C);                    // Confirmación
delay(500);
sonido(2000);

do{                                         // Esperar
datoprueba=inportb(puerto+1);
 }while ((datoprueba & 0xC0) != 0x80);
dato5=inportb(puerto);                      // Dato5
mensaje(8);
outportb(puerto+2,0x26);                    // Confirmación
```

```
      delay(500);
      sonido(2000);

     dato_difuso(dato1, 1,0);
     dato_difuso(dato2, 3,2);
     dato_difuso(dato3, 5,4);
     dato_difuso(dato4, 7,6);
     dato_difuso(dato5, 9,8);


/*-------------------------------------------------------------------*/

    window(3,20,79,25);
    gotoxy(2,2);
    for (i=2; i<75; i++){ gotoxy(i,2); cprintf(" "); }
// Reacondicionamiento del dato difuso

    factor_peso = 2048;
    Voltaje_entrada = datodifuso/(128*factor_peso);
    window(3,7,41,20);
    gotoxy(3,7);
    textbackground(1);
    cprintf(" Voltaje de entrada = %2.3f volts ",Voltaje_entrada);

       Voltaje_entrada_DAC=Voltaje_entrada*(255/9.991);
       if(fmod(Voltaje_entrada_DAC,1) < 0.5 ) Voltaje_entrada_DAC =
floor(Voltaje_entrada_DAC);
       if(fmod(Voltaje_entrada_DAC,1) >= 0.5 ) Voltaje_entrada_DAC =
ceil(Voltaje_entrada_DAC);

// Envio de Voltaje de entrada.
     mensaje(3);
     outportb(puerto+2,0x08); // Aviso de envio de vvoltaje de entrada.
                       // Pin 17=1 , pin 14=1 (en hardware son inversos)
     outportb(puerto,Voltaje_entrada_DAC);
     delay(500);
     sonido(2000);
```

```
// Monitor de la respuesta.

    int cont=0;
    double err, errWm, valornuevo=0;

    outportb(puerto+2,0x2C);            // Habilitar puerto.

    mensaje(1);
    for (;;){

     do{                                          // Esperar
      datoprueba=inportb(puerto+1);
       }while ((datoprueba & 0xC0) != 0xC0);
      valornuevo=inportb(puerto);                 // DATO
      mensaje(9);
      delay(500);
      sonido(3000);

      valornuevo = ((inportb(puerto)*10.5)/255)/0.003;;    // Leer dato
      errWm = ((Wm-valornuevo)/Wm)*100;

      window(44,7,79,20);
      textbackground(1);
      textcolor(14);
      gotoxy(2,3); cprintf(" Velocidad Angular = %0.2f rpm ",valornuevo);
      gotoxy(2,5); textbackground(1);
            for (i=2; i<35; i++){ gotoxy(i,5); cprintf("Í"); }
      gotoxy(2,7); cprintf(" Error porcentual  = %0.3f % ",errWm);
      gotoxy(2,9); textbackground(1);
            for (i=2; i<35; i++){ gotoxy(i,9); cprintf("Í"); }

      cont++;
      delay(1000);

      if (cont > 10 )break;
    }
```

```
    outportb(puerto+2,0x02); // Proceso completo.
                        // Pin 17=0 , pin 14=1 (en hardware son inversos)
    window(44,7,79,20);

// Tiempo de proceso.
    fin = time(NULL);
    gotoxy(2,11);
    textbackground(1);
    cprintf(" Tiempo utilizado = %2.3lf seg ",difftime(fin,inicio)-10);

// Continuar o terminar.
    continuar_terminar();
//Inicializar micro.
 //   outport(puerto+2,0);

  if (ciclo != 1)break;
  }

/****************** Fin del bucle principal *********************/

}   /* Fin de main */

/************************ Sub-rutinas ****************************/

// Composición del dato difuso.
void dato_difuso(double dato, int pot1, int pot2)
{
dig1=floor(dato/16);
dig2=fmod(dato,16);
datodifuso=datodifuso+(dig1*pow(16,pot1)+dig2*pow(16,pot2));
}

// Sonido de transferencia (bidireccional).
void sonido(int beep)
{
    sound(beep);
    delay(50);
```

```
   nosound();
}

// Mensajes
void mensaje(int msj)
{
 switch (msj) {
  case  1:
      window(3,20,79,25);
      gotoxy(2,2);
      printf("Esperando respuesta...              ");
      break;
  case  2:
      window(3,20,79,25);
      gotoxy(2,2);
      printf("Enviando Velocidad Angular...       ");
      break;
  case  3:
      window(3,20,79,25);
      gotoxy(2,2);
      printf("Enviando Voltaje de entrada...      ");
      break;
  case  4:
      window(3,20,79,25);
      gotoxy(2,2);
      printf("Recibiendo dato UNO...              ");
      break;
  case  5:
      window(3,20,79,25);
      gotoxy(2,2);
      printf("Recibiendo dato DOS...              ");
      break;
  case  6:
      window(3,20,79,25);
      gotoxy(2,2);
      printf("Recibiendo dato TRES...             ");
      break;
```

```
 case  7:
      window(3,20,79,25);
      gotoxy(2,2);
      printf("Recibiendo dato CUATRO...              ");
      break;
 case  8:
      window(3,20,79,25);
      gotoxy(2,2);
      printf("Recibiendo dato CINCO...               ");
      break;
 case  9:
      window(3,20,79,25);
      gotoxy(2,2);
      printf("Esperando estabilidad...               ");
      break;
      }
}

void ventanas()
{
// Ventana Inicial.
     clrscr();
     textcolor(15);
     textbackground(9);
     border(1,1,81,26,9);
     gotoxy(2,6);
     printf(" DATOS ");
     gotoxy(43,6);
     printf(" RESPUESTA ");
     gotoxy(2,19);
     printf(" MENSAJES ");

 // Ventana de Titulo.
      window(13,2,69,6);
      border(13,2,69,6,0);
      gotoxy(5,2);
      textbackground(1);
```

```
    cprintf("        CONTROL DIFUSO DE UN MOTOR DE CD          ");
    window(44,7,79,20);
    border(44,7,79,20,0);
    window(3,20,79,25);
    border(3,20,79,25,0);
}

//Limpiar ventana.
void limpiar()
{
    window(3,20,79,23);
    clrscr();
    textbackground(1);
    textcolor(15);
    border(3,20,79,25,0);
    window(44,7,79,18);
    gotoxy(2,3); cprintf("                                    ");
    gotoxy(2,7); cprintf("                                    ");
    gotoxy(2,11); cprintf("                                   ");
}

// Continuar o terminar el programa.
void continuar_terminar()
{
     window(3,20,79,25);
    gotoxy(2,2);
    printf(" Presione [C]ontinuar o [T]erminar. ");

    do{
     continuar = getch();
       if(tolower(continuar) == 'c'){
              ciclo = 1;
              limpiar();
              }
       else if (tolower(continuar) == 't'){
              ciclo = 2;
              }
```

```
      else
              ciclo = 0;
    }while (ciclo == 0 );
}

// Mostrar borde.
void border(int iniciox, int inicioy, int finx,int finy, int fcolor)
{
    textbackground(fcolor);
    finy=finy-inicioy;
    finx=finx-iniciox;

    int y, x;
    for(y=0; y<=finy-1; y++){
      for(x=0; x<=finx-2; x++){
       gotoxy(x,y);
       putch(' ');
      }
    }
    iniciox=1;
    inicioy=1;
    register i;
    gotoxy(1,1);
    for(i=0; i<=finx-iniciox; i++)
    putch('-');

    gotoxy(1,finy-inicioy);
    for(i=0; i<=finx-iniciox; i++)
    putch('-');

    for(i=2; i<finy-inicioy; i++){
    gotoxy(1,i);
     putch('|');
    gotoxy(finx-iniciox+1,i);
     putch('|');
    }
    gotoxy(iniciox,inicioy);
```

```
    putch('+');
   gotoxy(finx,inicioy);
    putch('+');
   gotoxy(iniciox,finy-1);
    putch('+');
   gotoxy(finx,finy-1);
    putch('+');
}
```

BIBLIOGRAFIA

1) Circuitos Integrados Lineales, Theodore F., Limusa, 1988.
2) COP(SA Family, National Semiconductors, 1999.
3) COP8 Developer Tools, National Semiconductors, 1998.
4) COP8 NeuFuz User's Manual, National Semiconductors, 1993.
5) COP8 Part Numering Methodology, National Semiconductors, 1998.
6) COP8 Preliminar, National Semiconductors, 1997.
7) COP8 User's Manual, National Semiconducters, 1996.
8) COP8SAx Designer's Guide, National Semiconductors, 1997.
9) Diccionario de la Informática, P. Guirao, Prisma.
10) Fuzzy computing, Gupta/Yamakawa, North-Holland, 1988.
11) Fuzzy Logic and Control, Jamshidi/Vadiee/Ross, Prentice Hall, 1993.
12) Fuzzy Logic, D. Mc. Nelly/P. Freinberg, Simon/ Schuster, 1973.
13) Ingeniería de Control Moderna, K. Ogata, Prentice Hall, 1980.
14) Introducción a la teoria de Conjuntos Borrosos, A. Kaufmann, 1982.
15) Microcontroladores PIC, José Ma. Angulo U./Ignacio Angulo M., McGraw Hill,1999.
16) Microcontroller, K. Hintz/D. Tabak, McGraw Hill, 1992.
17) Microcontrollers, Kenneth Hintz, McGraw Hill, 1992.
18) Microprocesadores, García Guerra, Limusa, 1988.
19) Neural Network and Fuzzy Systems, Bart Kosko, Prentice Hall.
20) Notas de Aplicación (varias), National Semiconductors, 1995.