FUZZY DYNAMIC UNIT

JOSÉ RICARDO ZAPATA GONZÁLEZ

UNIVERSIDAD PONTIFICIA BOLIVARIANA
FACULTAD DE INGENIERIA ELECTRONICA
MEDELLIN
2003

FUZZY DYNAMIC UNIT

JOSÉ RICARDO ZAPATA GONZÁLEZ

Trabajo de grado para optar por el titulo de Ingeniero Electrónico

Director

TONY PEÑARREDONDA CARAVALLO

Ingeniero Electrónico especialista en control y automatica

UNIVERSIDAD PONTIFICIA BOLIVARIANA

FACULTAD DE INGENIERIA ELECTRONICA

MEDELLIN

2003

Nota de aceptación

_____

_____

_____

_____

_____

Presidente del jurado

_____

Jurado

_____

Jurado

A mi Madre, por su tenacidad

A Mary, por el amor a la familia

A Caro por estar siempre conmigo

A Daniel por su energía

A Isabel C. por su creatividad

A July por su ternura

Y especialmente a mi Padre

Por su apoyo, constante dedicación y

Por ser el mejor modelo a seguir

AGRADECIMIENTOS

En el desarrollo de este trabajo intervinieron muchas personas sin las cuales no hubiese sido posible terminarlo.

Para comenzar, quiero agradecer a mi familia por su compañía y apoyo incondicional en mi formación como persona.

Al director del proyecto, el ingeniero Tony Peñaredonda, por su asesoría incondicional, tiempo de dedicación e interés en el proyecto y por ser un gran apoyo como persona e ingeniero.

A la Ingeniera Cristina Gómez S, por su compañía y gran ayuda en la elaboración del trabajo escrito, fue mi pilar en el desarrollo de este trabajo pues siempre me dio ánimos para continuar.

A los ingenieros electrónicos Jorge Mario Jiménez, David Fernando Álvarez, Juan Carlos Cardona y Julián Andres Aranzalez, por su ayuda y amistad en estos años de estudio.

Al Ingeniero Roberto Carlos Hincapié por su ayuda, como maestro y como persona.

A la Universidad Pontificia Bolivariana y al CIDI por facilitar los medios para presentar la tesis en la 115 convencion anual de la sociedad de ingenieria de sonido (AES), en new york.

A mis amigos y amigas que siempre me alegran el día.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA ANEXOS

GLOSARIO

FADER: potenciómetro de deslizamiento lineal, utilizado principalmente en las consolas de audio.

FEEDBACK: configuración de control con realimentación de las variables de la salida del sistema.

FEEDFORDWARD: configuración de control donde solo se tienen en cuenta las variables a la entrada del sistema.

RANGO DINÁMICO: es la diferencia en Decibeles entre la intensidad más fuerte y la más débil de una señal (acústica, eléctrica, etc).

RATIO: es la relación en decibeles entre las señales de audio de entrada y salida de una unidad dinámica.

SISTEMA EMBEBIDO: sistema diseñado para una o varias tareas especificas.

THRESHOLD: nivel del audio donde se activa la unidad dinámica, para cambiar los niveles de la señal.

UNIDAD DINÁMICA: sistema de audio que permite modificar el rango dinámico de una señal, partiendo de determinados parámetros fijados.

RESUMEN

En este trabajo de grado se presenta el diseño y desarrollo de un Sistema de Control Difuso para señales de audio en tiempo real, el cual permite ejercer un control inteligente sobre el rango dinámico de dichas señales.

Se trata de una aplicación novedosa de los conceptos de control inteligente en la solución de un problema de procesamiento de audio. Específicamente, se ilustrará el proceso de diseño de un controlador difuso para que desarrolle las funciones de una unidad dinámica de audio. Se pretende de esta manera mejorar el desempeño y nivel de distorsión de la señal de audio involucrada en el procesamiento.

INTRODUCCIÓN

El presente trabajo de grado, muestra el proceso de diseño y desarrollo de un Sistema de Control Difuso para señales de audio en tiempo real.
Hasta ahora el control inteligente (*Fuzzy Logic*) no había sido aplicado en el procesamiento del rango dinámico del audio a nivel de sistemas embebidos.

Como antecedentes a este proyecto, la Universidad Pontificia Bolivariana ha desarrollado estudios en ésta área como el proyecto de grado del profesor Carlos Peñuela titulado: " Lógica difusa y controladores Difusos", que consiste en la sintonía de un control PID (proporcional-integral derivativo)para el CODILUM(controlador lineal universal multivariable) con base en algoritmos difusos. En la Universidad Nacional se han presentado 2 proyectos de grado con base en algoritmos que se han usado en programas de simulación en lógica difusa, pero no trabajan en tiempo real.
También se tienen trabajos de grado con lógica difusa en tiempo real, este trabajo de grado se realizo específicamente para el CODILUM, titulado: "Control En Tiempo Real Con Lógica Difusa Para El CODILUM" el trabajo consistió en el diseño y elaboración de un software de control encargado de recibir los datos que ofrece el hardware del CODILUM (controlador lineal universal multivariable) y generar una acción de control con base en una matriz de interferencia que puede diseñarse para cualquier control deseado con la ayuda de un segundo software para diseño con lógica difusa.

El grupo de investigación A+D (Automática y Diseño, antes GRIAL), al cual pertenece el director de este trabajo, tiene dentro de sus líneas de investigación, el control inteligente. En este grupo se ha venido desarrollando un trabajo continuo en el tema de los controladores difusos, especialmente el

FMRLC (Fuzzy modeling reference learning control) . Los ingenieros Manuel Betancourt, Tony Peñarredonda y César Quintero han propuesto modificaciones al esquema original de este controlador con éxito notorio y difusión internacional. Este trabajo de grado aprovechó este bagaje investigativo para una aplicación novedosa en el área del procesamiento de audio.

De esta manera, a continuación se presenta un informe detallado del diseño e implementación tanto en hardware como en software de un prototipo que ejerce un control inteligente por medio de lógica difusa para que ejerza las funciones de una unidad dinámica de audio sobre un fader motorizado.

Con esto se pretende adicionalmente dar a conocer el funcionamiento de una unidad dinámica de audio convencional y demostrar cómo los controladores basados en lógica difusa pueden ser útiles en aplicaciones de procesamiento de audio.
Estos procesadores de dinámica últimamente han trabajado con sistemas electrónicos que por su naturaleza le introducen ruido a la señal. En este trabajo de grado, se propone un sistema cuya unidad dinámica se hace sobre un fader motorizado, que por sus de características eléctricas evita de manera considerable la distorsión armónica inherente de otros sistemas de procesamiento de audio en tiempo real, debido a sus componentes.

Un trabajo de esta naturaleza sería de gran interés para la comunidad de ingenieros de audio y los ingenieros de control. Se pretende realizar una aplicación de los controladores inteligentes en un terreno muy poco explorado por esta materia, la cual es el procesamiento de audio.ademas de un gran interés académico, pues sin importar cuales sean estos positivos o negativos, dejaran una experiencia acumulada en la ingeniería la cual podría servir de futura referencia en posteriores trabajos de esta naturaleza. Puesto que en la universidad solo se han desarrollado simulaciones o software acerca de los controladores difusos.

1. PROCESADORES DE DINÁMICA

Al realizar grabaciones de audio o mezclas en vivo, existe frecuentemente la necesidad de controlar el volumen de la señal de forma automática.

De esta manera se busca evitar que el volumen de salida sea excesivo, ya sea para no saturar los sistemas de audio (como amplificadores, Pre-amplificadores, etc), eliminar efectos desagradables como la presencia de distorsiones en la señal, o simplemente evitar dejar sorda a la concurrencia que esta en un concierto. Quizás lo que se pretende tener bajo control es la voz de un cantante que puede variar (desde un grito hasta un susurro) al acercarse o alejarse del micrófono. En otros casos puede que solo se quiera excluir ruidos externos que se colan cuando la señal no está presente.

Un claro ejemplo de dichas variaciones en el sonido se puede ver fácilmente en una pieza musical con partes tan suaves (*ppp pianisissimo*) en la que se necesita mucha atención para poder oírlas, y partes tan fuertes (*fff fortisissimo*) que podrían parecer estruendosas en la presentación.

Para llevar a cabo este tipo de controles sobre la señal, se utilizan los llamados procesadores de dinámica , que normalmente son usados como ayuda en presentaciones en vivo y aplicaciones de grabación multi-pista. Es poco común encontrarlos en reproducción, debido a que en la grabación ya se hizo el control del rango dinámico del sonido.

1.1. RANGO DINÁMICO

Es la diferencia en Decibeles entre la intensidad más fuerte y la más débil de una señal (acústica, eléctrica, etc).

El rango útil de los sistemas de grabación y reproducción no supera en promedio los 80 db (eléctricos análogos 80 db, magnéticos análogos 60 db, digitales 120 db). Por su parte, el rango dinámico generado por la voz o por instrumentos acústicos es alrededor de 130 db, sobrepasando el rango útil de los sistemas en mención. Como se observa el rango dinámico de los sistemas de grabación y reproducción digitales es el que más se acerca al rango útil de la música (dependiendo de su estilo o género), de esta manera, si la voz humana y los instrumentos (guitarras españolas, acústicas y eléctricas, bajo eléctrico, bombo, tambor, etc) que hacen parte de la pieza musical corresponden a estilos de música "acústica" (folk, clásica, jazz, etc) se trata de captar el mayor rango dinámico posible, mientras tanto en otros estilos como el rock, rap, hip hop, etc, se necesitan rangos dinámicos más reducidos (y en algunos casos casi nulos) ya que su atención se centra principalmente en el "impacto" sonoro y no tanto en los "matices" de la interpretación.

1.2. PRINCIPIO DE FUNCIONAMIENTO

Los procesadores dinámicos actúan modificando el rango dinámico de la señal de audio. Un procesador de Dinámica no es diferente al concepto de una persona que controla el nivel de volumen en un canal de mezcla, por ejemplo, si un cantante se encuentra grabando, entonces el sonido es recogido por un micrófono, cuando el cantante comience a cantar más fuerte (o se acerque al micrófono), el operador disminuirá el volumen del canal (para mantener un nivel adecuado). Es decir, trabajará como un compresor de audio. Cuando deje de cantar el operador bajará todo el volumen para evitar que se entre el

ruido que hay en el ambiente, trabajando como una compuerta de ruido (*Noise Gate*).

Cuando se está cantando, el operario no aumentará del todo el volumen del canal, por que podría aparecer un pico en la señal haciendo que dañe los sistemas de grabación, o los altavoces. Se encontraría entonces trabajando como un limitador, que es una forma específica de compresor.

En el proceso hay dos elementos, una señal cuyo volumen se varía, y alguien que está escuchando y encargada de tomar decisiones para cambiar el volumen dependiendo de los criterios, para decidir si hay o no un cambio relevante.

El gráfico a continuación detalla el proceso explicado anteriormente: el sistema auditivo detecta el volumen, el cerebro según los criterios que tenga al escuchar el sonido, ordenará a la mano si debe bajar, subir el volumen o no actuar, dependiendo de lo que sucede con la señal en el tiempo.

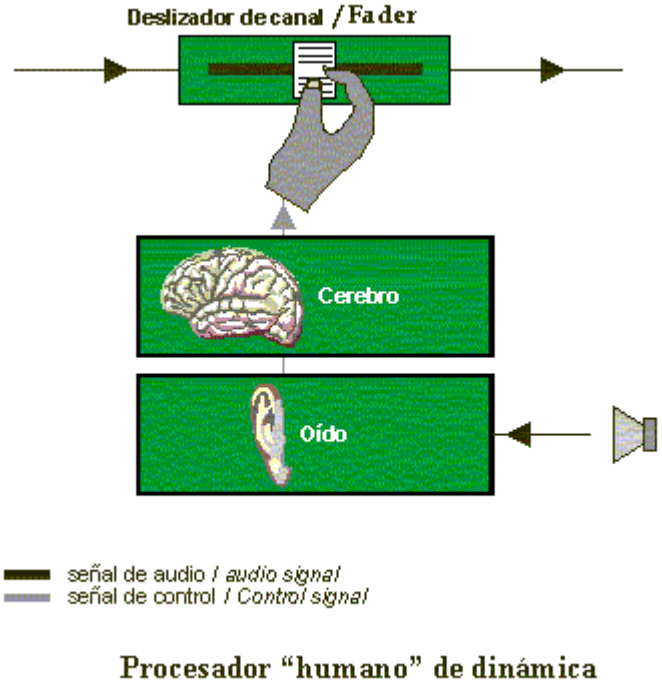


Figura 1. Procesador "humano" de dinámica. Fuente: www.doctorproaudio.com

Este procesador humano de dinámica tiene algunas limitaciones, para comenzar sólo puede controlar un canal, es lento y sus acciones no pueden ser exactamente repetibles, además el operador debe conocer la pieza musical

para poder anticipar en que momento subir la intensidad del sonido y en que momento bajarla, haciendo de este un proceso poco óptimo al requerir largas horas de grabación que a su vez elevan el costo. Este procedimiento era anteriormente usado.

Actualmente se recurre a un sistema electrónico que realice dichas funciones.

El sistema electrónico de procesamiento de dinámica tiene el mismo principio de funcionamiento al descrito anteriormente, y supera las limitaciones que presentaba el procesador humano de dinámica. El funcionamiento es el siguiente: la señal de entrada se divide en dos, una de estas dos copias será procesada a través de un sistema de ganancia controlada que normalmente es un ACV (amplificador controlado por voltaje o *VCA - voltaje controller amplifier*). La otra copia de la señal de entrada va a un circuito de detección que actúa sobre el sistema de ganancia ACV. Para que los cambios de volumen sean graduales (y no se escuchen variaciones bruscas durante los cambios del ACV), se utiliza un generador de envolvente el cual establece una rampa cuya forma permite una variación suave entre los cambios de nivel. Normalmente se puede decidir entre detectar la señal que se va a procesar o detectar una señal externa, en este último caso se habla de señal *side – Chain* (cadena lateral) o *key*.

En la siguiente figura se encuentra un diagrama de bloques que describe el funcionamiento del procesador de dinámica electrónico, en el cual se puede observar la división de la señal de entrada.

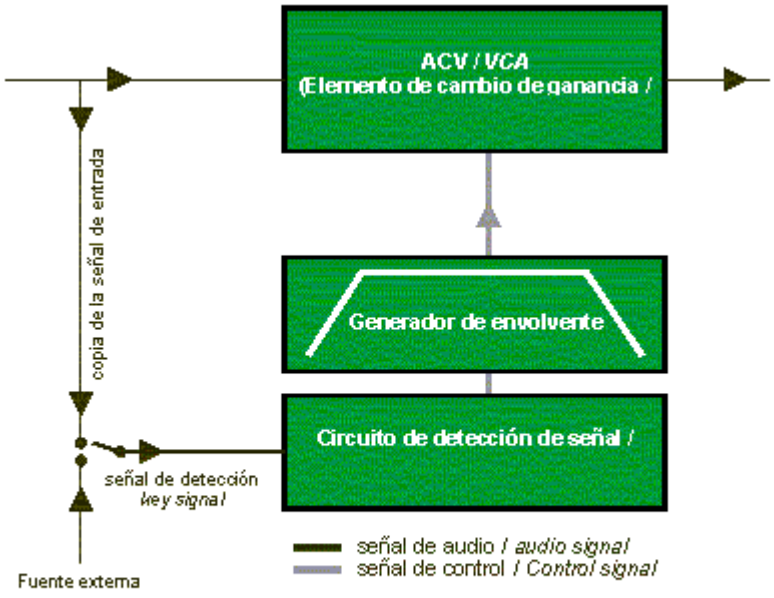


Figura 2. Procesador electrónico de dinámica. Fuente: www.doctorproaudio.com

Uno de los efectos secundarios de utilizar ACV (amplificador controlado por voltaje) en la señal de audio es la introducción de ruido, ya que los circuitos tienen un ruido inherente y presentan distorsión armónica total (distorsión no lineal y distorsión armónica), los mejores sistemas de ACV (que introducen menos ruido) resultan muy costosos, por lo que se encuentran en equipos de grabación de muy alta calidad.

Independientemente del circuito ACV, en los procesadores dinámicos debe existir un buen circuito de detección, el cual es complejo en su diseño y construcción.

El circuito de detección es el encargado de detectar cuando la señal ha pasado el nivel permitido, o el nivel donde empieza a actuar la unidad dinámica.
Las unidades digitales, no tienen el problema de los ACV, pero obtener buenos algoritmos para el procesado de dinámica es difícil ya que son complejos de realizar. Resultando también tediosa una implementación de una unidad dinámica digital de muy buena calidad.

Un buen procesador de dinámica debe realizar su función sin introducir ruidos, o cambios abruptos en el nivel del volumen de la señal, haciéndola poco natural.

1.3. DIAGRAMA DE BLOQUES GENERAL

Como se mencionó anteriormente, el funcionamiento de una unidad dinámica requiere básicamente 2 elementos: un circuito de detección, y un ACV (Amplificador controlado por voltaje), quedando el diagrama de bloques general, para las unidades de procesamiento dinámico de esta manera:

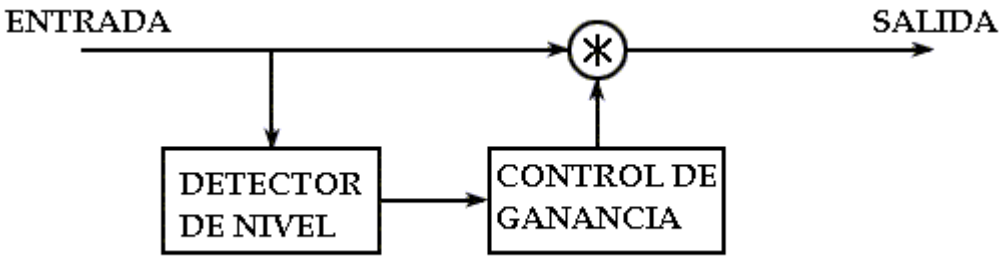


Figura 3. Diagrama de bloques unidad dinámica.

La mayoría de las unidades dinámicas tienen esta configuración, ya sea los compresores, los limitadores, expansores, compuertas de ruido y otros.

El detector de nivel y el controlador de ganancia deben ser lo suficientemente rápidos para actuar, ya que en el caso de un limitador, si se detecta un pico de voltaje y no es cortado rápidamente, entonces puede pasar a la grabación o a el sistema de amplificación produciendo ruidos, saturando los sistemas, o generando armónicos y/o distorsiones.

Como se puede observar en el diagrama de bloques, las unidades dinámicas no tienen realimentación, es decir, tienen una configuración *feedforward*, lo que puede llevar a problemas de control (como la introducción de distorsiones en

la señal o cambios en los sonidos captados) si la unidad no está bien calibrada.

1.4. TIPOS

En general las unidades de procesamiento dinámico se encuentran tanto en *Hardware* como en *Software*, las más comunes son:

1.4.1. Compresor / Limitador. Atenúa o limita el nivel de la señal desde determinado punto en la amplitud reduciendo el rango dinámico de la misma. El limitador es un caso específico del compresor.

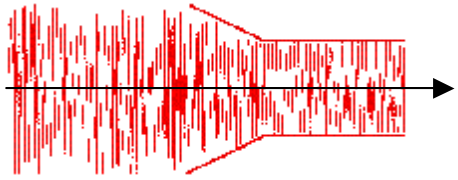


Figura 4. Compresión de una señal. Fuente: www.cybercollege.com/span/

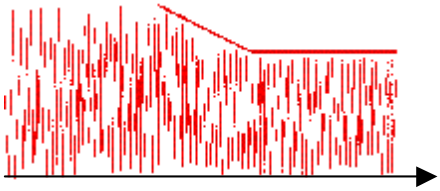


Figura 5. Limitación de una señal. Fuente: www.cybercollege.com/span/

1.4.2. Compuerta De Ruido (*Noise Gate*). enmudece las señales que se encuentran por debajo de determinado punto en el nivel de la señal. se utiliza para eliminar el ruido de fondo. Si se permite regular la atenuación, entonces se habla de "expansor hacia abajo" o *downward expander*.

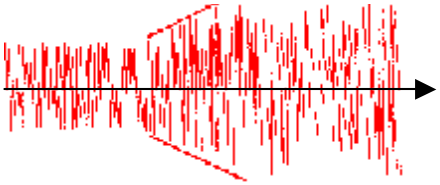


Figura 6. Expansión de una señal. Fuente: www.cybercollege.com/span/

1.5. COMPRESOR / LIMITADOR

El compresor es la unidad de procesamiento dinámico más utilizada. Está encargado de reducir el rango dinámico de la señal. Es de gran aplicación tanto en grabación, como en presentaciones en vivo. Normalmente los micrófonos vocales deben pasar a través del compresor, esto con el fin de tener la voz del cantante en un nivel óptimo, de tal manera que los sonidos débiles no se vean tan opacados por los sonidos fuertes ni por el ruido externo, y que los sonidos fuertes no terminen saturando los sistemas de audio o dañando la fidelidad de la señal.

Los compresores han adquirido gran importancia en algunos casos específicos como los mencionados a continuación:

- Cuando se tienen voces o instrumentos con un gran rango dinámico (guitarras, bajos, bombos, etc), ya que se deben minimizar las variaciones de nivel ocasionadas por cambios en la distancia entre la fuente sonora y el micrófono, o por aumento abrupto de la intensidad de la fuente sonora.
- Cuando se requiere grabar sobre un sistema que no soporta tanto rango dinámico como la fuente original
- Como protección ante posibles saturaciones
- Para suavizar los ataques de fuentes sonoras intensas
- Cuando se pretende conseguir una sensación de alta intensidad sin llegar a saturar o distorsionar la grabación

El compresor deja pasar la señal sin hacerle ningún cambio desde que no supere el nivel de intensidad definida, el punto en el cual el compresor comienza a trabajar se llama umbral (*threshold*), y a partir de este punto, el compresor comienza a actuar como un dispositivo de ganancia variable que depende directamente de la señal de entrada, de tal forma que si en la entrada del compresor hay una señal de intensidad que pasa el umbral (

threshold), este ajustará su ganancia inferior a uno, y si es una intensidad baja se ajusta a un valor más alto (pero no superior a uno), reduciendo de esta manera el rango dinámico de la señal.

El limitador es una forma específica de compresor, la única diferencia es que aplana la señal a partir del punto de *threshold* o umbral, todo nivel de intensidad que este por encima del punto de *threshold* o umbral.
Se podría decir que comprimir es atenuar de manera suave, pero limitar es hacerlo de forma brusca, esta es útil cuando se quieren evitar picos de voltaje que puedan dañar los equipos de amplificación, y por lo tanto se considera un sistema de protección .

1.5.1. Característica entrada / salida.

En la gráfica 7 de la característica de la entrada / salida del compresor se tienen varias partes importantes.

La zona 1 es la parte de la característica en la cual el compresor no trabaja, la señal sale con la misma intensidad (dB) como entró al compresor, o sea que hay una relación 1:1.

En la zona 2 se encuentra el punto de *threshold* o umbral, este es el punto donde el compresor empieza a trabajar, a partir de este nivel de intensidad la señal se empieza a comprimir.

En la zona 3 es donde se encuentra la señal por encima del punto de *threshold* o umbral, entonces la señal se comprime con cierta proporción, de pendiendo del requerimiento establecido, en la gráfica se puede observar una relación de compresión de 2:1, 4:1 y 10:1. Como se muestra en la ilustración 7, en esta zona la característica de entrada / salida cambia su pendiente; lo que quiere

decir la relación de compresión 2:1 es que por cada dos dB de entrada hay una salida de un dB, y de forma análoga para las otras relaciones de compresión.

Cuando la compresión supera o es igual a 10:1 se dice que esta trabajando como un limitador.
Aunque teóricamente se considera que la relación de limitación es infinito a uno (∝:1)

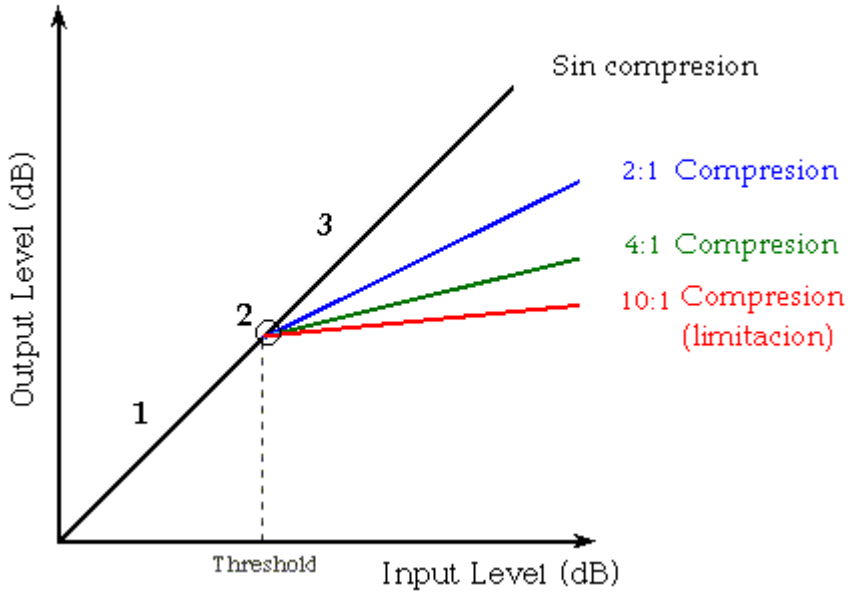


Figura 7. Característica entrada / salida de un compresor. Fuente: www.harmony-central.com

1.5.2. Respuesta del compresor / limitador

A continuación se ilustra la manera como se comporta el compresor / limitador en el tiempo cuando se le aplica una señal.

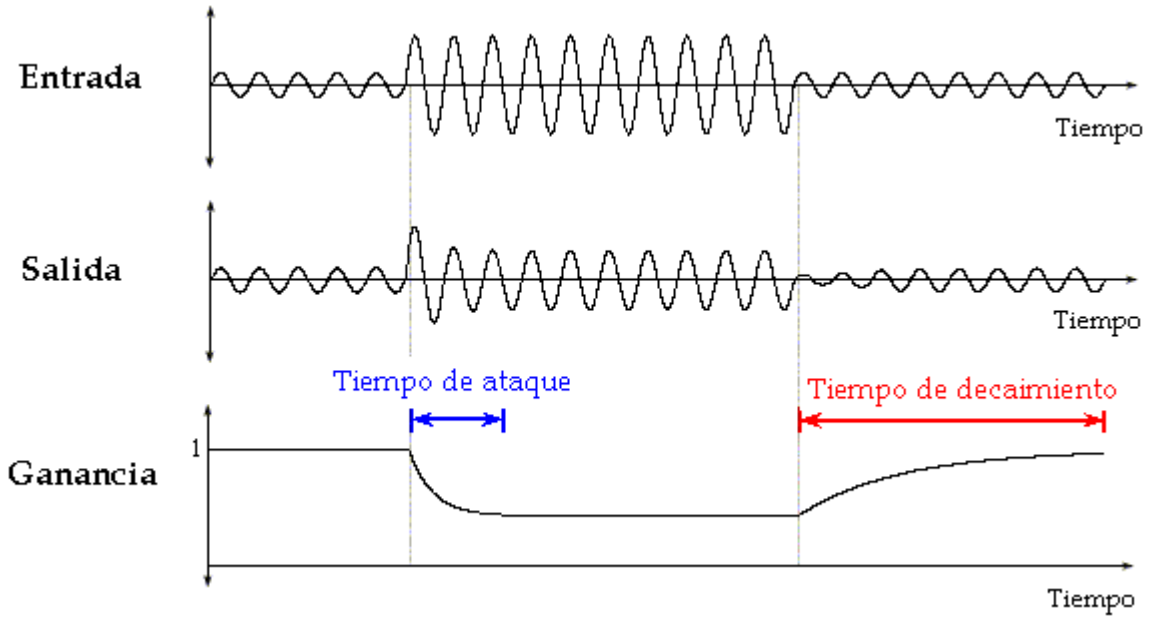


Figura 8. Respuesta del compresor / limitador en el tiempo. Fuente: www.harmony-central.com

Se puede apreciar que cuando la señal de entrada supera el punto de *threshold* o umbral, el compresor no hace el cambio de nivel de manera abrupta, sino que va cambiando de manera gradual la ganancia hasta que llega a la ganancia determinada, el tiempo que transcurre en este procedimiento es llamado tiempo de ataque, que es variable. Igualmente, cuando la intensidad de la señal en la entrada baja el compresor vuelve al estado en el cual la ganancia es igual a uno, pero el cambio tampoco es abrupto, y el tiempo que transcurre en este procedimiento es llamado tiempo de decaimiento o relajación.

1.5.3. Controles.

La compresión tiene varias variables a controlar, dependiendo del tipo de señal y de los requerimientos que se tengan estos se pueden modificar según sea necesario, podemos ver en la grafica los controles que tiene un compresor / limitador común.

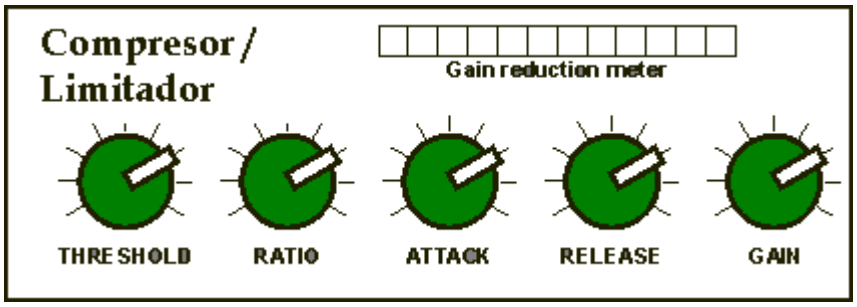


Figura 9. Controles del compresor / limitador. Fuente: www.doctorproaudio.com

1.5.3.1. Umbral (*Threshold*).

Es el parámetro fundamental en una unidad dinámica, establece en que punto comienza a actuar el compresor / limitador, lo cual significa que debajo de él, el compresor / limitador no actúa y deja pasar la señal tal como entro la unidad dinámica. Este parámetro es ajustable por el usuario.

A continuación se muestra un ejemplo de cómo varía el nivel (en dBs) de una señal al comprimirse con un umbral más alto o más bajo, se puede ver que en el primer ejemplo, el tercer pico pasa sin ninguna modificación, lo que no ocurre en el segundo ejemplo.

La señal gris es la original, y la señal verde es la procesada.

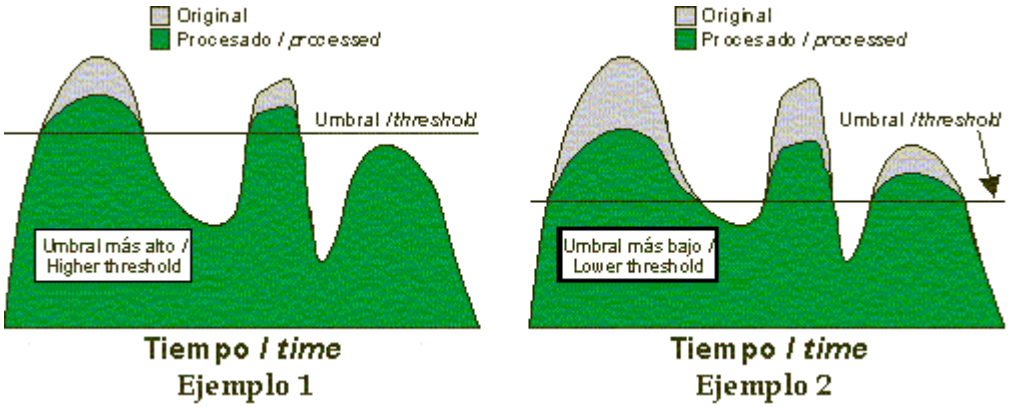


Figura 10. Umbral o threshold. Fuente: www.doctorproaudio.com

1.5.3.2. Tiempo de ataque (*attack time*).

Es el tiempo que transcurre desde que se presenta una señal de intensidad alta que pasa el umbral o *threshold*, hasta que el compresor / limitador ajusta

la ganancia a un valor determinado. Este cambio no puede ser abrupto ya que generaría distorsión en la señal, por que cambiaria su envolvente dando una sensación distinta del sonido. Con este podemos controlar que "tan rápido" debe hacerse la compresión.

Los tiempos mínimos de ataque pueden oscilar entre 50 y 500 μSeg (microsegundos), mientras que los tiempos máximos oscilan entre 20 y 100 mSeg (milisegundos). Podemos ver el tiempo de ataque en la figura 8.

1.5.3.3. Tiempo de decaimiento o relajación (*release time*).

Es el tiempo que pasa desde que hay una señal de baja intensidad, que se encuentra por debajo del umbral o *threshold* , hasta que ajusta su ganancia a uno, cuando la ganancia está por debajo de uno, en otras palabras es el tiempo que transcurre desde que el compresor pasa del estado de compresión (atenuación, limitación, etc), hasta que llega al estado donde deja pasar las señales sin afectarlas.

El tiempo de decaimiento es mayor al tiempo de ataque. y suelen oscilar entre los 40-60 ms y los 2-5 segundos. El tiempo de relajación no debe ser muy largo por que puede llegar a producir un efecto de "bombeo" (*pumping*), causado por los ciclos de activación y desactivación de la compresión. Si es muy rápido también puede cambiar la forma de la envolvente de la fuente sonora. Se puede encontrar el tiempo de decaimiento o relajación en la figura 8.

1.5.3.4. Relación de compresión (*ratio*).

Este parámetro indica cuanto debe ser comprimida la señal, después de pasar el umbral (*threshold*), la relación de compresión o *ratio* es la pendiente de la curva de característica de entrada / salida del compresor, en valores mayores al umbral (*threshold*) esto se muestra en la siguiente grafica.

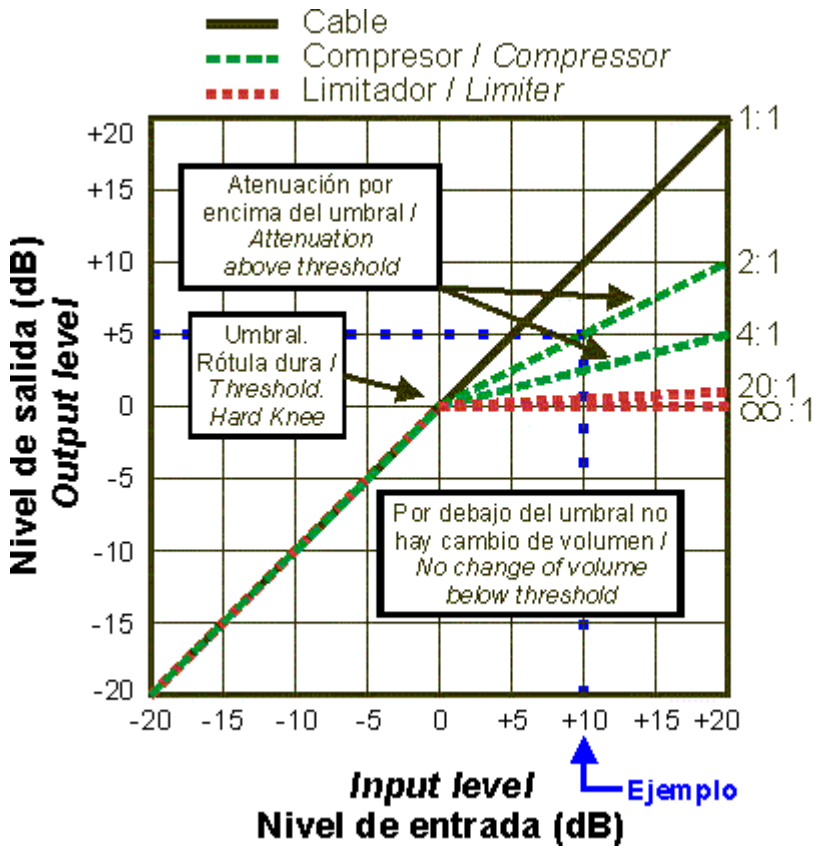


Figura 11. Relacion de compresion (*ratio*). Fuente: www.doctorproaudio.com

La relación de compresión normalmente oscila entre 1:1 (se lee uno a uno, esto es cuando no hay modificación de la señal, como si fuera un cable) y 40:1 (cuarenta a uno), cabe recordar que cuando la compresión es mayor de 10:1 el compresor comienza a funcionar como un limitador.

La relación es en dBs, aunque es adimensional. Por ejemplo, si se tiene una compresión de 4:1 lo que quiere decir es que por cada 4 dBs en la entrada, en la salida se tiene un dB. O como se puede ver en el ejemplo de la figura 11, las líneas que están en azul, indican que hay una compresión de 2:1, es decir que por cada dos dBs en la entrada se obtendrá un dB en la salida, y en el ejemplo de la entrada de 10 dBs se obtendrá una salida de 5 dBs.

El siguiente gráfico muestra los niveles de una señal comprimida y sin comprimir para diversas relaciones de compresión, desde poca compresión hasta compresión máxima (limitación). Las relaciones son 3:1 (ejemplo 1) ,

1.5:1 (ejemplo 2) e infinito:1 (limitación, ejemplo 3) ,nótese que se tarda un tiempo en llegar al nivel de umbral, esto debido al tiempo de ataque.
La señal gris es la original, y la señal verde es la procesada.

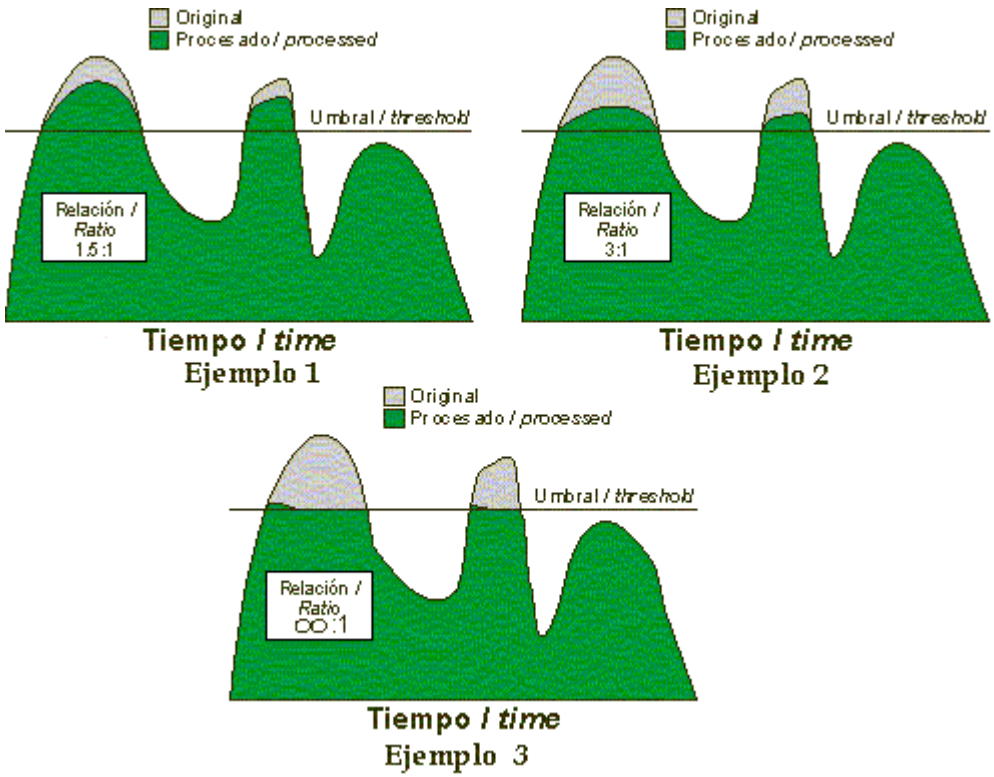


Figura 12. Ejemplos de relación de compresión, 3:1 , 1.5:1 y infinito:1. Fuente: www.doctorproaudio.com

Codo o rótula (*knee*):

Regula la transición entre el estado de no-procesamiento y procesamiento en un compresor, puede existir la opción entre una transición suave (*soft knee*, codo suave) o una transición dura (*hard knee ,* codo duro), en algunos compresores existe un control que permite variar entre la transición suave y la transición dura. Un caso especial es la transición *Overeasy*, llamada así debido a la empresa DBX, pero en realidad es equivalente a la transición suave.

La transición suave se hace muy necesaria cuando la señal de audio está alrededor del nivel de umbral (*threshold*) ya que el efecto de compresión es muy notable y si la señal fluctúa alrededor de este nivel se pueden introducir

distorsiones en ella, de ésta forma, la transición suave permite una compresión más suave y gradual.

El codo recibe este nombre por que es la parte en la gráfica de respuesta de entrada / salida del compresor donde se empieza a hacer el cambio. Ejemplos de los conceptos de transición dura (*hard knee*, codo duro o rotula dura) y transición suave (*soft knee*, codo suave o rótula suave) se presentan en las figuras 11 y 13.

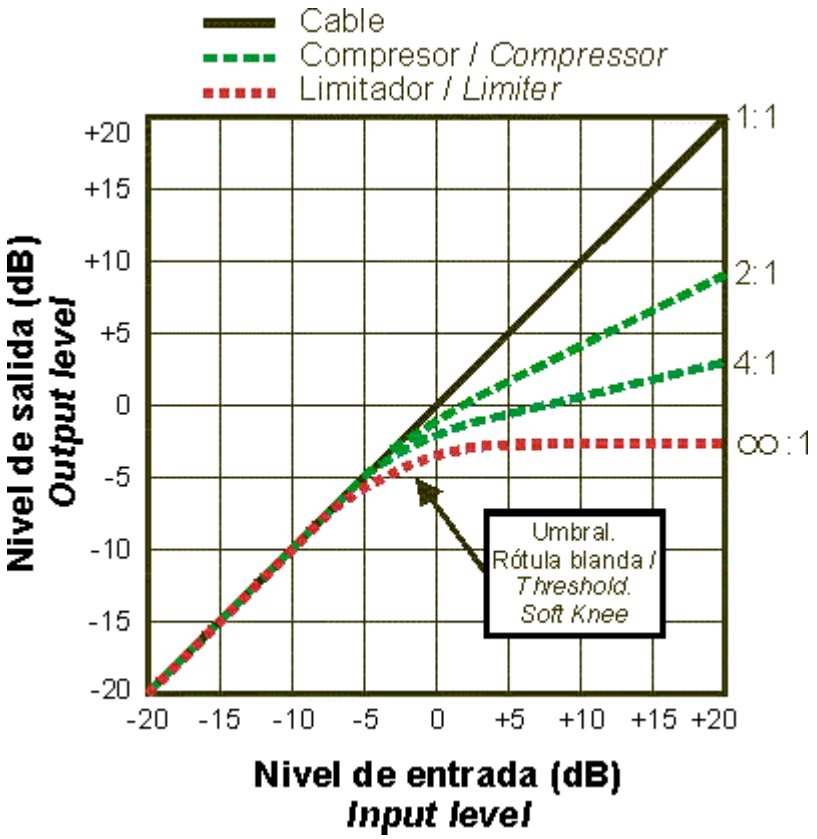


Figura 13. Transición suave (*soft knee,* codo suave o rotula suave). Fuente: www.doctorproaudio.com

1.5.3.5. Enlace estéreo (stereo link).

Cuando los procesadores de dinámica se utilizan para procesar una señal de dos canales (*stereo*), se hace necesario juntar el proceso de compresión/limitación entre ambos canales para que suceda en ambos de forma simultánea, si esto no se hace la imagen sonora será diferente y pueden

presentarse cambios en la ubicación espacial del efecto estereo, pues parecerá que cambia de un lado hacia otro.

1.5.3.6. Ganancia de salida.

Como se mencionó anteriormente, la función del compresor/limitador es atenuar la señal, lo cual se compensa subiendo un poco el volumen a la salida del sistema para aprovechar mejor el rango dinámico de los dispositivos que se conectan al compresor, sin embargo, al subir el volumen también se aumenta el nivel de los ruidos de fondo, por lo cual frecuentemente se utiliza la compuerta de ruido en conjunto con el compresor.

1.5.3.7. Automático.

Es común encontrar la posibilidad de variar cada uno de los parámetros vistos anteriormente (normalmente los tiempos de ataque y relajación) automáticamente en función de las características de la señal. El modo automático suele funcionar bien para una compresión sutil, pero cuando se requieren compresiones especiales o efectos, se recurre al modo manual.

1.5.3.8. Cadena lateral (*sidechain*).

Normalmente la señal que se está comprimiendo es la señal a la que se aplica la detección, para comprobar si pasa el umbral (*threshold*) o no. Sin embargo, en la mayor parte de los compresores es posible utilizar una señal externa en el circuito de detección a través de la cadena lateral (*side-chain*, a veces llamado *key*). El diagrama se puede observar en la figura 2 presentada anteriormente en este capitulo, donde se ilustra como se puede controlar el disparo de la compresión con una señal externa, auque la que se esté comprimiendo sea la señal principal de audio. La cadena lateral (*sidechain*) es utilizada para crear diferentes usos del compresor como se ilustrará más adelante.

1.5.3.9. Medidores.

Normalmente se puede encontrar en el compresor un medidor de atenuación o ganancia que es aplicada a la señal, que es representado por una cadena de *leds*, para poder evaluar si se está procesando y si se está haciendo de manera excesiva. También se pueden mostrar medidores de entrada y salida de la señal.

1.5.3.10. Limitador.

Según lo enunciado en páginas anteriores, para que un compresor se comporte como limitador, su relación de compresión debe ser mayor a 10:1, preferiblemente más grande. A diferencia de la compresión el limitador actuará como una barrera para todas las señales que superen el umbral (*threshold*), esto para evitar que los picos de señal dañen los amplificadores o saturen los dispositivos de amplificación, son más un sistema de protección que de procesamiento de audio. Los limitadores se activan de manera ocasional, de lo contrario será muy evidente, ya que al recortarse la señal, esta será distorsionada y afectará la calidad sonora.

El tiempo de ataque debe ser rápido, para evitar la saturación y los picos, normalmente el umbral (*threshold*) se ajusta dos o tres dBs por debajo del nivel máximo que no se desea rebasar, puesto que el limitador se demora un tiempo en llegar a su atenuación final, el limitador solo se activara en los picos más altos. La respuesta del limitador se puede observar en la figura 12 ejemplo tres.

1.6. COMPUERTA DE RUIDO.

La compuerta de ruido Junto con el compresor / limitador son las unidades de procesamiento dinámico mas importantes. La compuerta de ruido (*noise gate*) en la unidad dinámica encargada de eliminar el ruido de fondo que se

presenta cuando no hay señal. Lo que hace principalmente es cortar cualquier señal que se encuentre por debajo de un umbral(*threshold*) de nivel de intensidad (dBs), permitiendo eliminar ruidos de fondo (audibles cuando la fuente sonora baja su nivel de intensidad o se calla) o cortar sonidos de instrumentos que fueron captados por un micrófono destinado a captar otra fuente sonora.

Por consiguiente la compuerta de ruido tendrá dos estados: abierto (cuando la señal que entra es mayor del punto de umbral o *threshold*) y cerrado (cuando la señal no supera el punto de umbral o *threshold*).

Una de sus aplicaciones se observa en microfonía en la grabación de una batería también pueden hacer uso de las compuertas de ruido para evitar que recoger el sonido de unas percusiones con el micrófono destinado a otras. Por ejemplo, el micrófono del bombo sólo estaría activo en los golpes de bombo, mientras que el micrófono del redoblante sólo tendría abierta la puerta en los golpes a este instrumento, de forma que el sonido del redoblante no se presente en el micrófono del bombo y viceversa de tal manera que cuando no hay presencia de sonido en el micrófono la compuerta se cierra para evitar el ruido exterior (como respiraciones, siseo, etc). También es utilizado para quitar el ruido inherente que se presenta en el sistema de grabación ya que cuando el sonido se presenta en el micrófono el sonido de la fuente enmascara el ruido dado que su nivel de volumen es mas alto, eliminando de esta manera el problema del ruido en los espacios de silencio por medio de las compuertas.

La compuerta de ruido se da cuando la relación de atenuación es mayor o igual a 1:10. Este efecto se puede comprender mejor en la grafica 14 sobre la característica de entrada / salida de la compuerta de ruido / *downward expander* .

1.6.1. Característica entrada / salida.

En la grafica de la característica de la entrada / salida de la compuerta de ruido / *downward expander* se pueden observar varios aspectos importantes.

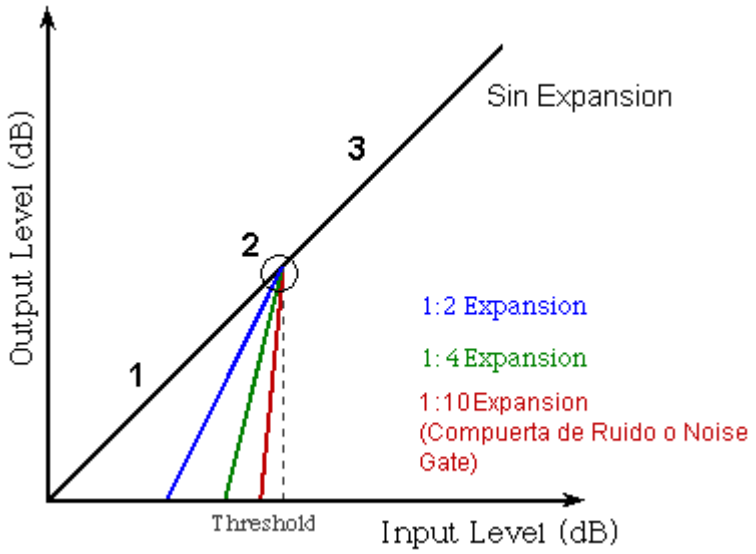


Figura 14. Característica entrada / salida de una compuerta de ruido / *downward expander*. Fuente: www.harmony-central.com

En la zona 1 es donde se encuentra la señal por debajo del punto de *threshold* o umbral, entonces la señal se atenúa. Puede que se elimine totalmente (compuerta de ruido) o con cierta proporción (*downward expander*), dependiendo del requerimiento establecido, en la grafica se puede observar una relación de expansión de 1:2, 1:4 y de 1:10. Como se puede ver en esta zona la característica de entrada / salida cambia su pendiente. Una relación de expansión 1:2 indica que por cada dB en la señal de entrada ubicada por debajo del punto de *threshold* se atenuara dos veces a la salida , igualmente para las otras relaciones de expansión hacia abajo(*downward expander*).

Cuando la expansión supera o es igual a 1:10 se dice que esta trabajando como una compuerta de ruido. Aunque teóricamente se considera que la relación de compuerta de ruido es uno a infinito (1:∞), ya que la pendiente es totalmente vertical.

En la zona 2 es donde se encuentra el punto de *threshold* o umbral, por debajo de este punto la compuerta de ruido se cierra.

La zona 3 es la parte de la característica en la cual la compuerta de ruido se encuentra abierta , la señal sale con la misma intensidad (dB) como entro la compuerta de ruido, o sea que hay una relación 1:1.

1.6.2. Respuesta de la compuerta de ruido/*downward expander* en el tiempo.

En la siguiente gráfica podremos observar la manera como se comporta la compuerta de ruido(*downward expander*) en el tiempo, cuando le es aplicada una señal.

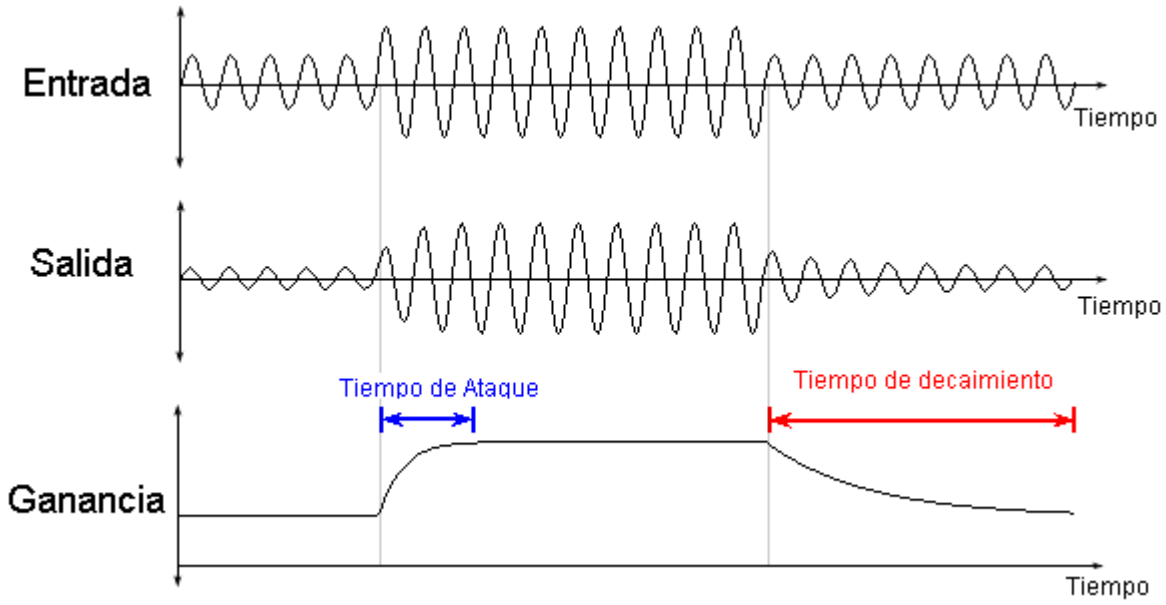


Figura 15. Respuesta de la compuerta de ruido/*downward expander* en el tiempo.
Fuente: www.harmony-central.com

En la figura es posible apreciar que cuando la señal de entrada supera el punto de *threshold* o umbral la compuerta de ruido no hace el cambio de nivel de manera abrupta, sino que va cambiando la ganancia hasta que llega a alcanza un valor igual a uno (relación 1:1), y el tiempo que se demora en hacerlo es llamado tiempo de ataque, el cual es variable. Análogamente cuando la intensidad de la señal en la entrada esta por debajo del punto de umbral la compuerta de ruido cambia su ganancia, pero el cambio tampoco es abrupto y

el tiempo que transcurre en este procedimiento es llamado tiempo de decaimiento o relajación.

Se puede observar en la grafica cuando la señal de entrada esta por debajo del punto de *threshold* , la señal a la salida se ve disminuida en cierta proporción.

1.6.3. Controles.

El ajuste de una compuerta de ruido es complejo, ya que puede necesitar características muy diferentes en función del tipo de señal, si se quieren evitar aperturas o enmudecimientos en falso. Por ello son necesarios numerosos controles. El siguiente gráfico muestra los controles mas comunes de una compuerta de ruido (*Noise Gate*) / *downward expander*.

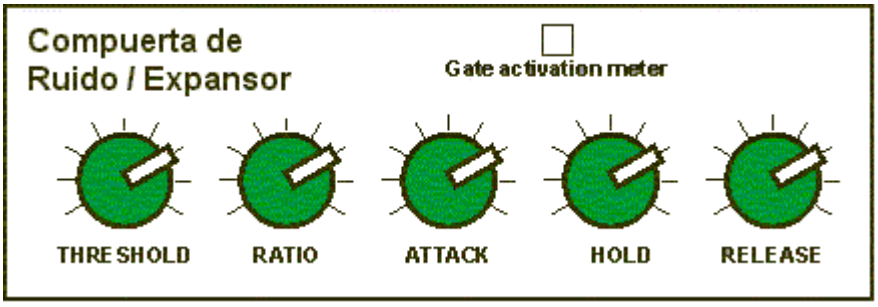


Figura 16. Controles de la compuerta de ruido / *downward expader.* Fuente: www.doctorproaudio.com

1.6.3.1. Umbral (*Threshold*).

Cuando la señal cae por debajo de este nivel umbral se pone en funcionamiento el procesador de dinámica y comienza a cerrarse la puerta. En general este control se ha de ajustar lo más bajo posible sin que ocurran aperturas en falso, puesto que así se preserva la señal deseada.

1.6.3.2. Tiempo de ataque (*attack time*).

Este es el tiempo que tarda la puerta en abrirse desde que la señal supera del nivel de umbral. Los tiempos de mínimos de ataque pueden oscilar entre 10 y

100 us (microsegundos) dependiendo del tipo de unidad, mientras que los tiempos máximos oscilan entre 200 ms (milisegundos) y 1 s. Cabe mencionar sin embargo, que estos valores varían dependiendo de fabricantes y modelos. Tiempos muy rápidos pueden crear distorsión, pues modifican la forma de onda de las frecuencias graves que son más lentas. Por ejemplo, un ciclo de 100 Hz dura 10 ms, con lo que un tiempo de ataque de 1 ms tiene tiempo de modificar la forma de onda, generando así distorsión. Por otro lado, un tiempo demasiado largo privará a un sonido de percusión de su ataque inicial. En general, debe ser lo más rápido posible sin causar "clics" o distorsiones en el ataque de las señales.

1.6.3.3. Tiempo de decaimiento o relajación (*release time*).

Es el inverso del tiempo de ataque, es decir, el tiempo que se tarda en pasar del estado sin procesar (sin atenuación, relación 1:1) al estado donde la señal se atenúa o enmudece. Los tiempos de relajación son mucho más lentos que los de ataque, y suelen oscilar, según modelos, entre 2-10 ms y 3-5 segundos. El gráfico muestra la diferencia entre conmutar con un tiempo de relajación más lento o más rápido.

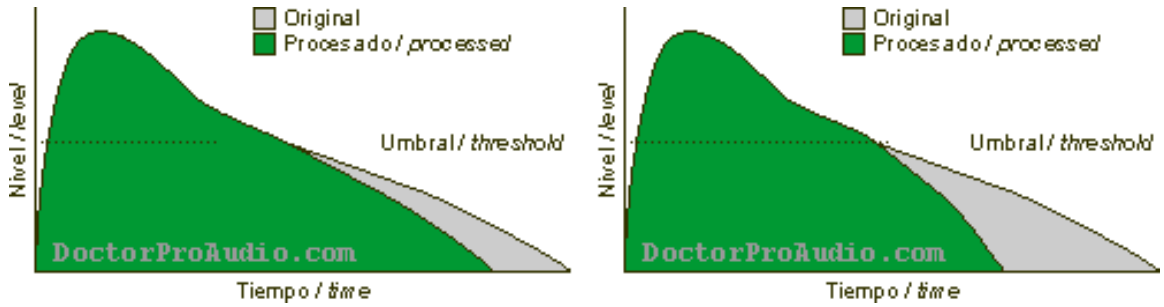


Figura 17.tiempos de relajación. Fuente: www.doctorproaudio.com

1.6.3.4. Tiempo de mantenimiento (*hold time*).

Es el tiempo mínimo que la puerta permanecerá abierta. Se usa para impedir que la puerta se cierre en falso si ocurre una caída de señal de corta duración. Se suele poder ajustar entre casi cero y hasta varios segundos. A continuación

podemos ver un gráfico que ilustra los diferentes tiempos que entran en juego en una puerta de ruido.

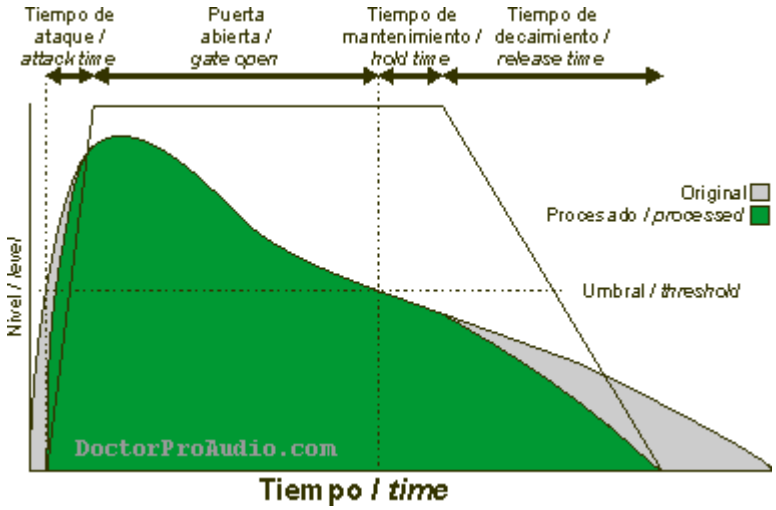


Figura 18. Tiempo de mantenimiento. Fuente: www.doctorproaudio.com

1.6.3.5. Relación de Conmutación (*ratio*).

Las puertas más sencillas cierran la salida del todo, aunque lo habitual en una unidad específica de conmutación es que se proporcione control sobre la cantidad de atenuación que se proporciona a las señales que no superan el nivel de umbral, dejando pasar una parte de la señal. La ventaja de dar cierta atenuación a las señales en vez de enmudecerlas totalmente es que cuando la puerta se abra lo hará de una forma más suave, puesto que la señal no tiene que partir de cero. La reducción de nivel se puede realizar con dos familias diferentes de aparatos. Las relaciones de conmutación, pueden ser con una compuerta de ruido o como un expansor hacia abajo(*downward expander*), esto se puede observar ver en la grafica 17.

1.6.3.5.1. La Compuerta de ruido.

Esta proporciona una atenuación prefijada, ya sea un enmudecimiento absoluto o bien cualquier valor intermedio, al que se denomina rango o profundidad (*range* o *depth*). Por ejemplo podríamos atenuar 20 dB o 40 dB,

o bien cerrar la puerta del todo (atenuación de -infinito). Normalmente lo que se hace es cerrar la compuerta cuado la señal de entrada este por debajo del punto de *threshold*.

1.6.3.5.2. El Expansor hacia abajo (*downward expander*).

Este tipo de modelos funciona de manera contraria al compresor, y proporciona un control de relación de conmutación (ratio) o pendiente de atenuación. De esta manera la señal se atenúa en mayor medida en cuanto menor sea su valor. La relación de atenuación funciona de manera equivalente a los compresores, especificando la cantidad de compresión (atenuación) que se aplica a la señal. Estas relaciones están expresadas en decibelios, así que por ejemplo una relación de 1:6, quiere decir que una señal que caiga 1 dB por debajo del aquí punto de *threshold* se reducirá en 6 dB, mientras que una señal que caiga 3 dB por debajo del mismo punto se reducirá a 18 dB. De igual manera una relación de 1:3 (uno a tres) significa que una señal que caiga 1 dB por debajo del punto de *threshold* se verá atenuada en 3 dB más (puesto que el nivel se reducirá de -1 dB a -4 dB. El signo negativo en estos dBs indica que están por debajo del umbral). A partir de 1:10 ya se considera que el expansor hacia abajo funciona como puerta de ruido, aunque en teoría una puerta ideal tendría una relación de 1:infinito (cualquier nivel de entrada por debajo del umbral entregará una salida cero).

En la práctica un expansor hacia abajo (*downward expander*) se usa de forma muy similar a la puerta de ruido pura, con la diferencia de que la expansión hacia abajo es más gradual y suave, y es más difícil equivocarse con los tiempos de ataque y relajación. El siguiente gráfico ilustra la diferencia entre el expansor hacia abajo con diferentes relaciones (ratio) y la puerta de ruido con diferentes rangos (profundidades) de atenuación.

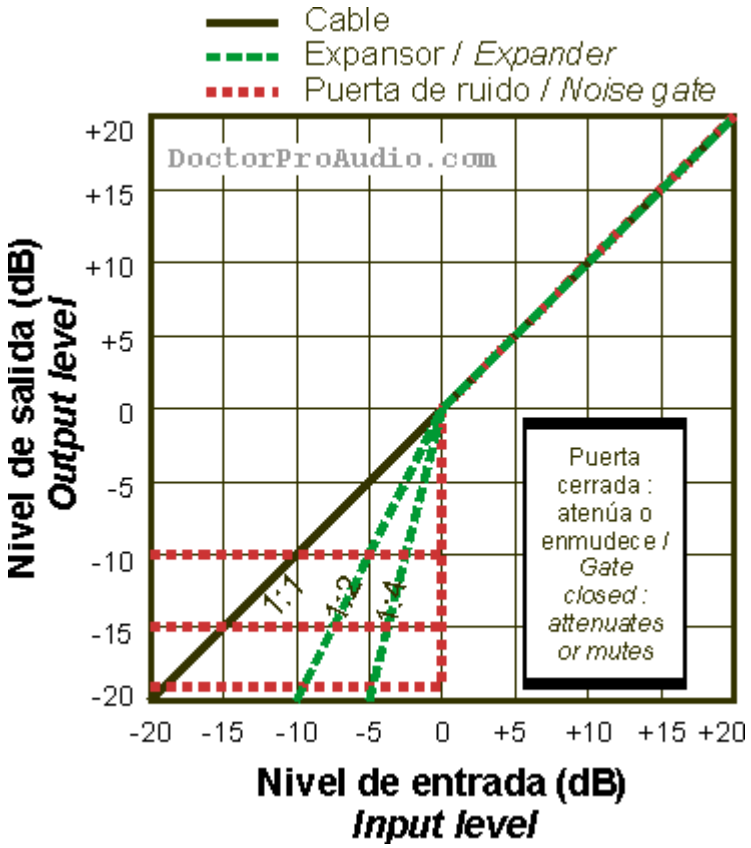


Figura 19. relación de puerteo (*ratio*). Fuente: www.doctorproaudio.com

1.6.3.6. Enlace estereo (*stereo link*).

En los procesadores de dinámica en general, cuando se usan para procesar una señal de dos canales (estéreo, o bien dos canales de sonido que se presentan juntos), se hace necesario enlazar la acción de conmutación en ambos canales para que suceda en ambos a la vez. De lo contrario, la imagen sonora será confusa y cambiante desde el centro hacia a un lado y hacia el otro. Las unidades monofónicas pueden tener un cable para enlazar varias unidades.

1.6.3.7. Automático.

Cada vez es más común que exista la posibilidad de controlar alguno de los parámetros listados (normalmente los tiempos de ataque y relajación) de forma automática en función de las características de la señal. Este control activa o desactiva esa opción. En general, el modo automático suele funcionar bien cuando se busca un efecto transparente y disimulado.

1.6.3.8. Cadena lateral (*sidechain*).

Normalmente la señal que se está conmutando es la misma que es monitorizada por el circuito de detección para comprobar si excede o no el umbral. Sin embargo, en la mayor parte de las puertas de ruido es posible utilizar una señal externa en el circuito de detección a través de la cadena lateral (*side-chain*, a veces llamado *key*). De esta forma la señal externa controla cuando se abre o cierra la puerta, aunque lo que se conmuta es la señal principal que entra a la puerta. Suele existir un conmutador que asigna el circuito de detección a una entrada externa para posibilitar esta función.

Lo más habitual es utilizar un ecualizador en la cadena lateral; de hecho algunas puertas de ruido vienen equipadas con ciertas capacidades de ecualización ya integradas con esta finalidad. Es posible por ejemplo atenuar los agudos en la señal que va al circuito de detección para evitar que los platillos abran la puerta del bombo.

1.6.3.9. Indicadores.

Existe un indicador luminoso(*LED*) que indica si la puerta está abierta o cerrada. También es común algún tipo de indicación que indique cuando se ha cruzado el nivel de umbral y cuando se ha terminado la envolvente de atenuación.

2. LÓGICA DIFUSA (*FUZZY LOGIC*) Y DISEÑO DEL CONTROL

2.1. INTRODUCCIÓN

La lógica difusa como método de control es utilizado en aquellos casos donde la planta o el sistema a controlar es bastante complejo y/o presenta alta no linealidad. El modelo matemático de tales sistemas es difícil de determinar.

Las propiedades de los sistemas de control se representan por ecuaciones integro – diferenciales, por lo cual, su análisis y síntesis se hace mediante la solución de estas. Cuando el sistema se supone lineal , se pueden utilizar para su solución los métodos de las transformadas (Laplace, Fourier, etc.).

Estos modelos, en la mayoría de los casos, se han basado en presunciones y linealidades, con el fin de simplificar sus variables y aplicar las transformadas anteriormente mencionadas. De esta manera, se puede encontrar una aproximación con métodos teóricos. Sin embargo, un sistema real es siempre, no lineal. Por tal razón, desde hace varios años se vienen buscando y trabajando métodos alternativos para la solución de tales procesos, uno de los más difundidos se basa en la Lógica Difusa (*FUZZY LOGIC*) .

La lógica difusa representa el conocimiento y la experiencia humana , para manipular e implementar un control sobre algún proceso. Este método, basado en las estadísticas y en el uso de variables lingüísticas, reduce la complejidad en el desarrollo del controlador y resulta bastante útil en los sistemas no lineales.

Las variables lingüísticas tienen cierto grado de veracidad (o certeza) o falsedad, y con éstas se arman proposiciones para el control.

En este capítulo se presenta una breve tutoría para entender el funcionamiento del control difuso, sin entrar en el trasfondo matemático. Se desarrollará un controlador de dos entradas y una salida, actuando sobre un proceso, que para este caso, será un control de nivel de audio. Se asumirá que la salida deseada se logra mediante la posición de un *fader* (o atenuador de nivel).

El diagrama básico del controlador difuso es el siguiente:

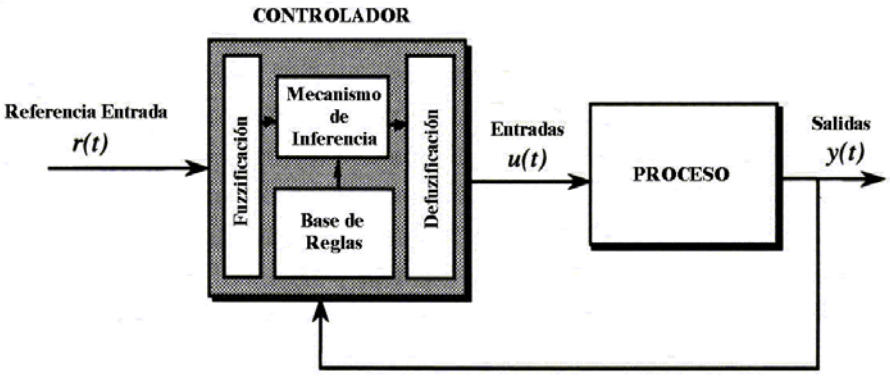


Figura 20. Diagrama básico del controlador difuso Fuente: Passino, Kevin M. Fuzzy control. Ohio,1998. p.23.

El controlador difuso esta compuesto por 4 bloques básicos: la fusificación, el mecanismo de inferencia, la base de reglas y la defusificación.

Este sistema esta compuesto por dos entradas la referencia r(t) y la realimentación de la salida de la planta y(t), además el controlador cuenta con una salida u(t) que es la entrada a la planta donde se realiza el proceso.

2.1.1. Fusificación (*Fuzzification*)

Este es el bloque inicial del controlador difuso y es el encargado de transformar las variables de entrada (r(t), y(t)) en variables del tipo lingüísticas.

Como resultado de la fusificación se obtienen los valores de certidumbre de cada conjunto difuso.

2.1.2. Base de reglas (*Rule base*)

Es la que contiene el conocimiento para tomar las decisiones para las acciones de control, la base de reglas se hace dependiendo de las acciones que se quieren realizar y según de las entradas. A determinadas entradas se obtiene una salida establecida por el diseñador del control.

2.1.3. Mecanismo de inferencia (*Inference mechanism*)

Es el encargado de tomar las decisiones de control, interpretando las entradas y aplicándole la base de reglas para obtener una salida con determinado grado de certeza.

2.1.4. Defusificación (*Defuzzyfication*)

Es el encargado de transformar las decisiones de tipo difuso de las etapas anteriores, en salidas de valor real para el control del sistema.

Mas adelante se hablará con detalle de cada uno de los bloques y cómo es su funcionamiento en un caso específico, como lo es el control de *fader* motorizado.

El controlador difuso se debe ver como un sistema de toma de decisiones, que leerá los valores de una referencia y los comparará con la salida de la planta o del sistema bajo control, y dependiendo de cómo es el resultado de esta operación se obtendrá una salida acorde.

Una forma simple de explicar la toma de decisiones en el controlador, se realiza mediante las preposiciones lingüísticas "Si... Entonces", en la cual se parte del conocimiento previo humano de los valores que se pretenden obtener a la salida.

Para diseñar un controlador difuso se debe seguir diferentes pasos:
Se debe escoger cuales son las entradas y salidas de sistema, luego se debe describirlas en el plano difuso con las funciones de membresía adecuadas, se debe crear la base de reglas, que es uno de los pasos importantes en el diseño del control difuso, ya que en este punto se implementa el conocimiento del experto para la toma de decisiones, de allí se genera una tabla con los valores obtenidos. Luego de realizados los pasos anteriores se arma todo el control difuso interconectando cada uno de los bloques.
Ahora se muestra el proceso de diseño paso a paso de un control difuso para un *fader* motorizado, y se explican mas detalladamente cada uno de los bloques del controlador.

2.2. CONTROL DIFUSO DE UN *FADER* MOTORIZADO

Para desarrollar el diseño de este control se debe tener claro que el *fader* es un potenciómetro lineal utilizado principalmente en aplicaciones de audio.
De esta manera, el sistema a considerar tiene como partes principales la entrada y la salida de la señal eléctrica de audio y un motor de directa (CD). Éste cambia la posición del *fader* afectando el valor de la salida respecto a la entrada de audio (puede atenuarla o hacer el valor de salida igual al de entrada).
La señal de audio tomará valores entre 0 y 5 Voltios, y el motor de directa se alimentará entre 0 y 5 Voltios (a mayor voltaje mayor su velocidad de respuesta)

2.2.1. Entradas y salidas del controlador difuso

En este sistema la variable a controlar es el rango dinámico de una señal de audio, por tal motivo, las entradas al sistema serán la salida y la entrada de la señal de audio al fader. Luego como ocurre en la mayoría de los controles clásicos (PID, proporcional integral derivativo), las entradas para el control serán el error *e* y la derivada del error *de/dt* , estas serán halladas con la salida de audio en el fader. La referencia del control será una función de la

entrada de audio del sistema y se indicará el nivel de audio que se desea a la salida del sistema. La salida del control será un nivel de voltaje y la polaridad para alimentar un motor de CD. El diagrama de bloques del control se puede ver en la siguiente figura

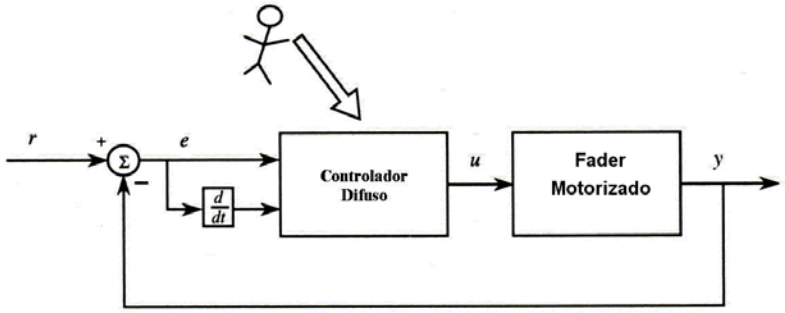


Figura 21. Controlador difuso realimentado Fuente: Passino, Kevin M. Fuzzy control. Ohio,1998. p.25.

En la figura 2, las entradas finales del controlador difuso serán el error (e(t)) que es igual a: *e(t)=r(t)-y(t)* y la derivada del error (*de/dt*). El significado físico de estas variables se interpreta como la distancia respecto a la referencia (valor que se desea en la salida), y qué tan rápido se esta alejando o acercando a ella.

Las entradas se deben asignar en el plano difuso. La manera de transformarlas en variables lingüísticas es la siguiente:

Las señal de audio oscilaran entre 0 y 5 Voltios, por lo tanto la referencia puede tomar valores entre 0 y 5 Voltios.

Con estas entradas físicas, al entrar al plano difuso se obtienen los siguientes rangos: el error entre –5 y 5, y igualmente con la derivada del error.

La salida hacia el motor CD, será un valor entre 0 y 9 Voltios, y un sentido de giro especificado por control.

2.2.2. La Base de reglas y el Conocimiento

El comportamiento de la planta es definido en términos de variables lingüísticas que describen la conducta dinámica de las entradas y salidas de

control. De esta manera es posible incluir la experiencia y el conocimiento previo adquirido por el diseñador quien define la base de reglas. Eso permite que, este control no convencional, sea sencillo de diseñar.

En la definición del controlador difuso para el *fader* se usan las siguientes descripciones

"Error" para describir *e(t)*

"Cambio en el error" para describir *de/dt*

"Salida" para describir *u(t)*

Esto, simplemente para otorgarle un nombre a las variables que se van a usar. Estas variables lingüísticas pueden tomar valores como:

"Negativo_ grande"

"Negativo_ pequeño"

"Cero"

"Positivo_ pequeño"

"Positivo_ grande"

Estos valores lingüísticos serán representados en las tablas con un número (también puede ser con letras u otra manera simple) para efectos de sencillez y compacidad. Las convenciones son las siguientes:

"-2" para representar "Negativo_ grande"

"-1" para representar "Negativo_ pequeño"

"0" para representar "Cero"

"1" para representar "Positivo_ pequeño"

"2" para representar "Positivo_ grande"

Luego de escoger las variables lingüísticas se procede a describir las situaciones que se pueden presentar en el *fader*, así:

- La palabra el "error es Positivo_ grande" se puede utilizar para representar la situación donde la posición del *fader* esta muy arriba respecto a la posición donde se tiene la referencia.

- la palabra el "error es cero" se puede utilizar para representar la situación donde la posición del *fader* se encuentre muy cercana a la posición de referencia y no necesariamente en la referencia (cabe recordar que la lógica difusa trabaja con rangos o espacios difusos).
- la palabra "el error es Positivo_ grande y el cambio en el error es Positivo_ pequeño" se puede utilizar para la situación donde la posición del fader está muy arriba respecto a la posición de la referencia y además se esta alejando de esta posición a un ritmo pequeño.

Y así sucesivamente. de esta forma se puede representar las diferentes situaciones en las que se puede encontrar el *fader* , es decir, se explica de manera lingüística la dinámica del sistema.

2.2.3. Las reglas

Ahora se utilizarán las descripciones lingüísticas para determinar reglas, que serán las encargadas de almacenar el conocimiento o la experiencia del diseñador. Por medio de ellas, el control difuso toma las decisiones de cómo actuar en cada uno de los casos. Se ilustrarán unos ejemplos de cómo se pueden obtener las reglas para el control del *fader*.

Cabe recordar que las preposiciones lingüísticas de la lógica difusa son de la forma "Si ... entonces" donde se halla una decisión de salida, dependiendo de las entradas obtenidas.

- *SI* el error es Positivo_ grande *Y* el cambio en el error es Positivo_ grande *ENTONCES* salida es Negativo_ grande.

Esto ocurre en la situación donde la posición del *fader* se encuentra muy por debajo respecto a la referencia, y además, se esta alejando de ella a una velocidad grande. Se puede observar entonces, que la acción a tomar es hacer el valor de la Salida grande con un sentido de giro positivo para lograr que el motor gire en la dirección opuesta, y que el *fader* se empiece a acercar a la referencia. Esto se puede observar mejor en la siguiente gráfica.

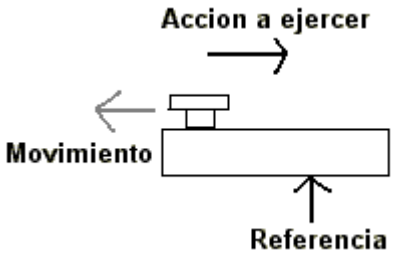


Figura 22. primera preposicion lingüística

- *SI* el error es cero *Y* el cambio de error es Positivo_ pequeño *ENTONCES* salida es Negativo_ pequeña.

Esto ocurre en la situación donde la posición del fader esta cerca de la referencia y viene con una velocidad pequeña, entonces la acción a tomar es hacer la salida pequeña y negativa (el sentido de giro) para contrarrestar la velocidad que tiene, evitando así que sobrepase la referencia por la inercia que lleva. Esto se muestra en la siguiente figura

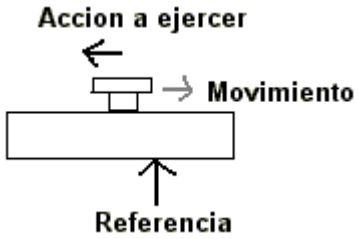


Figura 23. Segunda preposición

- *SI* el error es Negativo_ grande *Y* el cambio en el error es Negativo_ grande *ENTONCES* salida es Positivo_ grande.

Esto ocurre en la situación donde la posición del fader está muy arriba respecto a la referencia y además se esta alejando de esta a una velocidad grande. Entonces la acción a realizar es hacer la salida grande y negativa (sentido de giro) para hacer que el motor gire en la dirección opuesta y que el *fader* se acerque a la referencia. Este ejemplo de puede ver gráficamente en la siguiente ilustración.

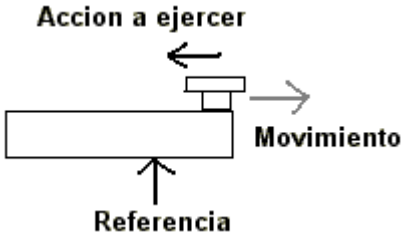


Figura 24. Tercera preposición

De esta manera se representan el conocimiento y las ideas del diseñador. Aún no se habla de valores exactos para la salida, si no que se determina en que espacio (rango) difuso, el cual es llamado función de membresía, se encuentra la variable de salida.

Por lo anterior, es que este método de control no convencional describe en cierta forma la manera del pensamiento humano para tomar las decisiones.

2.2.3.1. Tabla de reglas

De la manera como se explicó anteriormente se pueden seguir creando las reglas para el control difuso del fader motorizado y para cada una de las posibilidades lingüística que se puedan generar en el sistema. Como el número de variables de entrada es dos, y el número de valores lingüísticos que puede tomar estas variables es cinco, entonces se genera una tabla 5x5 obteniéndose 25 reglas que representarán las acciones a tomar en el sistema.

La forma más simple de escribir la tabla es utilizando números como se explicó anteriormente, de esta manera resulta:

Tabla 1. Tabla de reglas

SALIDA		Cambio en el error de/dt				
		-2	-1	0	1	2
Error e(t)	-2	2	2	2	1	0
	-1	2	2	1	0	-1
	0	2	1	0	-1	-2
	1	1	0	-1	-2	-2
	2	0	-1	-2	-2	-2

En la tabla 1 se facilita encontrar la salida para determinadas entradas. En el caso de la celda que esta sombreada la regla sería:

SI el error es Negativo_ pequeño *Y* el cambio en el error es cero, *ENTONCES* la salida es Positivo Pequeño.

2.2.4. Cuantificación del conocimiento

Hasta este punto sólo se han trabajado las variables de manera abstracta, ahora se cuantificaran numéricamente para automatizar el proceso del control. Se tienen entonces varios aspectos a cubrir:

2.2.4.1. Funciones de membresía

Las funciones de membresía representan el espacio o rango difuso en el que se encuentra una variable, y en este espacio se define qué tanta certidumbre hay de que una entrada pertenezca a determinada función de membresía. Teniendo en cuenta que la lógica difusa es un método estadístico, la certeza o certidumbre pueden tomar valores entre 0 y 1. Este indicador será representado por la letra μ.

Para explicar esto de manera simple, se recurrirán a unos ejemplos, que se basarán en la siguiente figura:

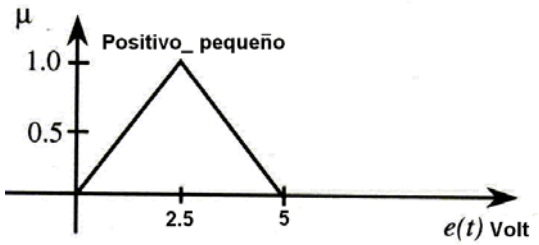


Figura 25. Función de membresía Positivo_ pequeño

- Si se tiene el error *e(t)* = 2 voltios, esta pertenecerá a la función de membresía Positivo_ pequeño y la certeza de que la variable este en esta función de membresía será μ=0.8.
- Si el error es *e(t)* = -1 voltios, entonces la certeza que tiene este valor de encontrarse en la función de membresía Positivo_ pequeño será μ=0.

- Si el error fuera *e(t)* = 2.5 voltios, entonces se tendría una certeza absoluta que este valor esta en la fusión de membresía Positivo_ pequeño ya que la certidumbre es igual a μ=1.

Se puede observar en el ejemplo que las funciones de membresía cuantifican las variables lingüísticas de manera continua, diciendo con qué certeza pertenece un valor a determinado espacio difuso.
Como se puede ver en la grafica anterior, la función de membresía tiene una forma triangular, pero esta solo es una de las formas que puede tomar, existen otros tipos de geometrías como se ilustra en la siguiente grafica.

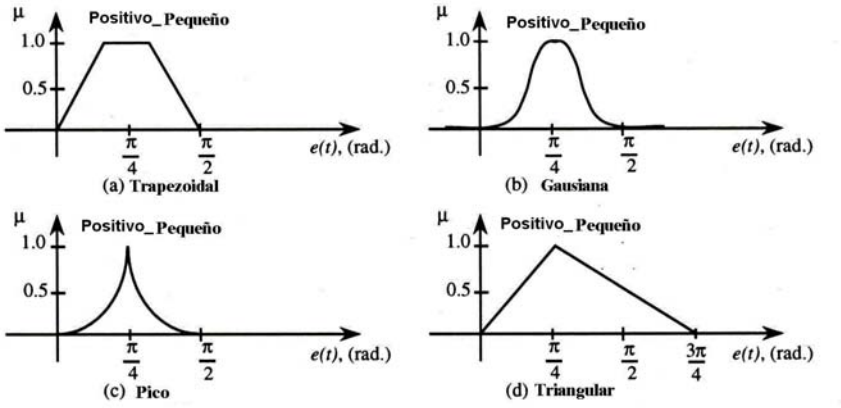


Figura 26. Formas de las funciones de membresía Fuente: Passino, Kevin M. Fuzzy control. Ohio,1998. p.31.

Existen diversas formas para las funciones de membresía que se escogen según como se quiera expresar el espacio difuso. Dependiendo de la forma de la función de membresía, esta puede ser más sencilla o más compleja en el momento de implementar el control difuso.
Para cada variable lingüística se determina su función de membresía, y se construye la gráfica del espacio difuso para cada una de las variables de entrada y salida que se tengan en el controlador. En este caso son el error *e(t)*, el cambio en el error *de/dt*, y la salida *u(t)*.

En el diseño del controlador difuso para el *fader,* se decidió implementar las funciones de membresía por medio de formas triangulares, ya que son mas fáciles de construir y necesitan menos tiempo de cómputo, factor que es importante teniendo en cuenta que se realizaran los cálculos en un microprocesador de 8 bits. Así, se obtienen las funciones de membresía y los espacios difusos que se pueden observar en la figura 5.

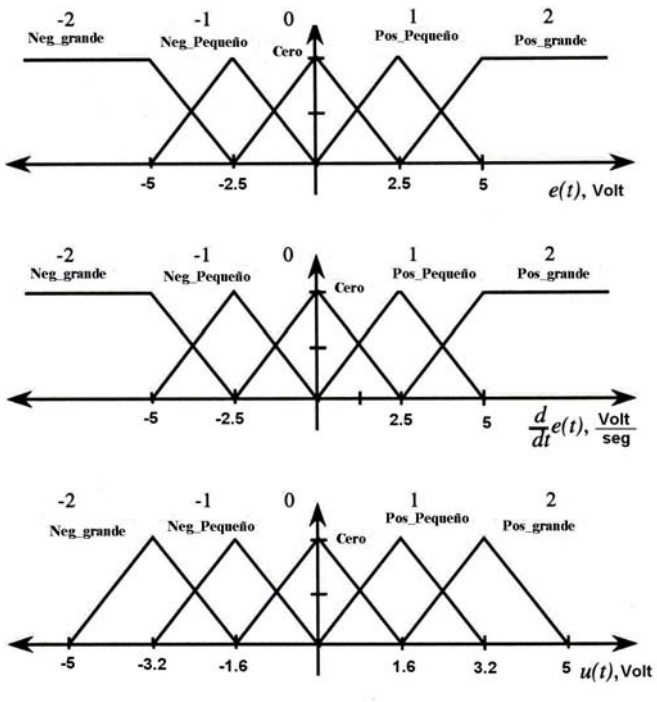


Figura 27. Cuantización de las variables.

Luego de explicar cómo se obtiene el conjunto del espacio difuso en el cual se van a tomar las decisiones, se describirá cómo trabaja cada bloque del control difuso, y cómo interacciona con las reglas creadas.

2.2.5. Fusificación (*Fuzzyfication*)

El proceso de fusificación, es relativamente sencillo. Básicamente consiste en hacer la lectura de las entradas del sistema a controlar, que en este caso serían la referencia y la salida de la señal de audio del *fader*, y basándose en estos valores calcular el error y cambio de error. Posteriormente, se evalúan

dichos valores en las funciones de membresía y se almacenan los valores de las certezas encontradas. Este resultado se entrega al siguiente bloque del control, que es el mecanismo de inferencia.

A manera de ejemplo, se puede tomar la referencia como 4 voltios y la salida del *fader* como 3 voltios, entonces, el error sería igual a *e(t)* = ref – *y(t)* , e(t)=1 voltio, y suponiendo un cambio en el error igual a *de/dt* =2 voltios/s., se obtiene:
Para el error e(t), los siguientes valores de certidumbre de las funciones de membresía:

u (Negativo_ grande) = 0 *u* (Negativo_ pequeño) = 0
u (Cero) = 0.6 *u* (Positivo_ pequeño) = 0.4
u (Positivo_ grande) = 0

Para el cambio en el error se obtienen las siguientes certezas:
u (Negativo_ grande) = 0 *u* (Negativo_ pequeño) = 0
u (Cero) = 0.2 u(Positivo_ pequeño) = 0.8
u (Positivo_ grande) = 0

Como se observa en la gráfica 5, el número máximo de funciones de membresía a las que puede pertenecer un valor son dos, entonces los cálculos solo se hacen para máximo dos funciones de membresía, ahorrando tiempo de cómputo.

2.2.6. Base de reglas y Defusificación (*Defuzzyfication*)

A continuación se ilustra como se utilizan las reglas descritas por la tabla Número 2. Se deben seguir dos pasos:
- Hacer las combinaciones posibles entra las funciones de membresía activas en el error y el cambio de error para hallar las funciones de membresía de la salida que están activas.

- Se halla el valor de certidumbre de cada una de las funciones de membresía activas en la salida, aplicando los criterios de la función "mínimo" o la función "producto de certezas".

Para explicar más claramente se sigue con los valores tomados en el ejemplo anterior, entonces observando en la tabla 1 de reglas se obtiene que las funciones de membresía activas para la salida serian las resultantes de las siguientes preposiciones:

- *SI* el error es cero *Y* el cambio en el error es cero *ENTONCES* la salida es cero.
- *SI* el error es cero *Y* el cambio en el error es Positivo_ pequeño *ENTONCES* la salida será Negativo_ pequeño.
- *SI* el error es Positivo_ pequeño *Y* el cambio en el error es cero *ENTONCES* la salida será Negativo_ pequeño.
- *SI* el error es Positivo_ pequeño *Y* el cambio en el error es Positivo_ pequeño *ENTONCES* la salida será Negativo_ grande.

También se pueden encontrar estos resultados en la tabla siguiente

Tabla 2. funciones de membresía de la salida activas

SALIDA		Cambio en el error de/dt				
		-2	-1	0	1	2
Error e(t)	-2	2	2	2	1	0
	-1	2	2	1	0	-1
	0	2	1	0	-1	-2
	1	1	0	-1	-2	-2
	2	0	-1	-2	-2	-2

Ahora, para hallar la certidumbre que tiene cada una de estas funciones de membresía de la salida, se recurre al operador mínimo o el operador producto. Esto debido a que la cuantificación matemática del operador "*Y* " en la preposición *SI ... Y... Entonces* , se puede hallar por cualquiera de estos dos

operadores dependiendo del criterio del diseñador. La definición de estos operadores es la siguiente:

Mínimo: Este operador escoge el menor valor entre dos números, por ejemplo: mínimo(0.8,0.4) = 0.4.

Producto: se hace el producto entre dos valores numéricos que se tengan, por ejemplo: producto(0.8,0.6)= 0.48.

Aplicando el operador mínimo a las conclusiones de las preposiciones anteriores se encuentran las siguientes certezas para las funciones de membresía de la salida:

- *1) u* (Cero) = mínimo (0.6,0.2)= 0.2
- *2) u* (Negativo_ pequeño) = mínimo(0.6,0.8) = 0.6
- *3) u* (Negativo_ pequeño) = mínimo(0.4,0.2) = 0.2
- *4) u* (Negativo_ grande) = mínimo(0.4,0.8) = 0.4

se pueden ver gráficamente estas respuestas en las siguientes graficas

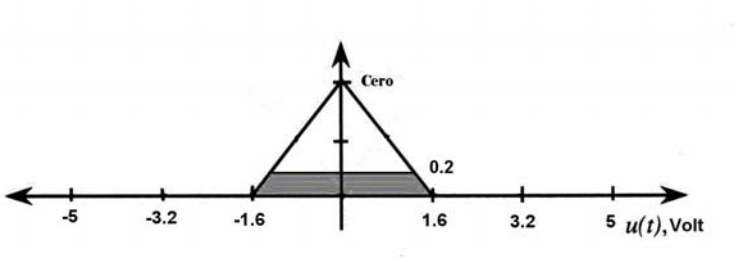


Figura 28. certeza de la función de membresía de salida 1

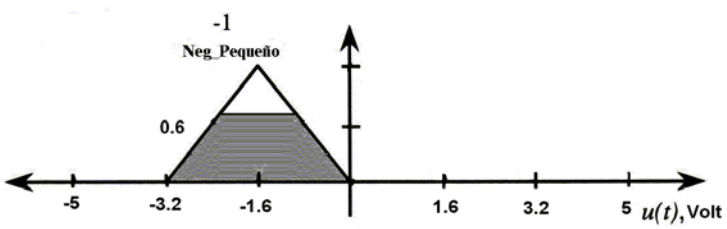


Figura 29. certeza de la función de membresía de salida 2

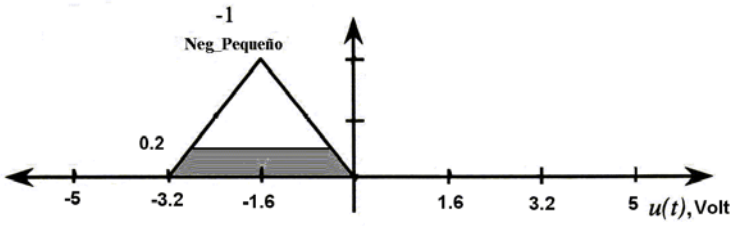


Figura 30. certeza de la función de membresía de salida 3

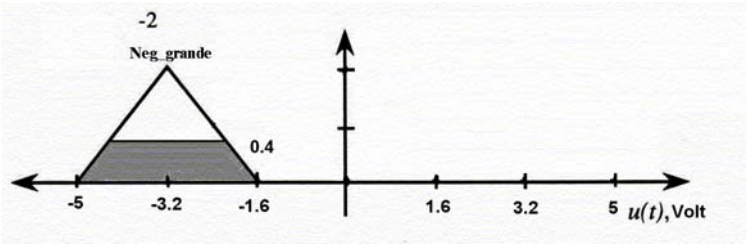


Figura 31. certeza de la función de membresía de salida 4

2.2.7. Defusificación (*Defuzzyfication*)

También llamada concreción, una vez halladas las funciones de membresía activas de la salida, y determinada la certidumbre o certeza de cada una, se dispone a hallar el valor numérico de la acción a tomar.

Existen varios métodos de defusificación, uno de los más comunes es el del Centro de Gravedad (COG *center of gravity*). Este método toma las áreas de las funciones de membresía obtenidas del mecanismo de inferencia y encuentra su centro de gravedad el cual se convierte en la salida del control.
Como se utilizaron las funciones de membresía con forma triangular, su área es fácil de encontrar. Esta corresponde al área sombreada que se puede observar fácilmente en las graficas anteriores.

El centro de gravedad esta definido por la siguiente fórmula:

$$u^{crisp} = \frac{\sum b_i \int u_{(i)}}{\sum \int u_{(i)}}$$

donde $\int u_{(i)}$ es el área sombreada de la función de membresía activa y la salida será u^{crisp} donde *crisp* denota el valor exacto o concreto de la salida del control. Esta es la formula clásica para hallar el centro de gravedad de cualquier área, pero se debe tener en cuenta que:

El área sombreada de las funciones de membresía es un valor finito.

- para hallar $\int u_{(i)}$, el área de un triangulo truncado simétrico (área sombreada) se utiliza la siguiente formula:
- $w(h-\frac{h^2}{2})$
- Donde h es la altura a la que se corta el triángulo y w es el ancho de la base del triángulo.

b_i es el centro de la función de membresía respecto al eje horizontal.

Para los ejemplo anteriores obtenemos que:

$$u_{crisp} = \frac{(0)(0.576)+(-1.6)(1.344)+(-1.6)(0.576)+(-3.2)(1.024)}{0.576+1.344+0.576+1.024} = -1.8036$$

según lo anterior, tenemos que aplicar al motor CD un voltaje de 1.8 con sentido de giro negativo.

Si suponemos el error igual a *e(t)* = 0 y el valor del cambio en el error igual a *de/dt* =0.625, haremos el mismo procedimiento para hallar el valor de salida.

Todo el procedimiento se puede representar de manera gráfica en la siguiente figura:

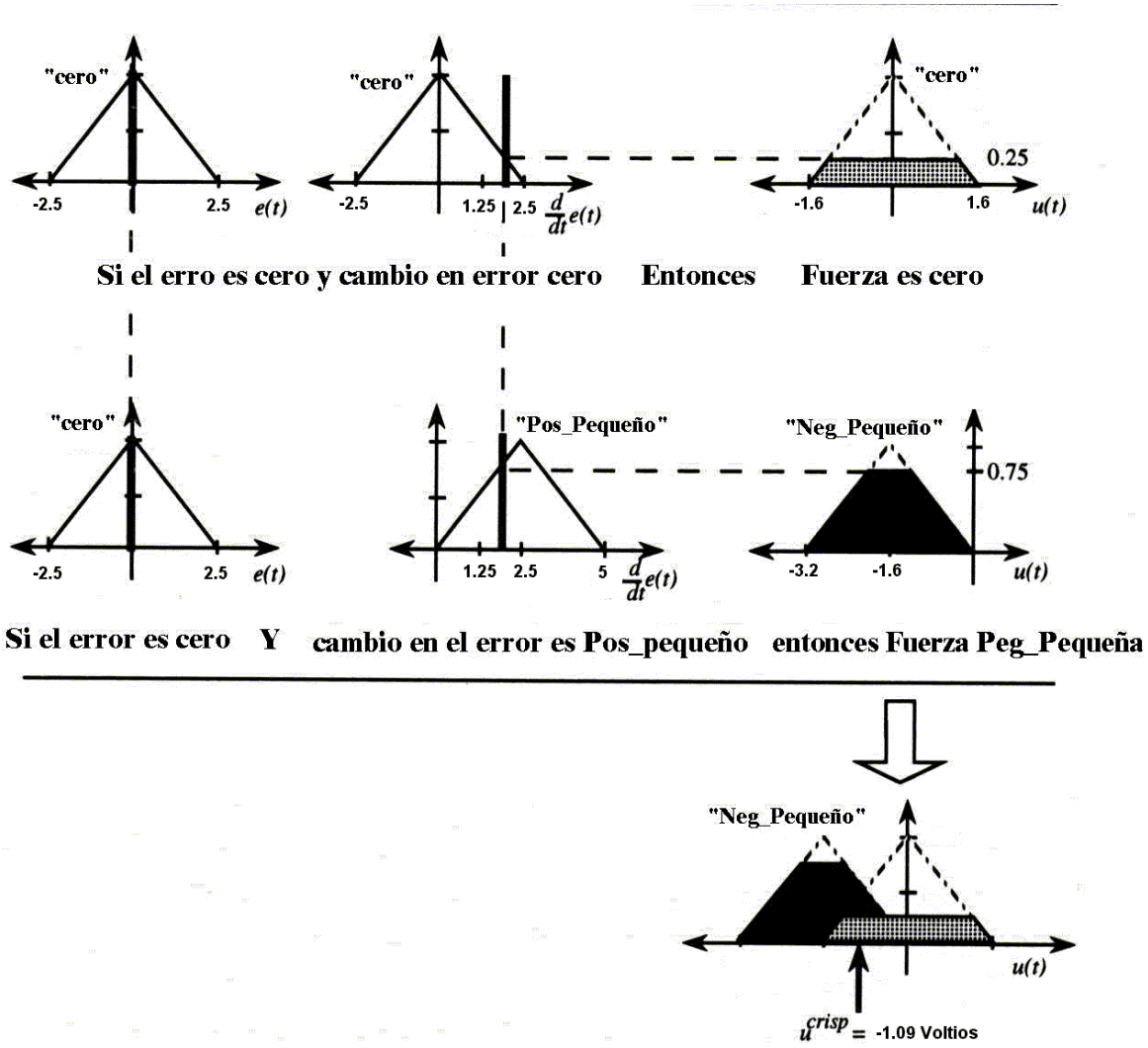


Figura 32. procedimiento gráfico del análisis difuso

3. HERRAMIENTA DE DESARROLLO

Metrowerks y Motorola han dejado disponible al público una edición especial del entorno de desarrollo para microcontroladores HC08: CodeWarrior v2.0, la cual es libre y puede compilar hasta 4 Kbytes de código C. Una de las ventajas importantes de esta nueva versión es que adiciona el simulador de P&E Microcomputer Systems Inc., el cual cuenta con una máquina virtual que permite simular la CPU, periféricos e interrupciones de todos los microcontroladores HC08 actuales, lo que facilita el proceso de depuración de las aplicaciones desarrolladas en lenguaje C.
Mas información acerca de este software se puede encontrar en: http://www.metrowerks.com/embedded/motoHC08/

3.1. AMBIENTE DE DESARROLLO INTEGRADO: CODEWARRIOR.

El CodeWarrior entorno integrado de desarrollo (IDE, *Integrated Development Environment*) es una aplicación que provee un grupo de herramientas para desarrollar software usando una interfaz gráfica (GUI).
El entorno integrado de desarrollo (IDE) permite:
- Editar código.
- Navegar a través del código.
- Examinar el código.
- Compilar el código.
- Enlazar código.
- Simular el código

Adicionalmente permite configurar opciones para generación de código, manejo de proyectos y configuración de herramientas.

Las herramientas con las que cuenta el IDE incluyen:

- Editor de código fuente.
- Navegador de código fuente.
- Compilador.
- Enlazador.
- Ensamblador.
- Depurador Y Simulador.

3.1.1. Requerimientos del sistema.

Los requerimientos del sistema de cómputo para correr el CodeWarrior para la familia 68HC908 de Motorola son los siguientes:

Hardware:

- Microprocesador Intel Pentium o AMD K6 class de 133 MHz.
- Memoria RAM de 64MB.
- 130 MB de espacio en disco duro libre.

Software:

- CD-ROM para la instalación del programa
- Windows 95/98/NT (se recomienda NT 4.0)

El software se puede descargar de la página del fabricante, en www.metrowerks.com/hc08/, y para más información se puede escribir a info@metrowerks.com

3.1.2. Herramientas integradas para la familia mc68hc908.

CodeWarrior ambiente integrado de desarrollo (IDE): esta es la aplicación que permite la creación del software. El IDE controla los proyectos, el editor de código fuente, el navegador, el compilador, el enlazador y el depurador.

Compilador: el compilador incluido en el CodeWarrior para la familia 68HC908 es un compilador de C que cumple el estándar ANSI C. Utilizado en conjunto

con el enlazador, permite generar aplicaciones para la familia 68HC908 y librerías.

Ensamblador: el ensamblador para la familia 68HC908 es un ensamblador autónomo con una sintaxis fácil de utilizar.

Enlazador: el enlazador es del tipo *Executable and Linker Format* (ELF). Permite generar archivos ELF para la aplicación y soporta el formato de salida *S-record*.

Depurador (*Debug*) y Simulador: el depurador integrado en el CodeWarrior permite controlar la ejecución del programa y ver el comportamiento interno de la aplicación mientras el programa está corriendo. El depurador es utilizado para encontrar problemas en la aplicación y en el software. El depurador y simulador se encuentran integrados en esta versión y son conocidos como el software Pemicro de P&E Microcomputer Systems Inc, el cual es muy utilizado para el desarrollo de proyectos con los microcontroladores de la familia 68HC08. Se puede encontrar mas detalles en la pagina web: http://www.pemicro.com/

3.1.3. Tutorial y creación de proyectos.

El desarrollo de una aplicación en el IDE del CodeWarrior para la familia 68HC908 de Motorola responde a la filosofía de trabajo orientado a proyectos. En pocas palabras una aplicación es encapsulada dentro de un proyecto.

Para la creación de proyectos nuevos con el CodeWarrior se utiliza un tipo de plantilla especial o estructura de programa prediseñada denominada *project stationary,* el cual es una plantilla que describe un proyecto prediseñado, con archivos de código fuente completos, librerías y todas las configuraciones apropiadas para el compilador y el enlazador.

Un buen tutorial para la utilización del Codewarrior de Metrowerks se puede obtener de la dirección Web: http://microe.udea.edu.co/cursos/ieo-944

Este tutorial se encuentra en el anexo A.

3.2. LENGUAJE C PARA EL 68HC908

El ensamblador o *assembler* como lenguaje de programación permite un uso mas eficiente de los recursos debido a que permite optimizar el código del programa y el uso de la memoria. Pero también tiene sus inconvenientes, el tiempo de desarrollo es muy largo; el proceso de cambiar un programa es complejo; el código tiene que estar bien documentado para poder entender de manera rápidamente el funcionamiento del programa. No permite una buena programación estructurada.

Por otro lado en una programación en lenguaje C , el tiempo de desarrollo es mas corto, el código es reutilizable, el manejo de punteros es mas fácil, se pueden tener complejas estructuras de programación, se dispone de librerías que ayudan al desarrollo de el código, y una de las principales ventajas es que se puede insertar código en ensamblador permitiendo de esta manera la reescritura de algunas partes del mismo al ensamblador. Sin embargo, el trabajo con lenguaje C trae sus inconvenientes como, mayor uso de memoria y menor eficiencia que el ensamblador.

En la industria se prefieren los desarrollos de programación en lenguaje C para microcontroladores dado que toma mucho menos tiempo de desarrollo al poder hacer reutilización de códigos generados anteriormente. Además, este lenguaje permite tener portabilidad por dos razones, el programa que se genera para un microcontrolador puede funcionar en otro diferente (en la herramienta de desarrollo sólo se especifica el microcontrolador con el que se esta trabajando), y el archivo que contiene el código en lenguaje C generado con una aplicación , se puede cargar con otras aplicaciones (para otros microcontroladores) que soporten el lenguaje C.

Los formatos de variables básicas que se manejan en la familia HC908 se pueden ver en la siguiente tabla:

Tabla 3. Tipos de Valores Escalares

Tipo de Dato	Tamaño	Rango de Valores Definidos	
		Mínimo	Máximo
Char (*unsigned*)	8 bits	-128	127
Signed Char	8 bits	-128	127
Unsigned Char	8 bits	0	255
Signed Short	16 bits	-32768	32767
unsigned short	16 bits	0	65535
Enum (S*igned*)	16 bits	-32768	32767
Signed Int	16 bits	-32768	32767
Unsigned Int	16 bits	0	65535
Signed Long	32 bits	-2147483648	-2147483647
Unsigned Long	32 bits	0	4294967295

Para las variables de punto flotante se pueden ver los valores en la siguiente tabla:

Tabla 4. Tipos de Valores de Punto Flotante

Tipo de Dato	Tamaño	Rango de Valores Definidos	
		Mínimo	Máximo
Float	32 bits	1.17549435E-38F	3.402823466E+38F
Double	64 bits	2.2259738585972014E-308	1.7976931348623157E+308
Long Double	64 bits	2.2259738585972014E-308	1.7976931348623157E+308

La herramienta de desarrollo Codewarrior permite trabajar con lenguaje estándar ANSI C, C++ y con lenguaje ensamblador (*assembly*).

Para lograr un trabajo óptimo con las herramientas planteadas, se acostumbra integrar en el código en *assembler* con el lenguaje C, lo cual se puede lograr de varias maneras, como se describe a continuación.

3.2.1. Insertar Funciones En Ensamblador.

Para especificar que una parte del código debe ser interpretado en lenguaje ensamblador (Asembler, *assembly*), se utiliza la palabra reservada asm. La sintaxis es la siguiente:

```
asm función
{
        declaraciones locales
        instrucciones en ensamblador
}
```

3.2.2. Insertar lenguaje ensamblador en línea.

para insertar lenguaje ensamblador en alguna línea del código utiliza la siguiente sintaxis:

```
asm { mov A,x}
```

4. DISEÑO Y DESARROLLO DE HARDWARE

4.1. DISEÑO DEL HARDWARE

La unidad dinámica que se implementa en este prototipo es un compresor de audio, el hardware está conformado por los componentes mostrados en el siguiente diagrama de bloques:

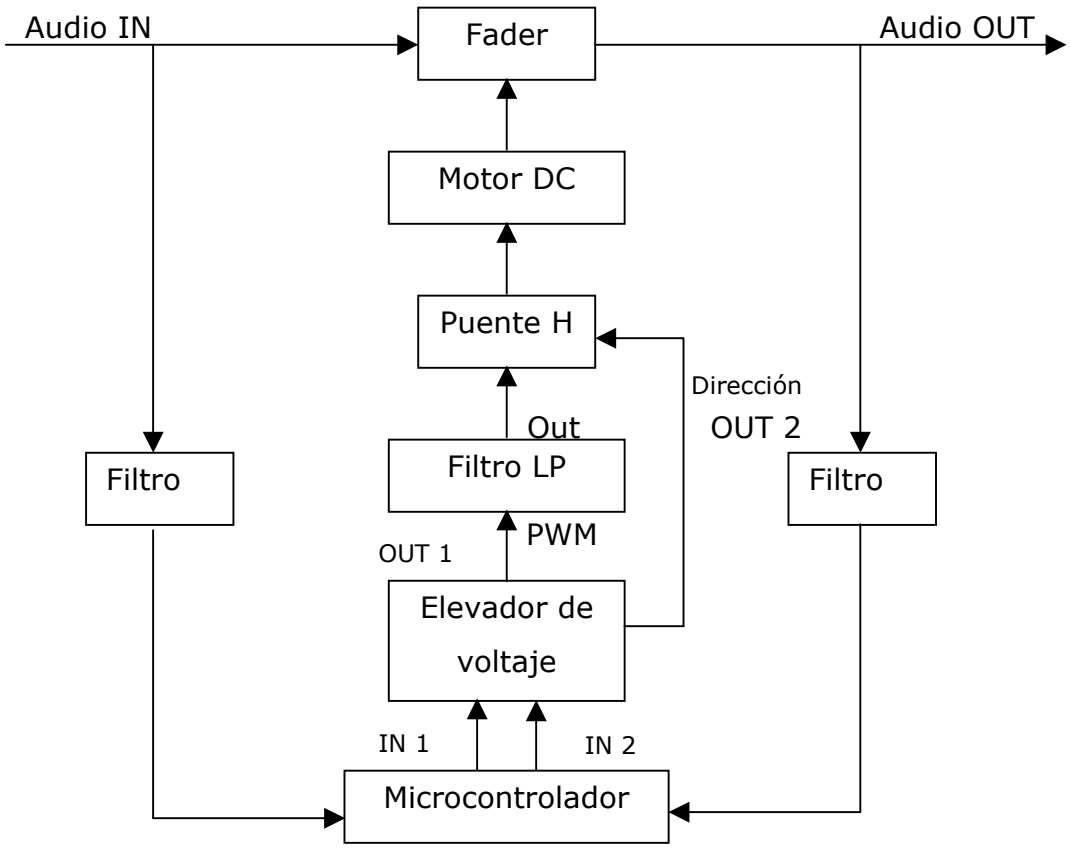


Figura 33. Diagrama de Bloques

4.1.1. Filtro Audio

Este filtro pasabajos se encargara de evitar los problemas que se pueden presentar por *aliasing* con la señal de audio al ser muestreada por el microcontrolador, además de filtrar el ruido de alta frecuencia que se pueda introducir en la señal.

4.1.2. Microcontrolador

Se utiliza un microcontrolador de 8 bits ampliamente difundido y conocido en el medio como lo es el Motorola MC68HC908GP32, del cual se obtiene suficiente información técnica de apoyo y desarrollo. Además, posee módulos de conversión análogo /digital y salida PWM (*pulse width modulation*)

El microcontrolador realiza las acciones de control mediante lógica difusa (*fuzzy logic*) sobre el motor DC, el cual manipula el fader. Para realizar dicho control se toma como entradas la señal de audio a regular y la salida del fader. Ver diagrama de bloques de la Figura 33.

El comando sobre el motor DC se basa en el módulo PWM (modulación por ancho de pulso-*pulse width modulation*). La CPU soporta Lenguaje de programación estándar y versátil, especialmente lenguaje C, para futuras actualizaciones del programa principal.

4.1.3. Elevador de voltaje

Teniendo en cuenta que las salidas del microcontrolador (dirección y PWM) son señales que oscilan entre 0 y 5 Voltios, y que el motor DC requiere mayores voltajes para su optimo funcionamiento, es necesario elevar el voltaje que entra al puente H.

4.1.4. Filtro LP (pasabajo - *low pass*)

Este filtro pasabajos es necesario para alisar la señal de PWM, obteniendo una señal análoga con nivel de directa aproximadamente constante.

4.1.5. Puente H

El puente H recibe la señal de dirección proveniente del elevador de voltaje, la cual indica el sentido de giro del motor. También tiene como entrada la señal analógica resultante del PWM filtrado, que maneja la cantidad de potencia entregada al motor DC.

4.1.6. Motor DC

Este elemento electromecánico, que esta acoplado al potenciómetro lineal a través de una cuerda, recibe señales de dirección de giro y velocidad del puente H.

4.1.7. Fader

El fader es un potenciómetro lineal utilizado para variar el volumen de señales de audio. La intensidad del volumen de la señal de salida de audio depende de la posición en la que se encuentra la perilla o cursor del mismo. Esta posición se varía con una cuerda enhebrada de tal forma que el movimiento rotatorio del motor se convierte en movimiento lineal en el fader.

4.2. IMPLEMENTACIÓN DEL HARDWARE

4.2.1. Microcontrolador

El microcontrolador seleccionado para desarrollar las tareas de control en el prototipo, es el motorola 68HC908GP32.

Entre las características mas importantes del microcontrolador se pueden destacar las siguientes:

- CPU de 8 bits de datos.
- Arquitectura CISC (complex instruction set)
- 87 instrucciones.
- 32Kbytes de memoria tipo Flash de programa

- 512 bytes de memoria RAM
- Módulo de conversión A/D (análogo digital), comunicación serial asincrónica, PWM, keyboard *interrup*, etc.

La selección de este microcontrolador se basó en la información técnica y el conocimiento previo sobre su manejo, además de la disponibilidad de herramientas de desarrollo de alto nivel como compiladores de lenguaje C.
El diagrama esquemático de la distribución de pines y conjunto de módulos se puede encontrar en el anexo B.

4.2.2. Elevador de voltaje

Para desarrollarlo se utilizó el integrado LM358, que es un amplificador operacional doble dispuesto en configuración de comparador. Se obtienen valores más altos de voltaje, dependiendo del valor de la alimentación del integrado. Se obtienen, entonces, los siguientes valores de entrada y salida para el dispositivo.

Tabla 5. valores de entrada / salida del elevador de voltaje

ENTRADA	SALIDA
5 Voltios	9 Voltios
0 Voltios	0 Voltios

El esquema electrónico de la configuración del comparador es el siguiente:

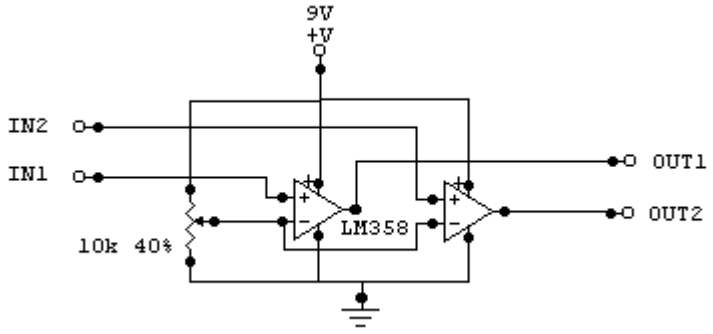


Figura 34. Esquema del Elevador de voltaje

La salida del PWM del microcontrolador se conecta al IN1 del elevador de voltaje, y la salida OUT1 se lleva al filtro. Análogamente, la salida de dirección del microcontrolador se conecta al IN2 del elevador de voltaje, y la salida OUT2 se conecta al pin de dirección del puente H que corresponde al pin 6.

4.2.3. Filtro LP (pasabajo - *low pass*)

Se implementa un filtro pasivo, teniendo en cuenta la sencillez de su realización y su estabilidad. Se utiliza un reóstato en el filtro RC, el cual brinda la posibilidad de ajustarlo a las mejores condiciones de comportamiento.

El diagrama esquemático del filtro pasabajos realizado es:

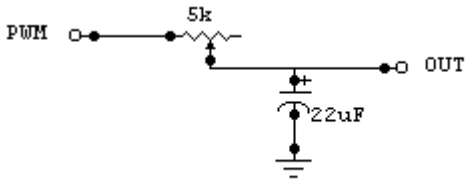


Figura 35. Esquema del Filtro pasabajos

4.2.4. Puente H

Para el puente H se utiliza el integrado TA7288P, en vista de que es fácil de conseguir en el mercado local y tiene, además, la capacidad de manejar la potencia y dirección de giro que se entrega al motor. Adicionalmente, este integrado puede operar uno o dos motores. Más datos acerca de este puente H se pueden encontrar en la hoja de datos del anexo C.

En la siguiente figura se puede observar la disposición de pines del integrado TA7288P.

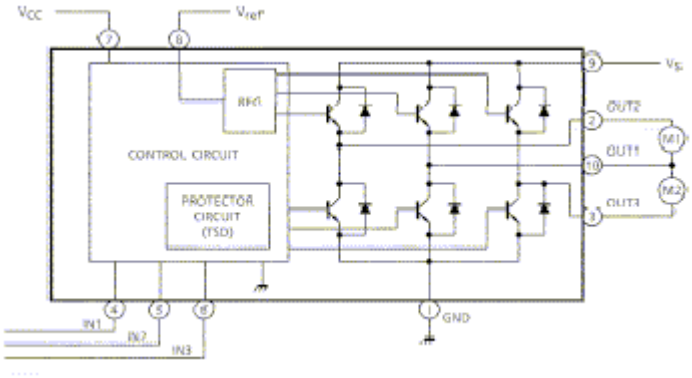


Figura 36. Esquema circuital del Puente H TA7288P

La conexión de los pines de este dispositivo queda establecida de la siguiente manera:

Tabla 6. Disposición de pines del puente H TA7288P

Pin	1	2	3	4	5	6	7	8	9	10
conexión	Tierra	Motor	Aire	9V	Tierra	Dirección	9V	PWM filtrado	9V	motor

4.2.5. Filtro de audio

Se implemento un filtro activo, para evitar la perdida de energía de la señal de audio, se diseño un filtro *butterworth* de orden 2. con una frecuencia de corte de 10 Khz.

El diagrama esquemático del filtro pasá bajo de audio, es el siguiente:

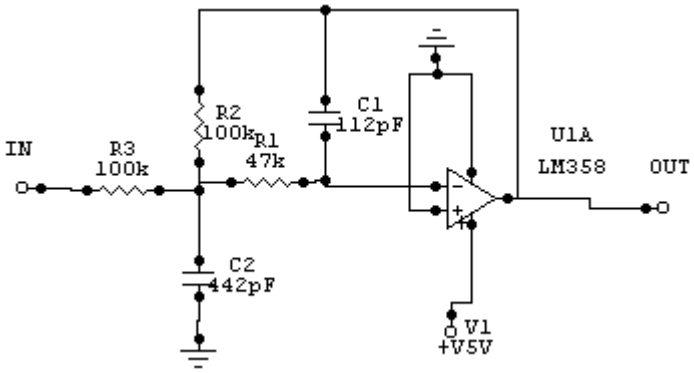


Figura 37. Esquema del filtro de audio

4.2.6. Fader y Motor DC

Dado que se necesitaba un fader con motor DC incorporado, para no implementar un acople mecánico entre estos, se utiliza un fader D460427 de la empresa Penny&Giles®. La velocidad del motor se controla con un PWM con frecuencia de 120HZ, pues con este valor se logran movimientos muy suaves en el motor.

El diagrama esquemático del fader motorizado D46027 de Penny&Giles® se encuentra en el anexo D.

Figura 38 Fader motorizado D46027 de Penny&Giles

5. DISEÑO Y DESARROLLO DE SOFTWARE

5.1. DISEÑO DEL SOFTWARE

El prototipo implementado es un compresor de audio, que se encargará de controlar el rango dinámico de una señal de entrada como se explicó en el capítulo 1. El software corresponde a un algoritmo de control inteligente basado en la lógica difusa.

El software se desarrolló en lenguaje C, que proporciona facilidades de implementación y mantenimiento. El programa consta básicamente de un llamado a rutinas que tienen funciones bien definidas, de tal manera que un conjunto de ellas se encargan de cada parte del control difuso, de la lectura de las entradas al microcontrolador, de la salida PWM y de los parámetros de configuración del compresor, *ratio* y *threshold*.
Para entender el diseño del software se debe tener claro el funcionamiento del controlador difuso simple de dos entradas una salida, tema que fue tratado anteriormente en el capítulo 2.

A continuación se explican las rutinas implementadas en el software:

5.1.1. Inicialización

En esta rutina el microcontrolador inicializa sus variables de manejo de puertos de salida y entrada, interrupciones, y los módulos a utilizar entre los cuales figuran el conversor análogo / digital , la comunicación serial asincrónica y el *timer* para generar el PWM.

5.1.2. Comunicación con el PC

Se realiza el establecimiento de la comunicación con el PC para obtener los valores de *ratio* y *treshold*, con los que funcionará el controlador difuso.

5.1.3. Datos del conversor A/D (análogo / digital)

Se toman los datos de las conversiones análogo digitales, y se da la multiplexación de canal, esto debido a que el microcontrolador solo puede hacer la conversión de un canal a la vez.

5.1.4. Referencia de control

Se determina la referencia a utilizar en el controlador difuso, siguiendo el esquema descrito a continuación:

Señal de Entrada < *Threshold* $\Rightarrow$ Señal de Salida = Señal de Entrada
Señal de Entrada > *Threshold* $\Rightarrow$ Señal de Salida = Señal de Entrada / Ratio

5.1.5. Error

En esta rutina se halla el error y la derivada del error, datos de entrada que requiere el controlador difuso para actuar.

5.1.6. Fuzificación

Tomando como entradas los valores de error y derivada de error, esta rutina se encarga de modificar estas entradas de manera que puedan ser interpretadas y comparadas por la Base de Reglas.

5.1.7. Mecanismo de inferencia

Evalúa las reglas que son pertinentes en el tiempo actual, con el fin de encontrar el valor de la entrada al controlador y sacar las conclusiones dependiendo de las reglas activas.

5.1.8. Defuzificación

En esta rutina se convierten las conclusiones obtenidas por el mecanismo de inferencia en salidas del microcontrolador, es decir, entradas reales para la planta.

5.1.9. Actuador

Se toma el valor de la salida de la rutina de defuzificación y se determina el sentido de giro del motor, adicionalmente, se escala el valor de la salida de la rutina de defuzificación para acomodarlo a los 8 bits con los que trabaja el microcontrolador seleccionado. Sin embargo, la configuración de su PWM es en 16 bits.

5.1.10. Interrupción del *timer*

En esta rutina de atención a la interrupción del *timer*, se configura la salida del PWM, es decir, el ancho del pulso, ya que el período es constante (120Hz).

5.2. IMPLEMENTACIÓN DEL SOFTWARE

Para la implementación del software en el microcontrolador 68HC908GP32 bajo el esquema de subrutinas, se tomaron como base los diagramas de flujo que especifican el funcionamiento de cada una de ellas. El diagrama de flujo general del programa se encuentra en la Figura 39, donde se puede observar los subprogramas que lo conforman y las interrupciones que utiliza.

EL programa esta formado por rutinas de inicialización del microcontrolador, inicialización de los módulos, subrutinas de control difuso y por la rutina de atención a la interrupción del PWM. El código fuente en lenguaje C con el programa se encuentra en el anexo E.

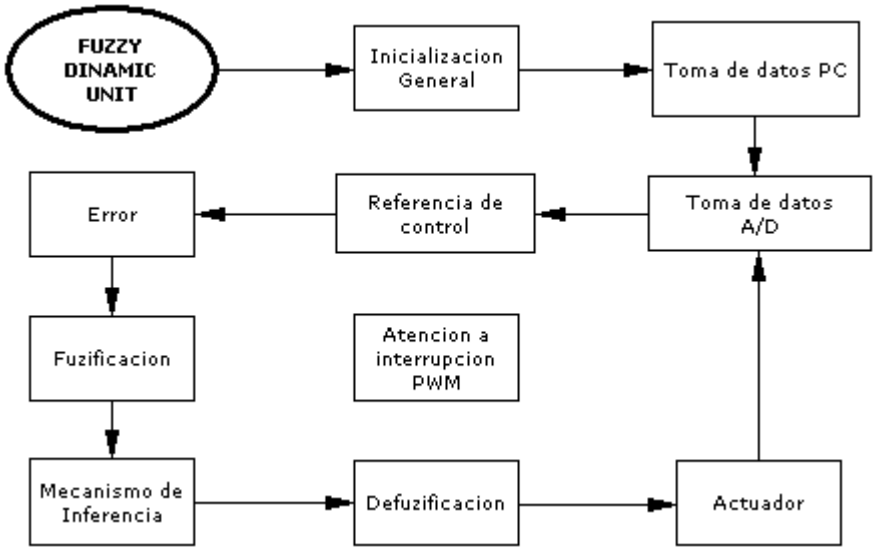


Figura 39. Diagrama de flujo general

5.2.1. Inicialización

Se cuenta con varias subrutinas para inicializar los registros del microcontrolador y de los módulos que se utilizan en el prototipo. El diagrama esquemático general de las inicializaciones es el siguiente.

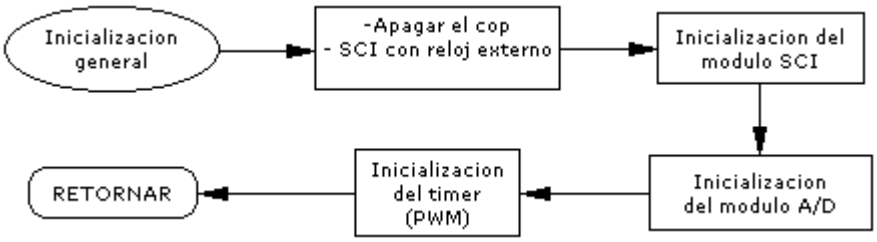


Figura 40. Diagrama de flujo de inicialización general

En la inicialización de microcontrolador se deshabilita el modulo COP, y se alimenta el modulo de comunicación serial con el reloj externo de 4.952 Mhz.

A continuación se presenta el diagrama de inicialización de cada módulo de manera más específica:

La comunicación serial se configuró con una velocidad de 19200 b, 8 bits de datos y 1 bit de parada.

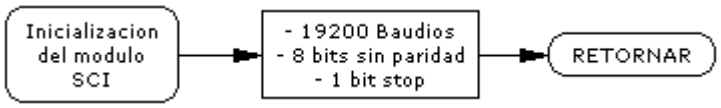


Figura 41. Diagrama de flujo de la inicialización de la comunicación serial

El modulo de conversión análogo / digital, se escogió con un *preescaler* de uno, de esta manera se tiene una velocidad de muestreo aproximadamente de 1 Mhz, y se habilita el canal 0 para la primera conversión.

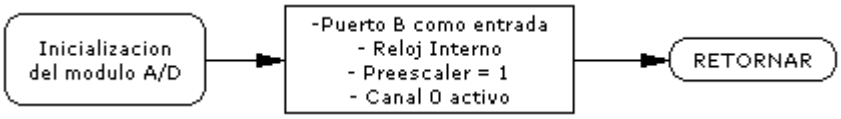


Figura 42. Diagrama de flujo de la inicialización del conversor A / D

Para configurar el TIM, módulo que genera la onda PWM, se tomó un periodo de 120 hz, con el cual se establece el tiempo del periodo de la señal. Se habilita el módulo para que genere interrupciones que generan la señal PWM. Se utiliza el modo PWM *unbuffered* por ser el mas sencillo de controlar y configurar.

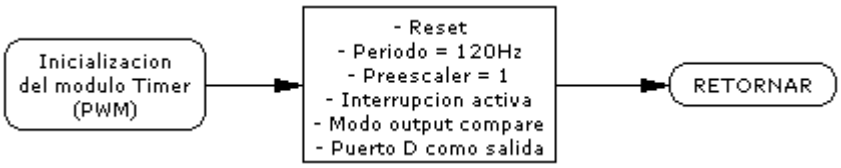


Figura 43. Diagrama de flujo de la inicialización del PWM

5.2.2. Comunicación con el PC

La comunicación serial asincrónica se trabaja a 19200 bps, ella obtiene los valores de *ratio* y el punto de *threshold* desde el computador necesarios para el trabajo del controlador difuso. Toma estos valores y los guarda en la memoria RAM del sistema.

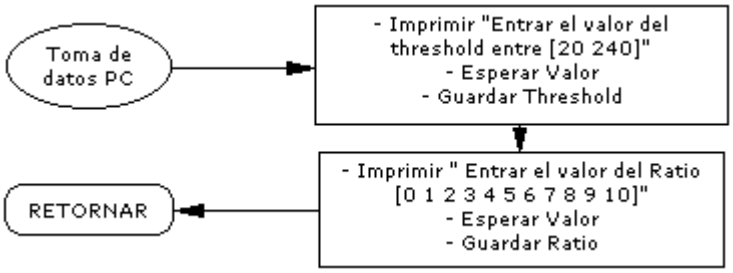


Figura 44. Diagrama de flujo comunicación PC

Se utiliza el registro *threshold* donde se guarda el valor de umbral, y el registro Ratio donde se encuentra el valor para la proporción de compresión.

5.2.3. Datos del conversor A/D (análogo / digital)

Esta rutina se encarga de obtener los valores de la conversión análogo / digital de las entradas de audio al prototipo. Debe multiplexar los canales ya que el módulo hace la conversión de un canal al tiempo.

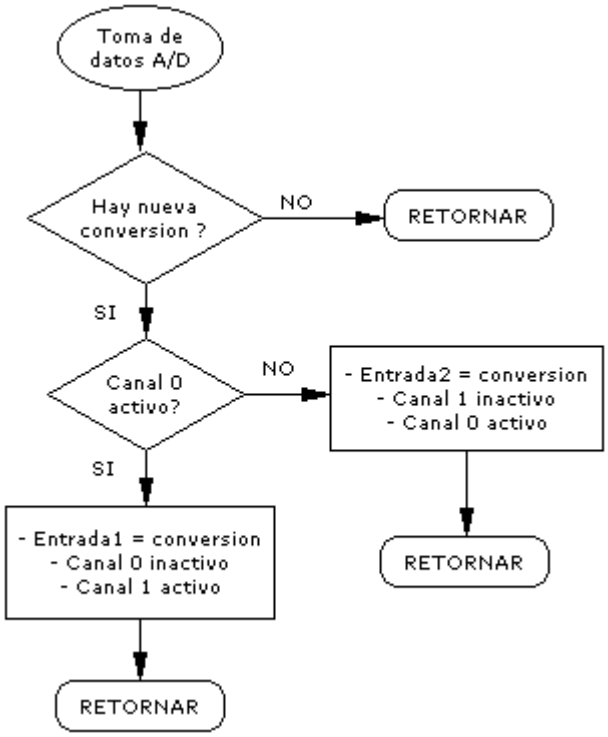


Figura 45. Diagrama de flujo conversión A / D

En el registro entrada1 se guarda la conversión del canal 0, que es la intensidad de la entrada de audio del sistema, y en el registro entrada2 se guarda la conversión del canal1, que es la intensidad de la realimentación de audio del sistema.

5.2.4. Referencia de control

Esta rutina se encarga de hallar la referencia para el controlador difuso, de la siguiente manera:

Si la entrada sobrepasa el punto de *threshold*.

Referencia = entrada1 / Ratio

La salida debe ser proporcional a la intensidad de la entrada, ya que en este punto el sistema empieza a comprimir la señal.

Si la entrada es menor del punto de *threshold*.

Referencia = entrada1

ya que en esta zona la relación es 1:1 y la entrada deber ser igual a la salida

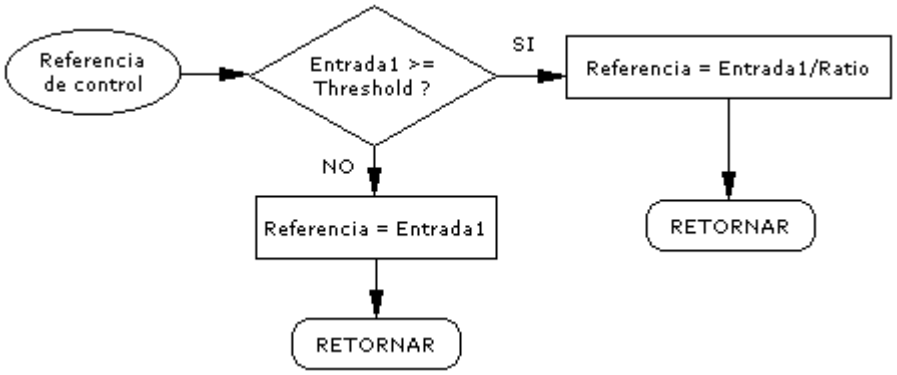


Figura 46. Diagrama de flujo para obtener la referencia

5.2.5. Error

Esta rutina halla el error y la derivada del error (de/dt), que son las entradas del controlador difuso.

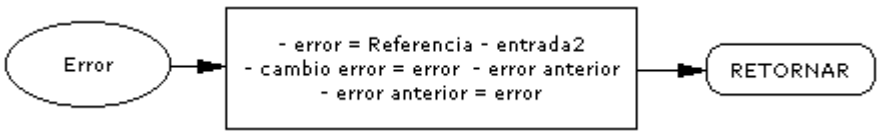


Figura 47. Diagrama de flujo para hallar el error y el cambio del error (de/dt)

Se obtienen dos registros, que son el error y el cambio en el error.

5.2.6. Fuzificación

Con las entradas del error y el cambio en el error, se determinan cuáles funciones de membresía están activas, y cuál es su certidumbre o certeza. Para esta rutina se tienen dos arreglos de vectores, son dos matrices de 2 x 5, puesto que se tienen 5 funciones de membresía para el error y para el cambio del error.

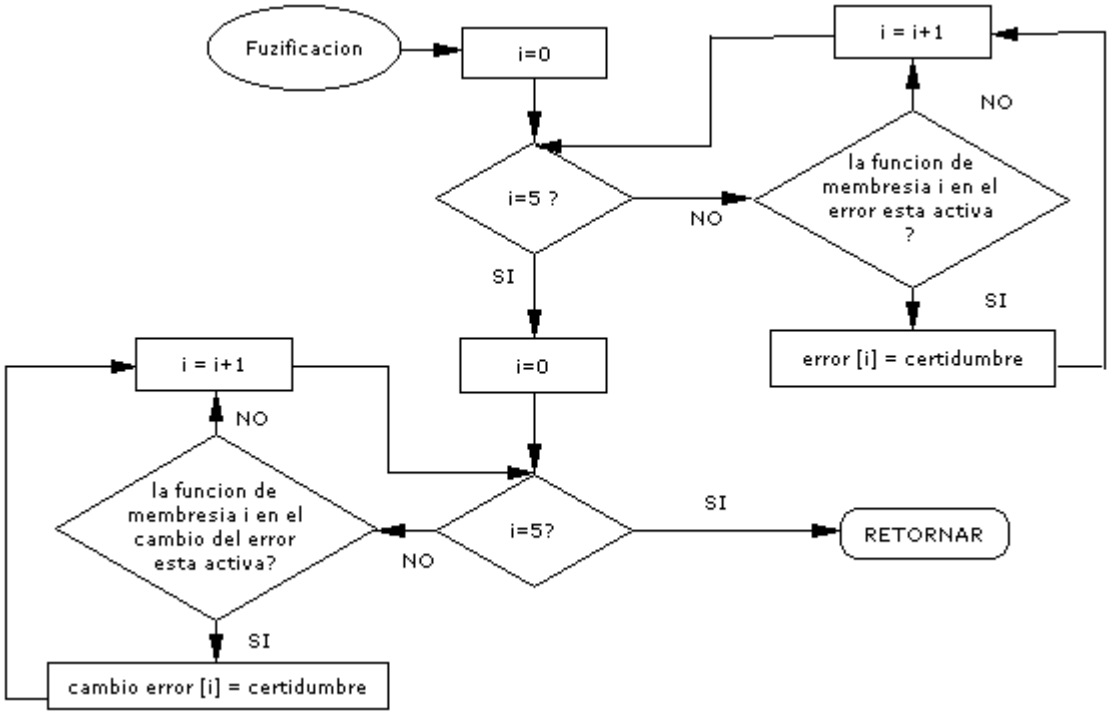


Figura 48. Diagrama de flujo fuzificación

5.2.7. Mecanismo de inferencia

Con los datos obtenidos anteriormente, es posible determinar la salida activa dependiendo de las combinaciones del error y su derivada. Esto se define en la base de reglas (la base de reglas es la que se obtuvo en la Tabla 1 en el capitulo 2). De esta manera se encuentra el centro de la función de membresía activa de la salida y su certidumbre. Esta se obtiene seleccionando el mínimo de las certidumbres entre el error y el cambio en el error.

De esta rutina se obtiene una matriz 2 x 4 donde se almacena la certidumbre de la función de salida y también el centro de la base de la función de membresía activa. Son 4, ya que ese es el número máximo de combinaciones posibles entre las funciones de las entradas.

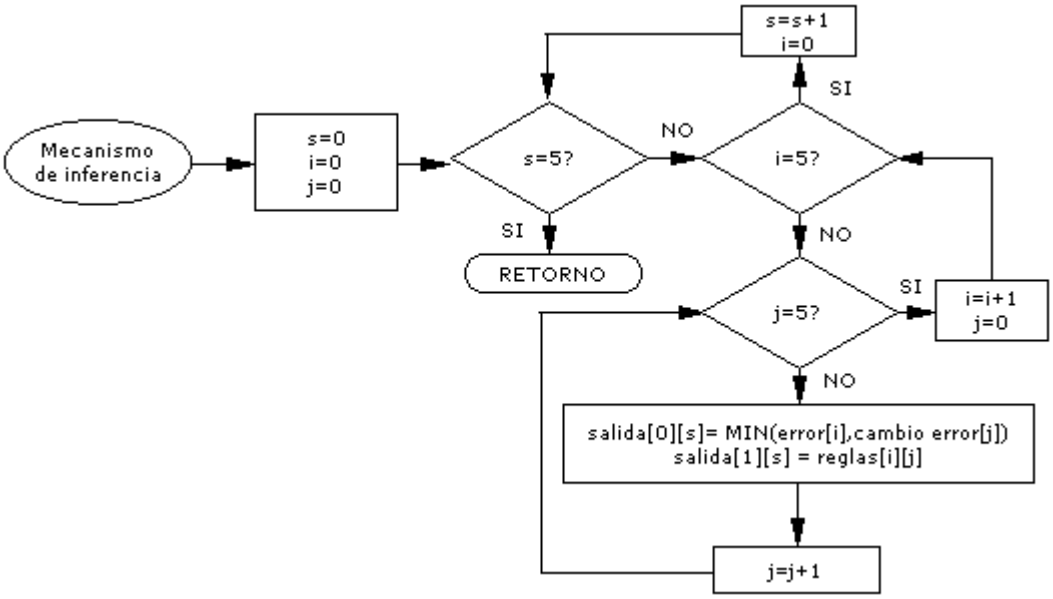


Figura 49. Diagrama de flujo del mecanismo de inferencia

5.2.8. Defuzificación

Teniendo los centros de las funciones de membresía de salida y su certidumbre, se halla el valor final del controlador difuso mediante el método del centro de gravedad (COG), descrito en el capitulo 2. De esta rutina se encuentra un registro llamado *final* el cual contiene el valor de la salida del controlador difuso.

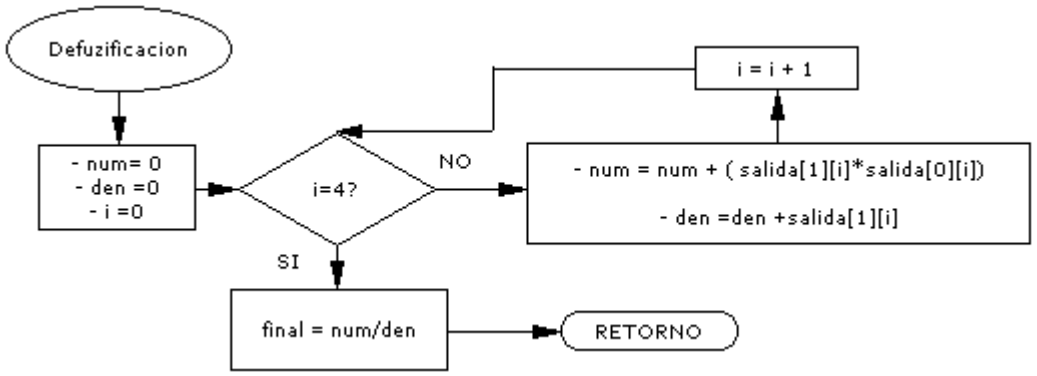


Figura 50. Diagrama de flujo de la defuzificacion

5.2.9. Actuador

Con el valor de la salida del controlador difuso se encuentra la dirección de giro del motor. Si el valor del registro final es mayor que cero, entonces el sentido de giro es positivo, y si es menor, el sentido de giro es negativo. El valor absoluto del registro final es utilizado para configurar el ancho del pulso para el modulo que genera el PWM.

Con el registro *dirección* se configura la dirección de giro del motor. Esta salida va hacia el puente H.

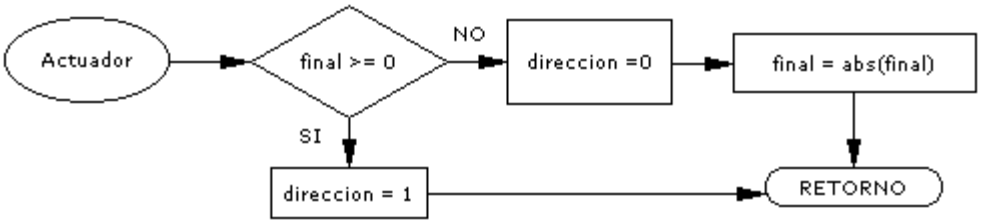


Figura 51. Diagrama de flujo del actuador

5.2.10. Interrupción del timer

La interrupción del módulo Timer, que es el generador de la onda PWM, se utiliza para configurar el nuevo valor del ancho del pulso. Esta interrupción se dispara cada vez que se termina un periodo de la onda PWM.

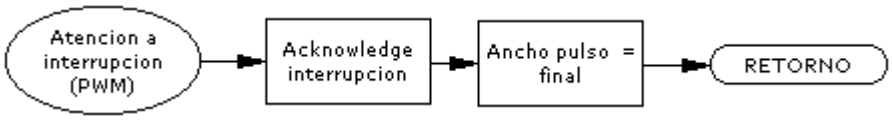


Figura 52. Diagrama de flujo de la atención a la interrupción

6. RESULTADOS

Al observar el funcionamiento del prototipo, se encontró una disminución en el rango dinámico de la señal de audio cuando esta sobrepasó el punto de *threshold*. Los niveles en la salida nunca estuvieron por encima del valor preestablecido por el usuario.

Una de las preocupaciones que existían era la capacidad mecánica del *fader* de desplazarse con suficiente rapidez para lograr mantener el nivel deseado. Se hicieron ensayos con toda clase de señales, desde las generadas con instrumentos de laboratorio hasta con música de diferentes géneros. El funcionamiento fue siempre satisfactorio aun ante cambios de gran tamaño y velocidad.

El software, según lo previsto, trabaja en tiempo real cerrando el lazo de control, desde que se hace el muestreo de la entrada hasta que se envía el comando de movimiento. Se encontró que el software tiene un buen rendimiento y requiere poca memoria para su ejecución. Esto lo hace importante para un sistema embebido de tiempo real.

En el diseño del controlador difuso para el *fader,* se decidió implementar las funciones de membresía por medio de triángulos isósceles, ya que son mas fáciles de manipular y necesitan menos tiempo de cómputo, factor que es importante teniendo en cuenta que se realizan los cálculos en un microprocesador de 8 bits. Se utilizaron 5 funciones de membresía, para el diseño del control difuso, ya que con más funciones de membresía se comprometía el control en tiempo real ya que necesitaría mas tiempo de computo. Con este diseño se obtuvo un buen desempeño de procesamiento del microprocesador.

La distorsión medida a la salida de la unidad dinámica difusa es ligeramente la misma de la entrada. Esta es, sin lugar a dudas la principal bondad del diseño.

Fue necesario que la muestra de control se obtuviera del promedio de cinco de las muestras tomadas de la señal, pues al realizar la acción de control sobre una sola muestra se observó que el comportamiento del sistema era demasiado rápido y no se podía sintonizar.

7. CONCLUSIONES

El software, diseñado e implementado, tiene un buen rendimiento y requiere poca memoria para su ejecución. Esto lo hace importante para un sistema embebido de tiempo real.

Se realizó un controlador difuso en un dispositivo de procesamiento de 8 bits, y se comprobó que esta resolución es suficiente. Esto se debe a que la digitalización de audio es sólo para efectos de lectura o monitoreo de la señal de entrada, más no para transformar la señal desde la entrada a la salida.

Si se tiene un mayor número de conjuntos difusos o funciones de membresía, se puede lograr un control más suave y más exacto sobre el proceso, pero implica más operaciones a realizar en el microcontrolador, y puede afectar de manera considerable el tiempo de ejecución del control difuso. Se debe establecer un compromiso con el fin de no afectar la operación en tiempo real.

Puesto que la señal de audio únicamente pasa por un potenciómetro de deslizamiento lineal (*fader*), ésta no se afecta por la distorsión introducida por dispositivos semiconductores ni por su ruido inherente.

Se comprueba que para aplicaciones de tiempo real en el domino del audio, no necesariamente hay que recurrir a Procesadores Digitales de Señales, DSPs. La simplicidad del concepto, pese a contener técnicas avanzadas de control inteligente, permite la utilización de microcontroladores no especializados de fácil acceso.

Una alternativa al prototipo obtenido sería la utilización de un motor paso a paso, en vez del motor CD(corriente directa), esto debido a que a bajos voltajes el motor no responde de manera adecuada y puede conllevar a vibraciones que afectan el sistema de audio. Sin embargo, se tendría que estudiar el efecto de la discretización de la posición del *fader*.

Debido a los pocos elementos de hardware necesarios para su implementación, se puede incorporar este sistema en las consolas que tengan *fader* motorizados las cuales son muy populares tanto en estudios de grabación como en sonido en vivo. De esta manera se haría una automatización "inteligente" de las mismas y se podría ofrecer en ellas funciones especializadas como las de unidades dinámicas.

El sistema puede servir de apoyo de seguridad para mantener las señales de audio en un nivel que no cause daños. Además, no importaría la potencia que maneje la señal, ya que se puede manejar con un *fader* de mayor capacidad de corriente, sin necesidad de cambiar la lógica de control.

La configuración *feedback* que provee el prototipo permite tener un buen control de la señal de audio, ya que las unidades dinámicas regulares utilizan una configuración *feedforward* por lo tanto no hay corrección en la acción tomada.

El costo de este sistema es demasiado bajo en comparación con sistemas de audio de alta calidad con bajo nivel de ruido, ya que estos utilizan semiconductores especializados y de muy alto costo.

BIBLIOGRAFÍA

AGUILAR, Luis. C++ a su alcance un enfoque orientado a objetos. Madrid, McGRAW-Hill 1995. 851p.

Doctor proaudio. Procesadores de Dinámica, [on-line] doctor proaudio, 2001 <http://www.doctorproaudio.com/doctor/temas/dinamica.htm> [consulta: 1 febrero 2003].

HABER, Rodolfo. PhD. Introducción al control borroso. Portafolio Consultores e.a.t. En: MEMORIAS SEMINARIO INTERNACIONAL SOBRE CONTROL INTELIGENTE DE PROCESOS. Ponencias de primer Seminario Internacional sobre control Inteligente. Medellín, Marzo de 1995.

Harmony central. Effects Explained: Compression/Limiting. [on-line] Harmony central, 1996 < http://www.harmony-central.com/Effects/Articles/ Compression/> [consulta: 1 febrero 2003].

KEVIN M. PASSINO HOME PAGE Kevin M. Passino. Department of Electrical Engineering Ohio State. May 4 1999. Colombus Ohio. June 25 of 1999. <http://eewww.eng.ohio-state.edu/~passino/ic_code.htm>,[Consulta 17 enero 2003].

KOSINA, Charles. Embedded multitasking. En: Circuit Cellar. Vernon, No. 127 (Feb 2001); p.1-7.

MOTOROLA. MC68HC908GP32 technical data: Hcmos Microcontroller Unit FLASH. United States of America: MOTOROLA, 2001. 395p.

PASSINO, Kevin M. and Yurkovich, Stephen. FUZZY CONTROL. Ohio, United States. Addison-Weley 1998. 475p.

TANENBAUM, Andrew S. Sistemas operativos modernos. México: Prentice-Hall Hispanoamericana, 1993. 825p.

Universidad de Antioquia. Curso académico de sistemas embebidos [on-line] Universidad de Antioquia,2003 < http://microe.udea.edu.co/cursos/ieo-944/docsmain.htm> [consulta: 18 Enero 2003].

ANEXO A
TUTORIAL BASICO CODEWARRIOR

Este es un tutorial, para aprender el manejo basico de la herramienta de desarrollo CODEWARRIOR de METROWERKS, este tutorial se encuentra en la pagina web http://microe.udea.edu.co/cursos/ieo-944/files/

UNIVERSIDAD DE ANTIOQUIA
DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA
CURSO DE SISTEMAS EMBEBIDOS
UTILIZACIÓN DE CODEWARRIOR V.2.0 CON SIMULACIÓN DE PEMICRO

Para proyectos en lenguaje c para microcontroladores HC08

Elaborado por : Jorge Andrés Baena : abaena@microe.udea.edu.co
Revisado por : Mauricio Álvarez Mesa : malvarez@udea.edu.co

Versión : 1.0 (11/06/2002)

Introducción

Metrowerks y Motorola han dejado disponible al público una edición especial del entorno de desarrollo para microcontroladores HC08: *CodeWarrior v2.0*, la cual es libre y puede compilar hasta 4Kbytes de código C. Una de las ventajas importantes de esta nueva versión es que adiciona el simulador de P&E Microcomputer Systems Inc., el cual cuenta con una máquina virtual que permite simular la CPU, periféricos e interrupciones de todos los microcontroladores HC08 actuales, lo que facilita el proceso de depuración de las aplicaciones desarrolladas en lenguaje C.

El instalador completo de CodeWarrior v2.0 se puede obtener por ftp (anónimo) desde el servidor del grupo de Microelectrónica: **ftp://microe.udea.edu.co** ingresando al directorio: **pub/embebidos/codewarrior**

Comenzando con proyectos

Este tutorial es una introducción rápida para crear proyectos en lenguaje C, utilizando el compilador Metrowerks CodeWarrior v2.0 y el simulador ICS de PEMICRO. Para el tutorial se desarrolló un programa en C para el microcontrolador HC08GP32, el código fuente esta disponible en http://microe.udea.edu.co/cursos/ieo-944/files/demoHC08

El primer paso es crear el proyecto. Para esto seleccione File | New…

Con esto aparecerá la ventana que se muestra en la figura 1. Escoja HC08 Stationery, la carpeta donde va a ubicar su proyecto y el nombre del proyecto.

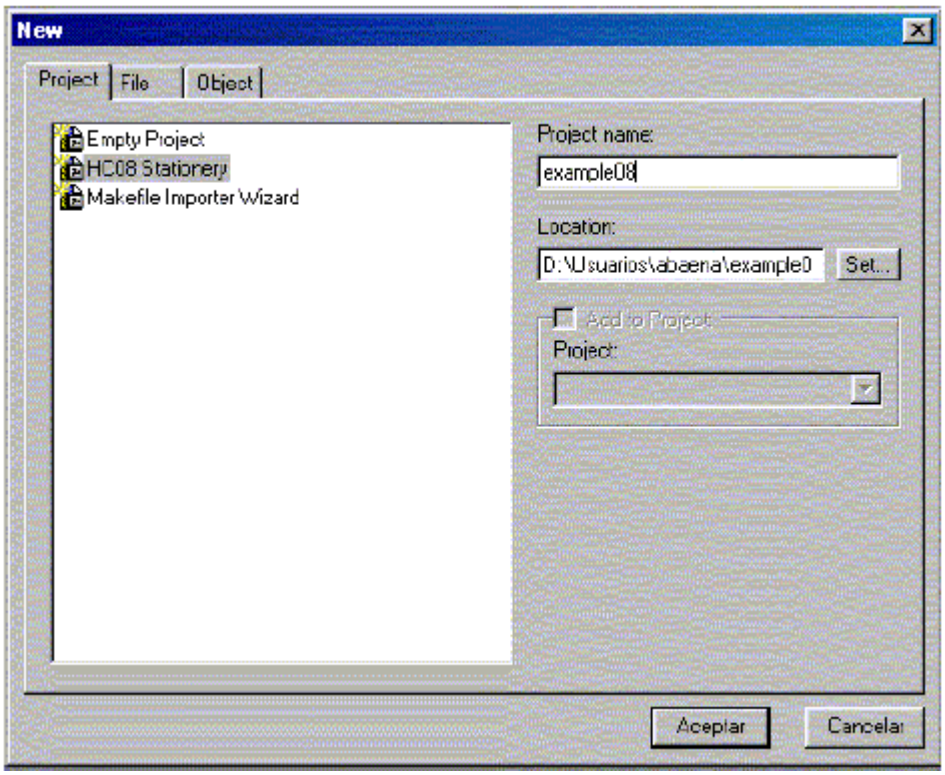


Figura 1. Crear proyecto nuevo

Cuando presione Aceptar, aparecerá otra ventana para escoger el tipo de sistema de desarrollo que va a utilizar (Stationery), como se desea utilizar la simulación de PEMICRO se escoge PEDebug y el microcontrolador correspondiente. Este modo es por defecto sólo para lenguaje ensamblador, por lo que solo aparece la opción de Asm.

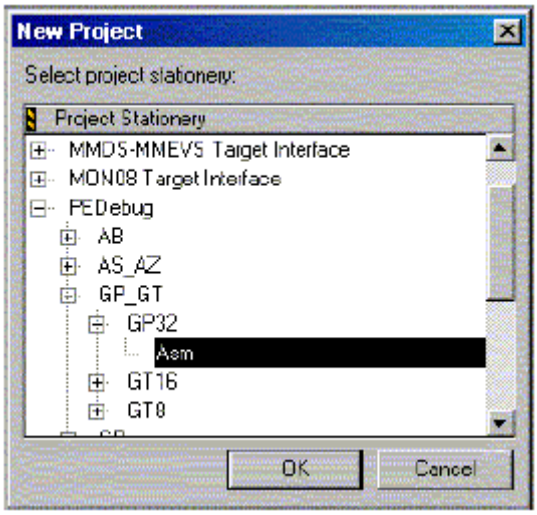


Figura 2. Nuevo Proyecto

Cuando presione OK *Codewarrior* creará una carpeta con el nombre que le dio a su proyecto y creará un proyecto de ejemplo en lenguaje ensamblador.

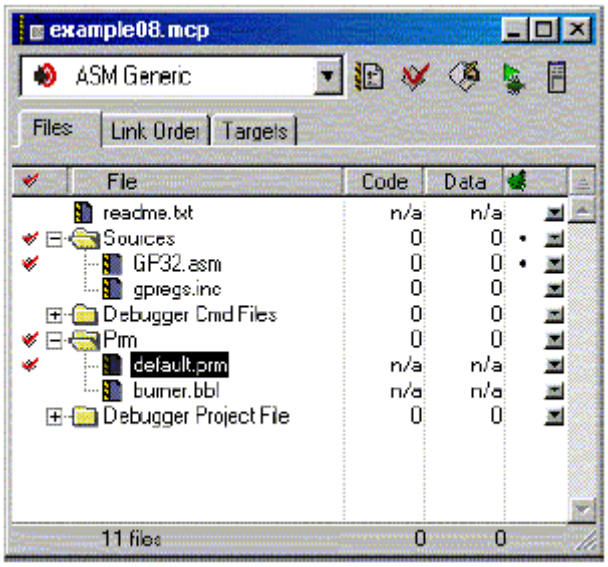


Figura 3. Proyecto generado automáticamente por CodeWarrior.

Como el proyecto a realizar es en lenguaje C, se deben remover los archivos GP32.asm y gpregs.inc del proyecto, y adicionar la librería estándar de C y los archivos fuente necesarios. Para eliminar archivos del proyecto simplemente se seleccionan y se presiona la tecla Suprimir (esto no elimina los archivos del disco duro). Para adicionar archivos al proyecto se resalta la carpeta sources y en el menú se escoge: Proyect | Add Files…

Adicionar librería y archivo de arranque para trabajar en lenguaje C

Siempre se deben agregar dos archivos obligatoriamente: La librería estándar de C y el archivo Start08.c. Estos archivos se encuentran en las siguientes rutas:

\CodeWarrior HC08_V2.0\lib\hc08c\lib\ansi.lib.

\CodeWarrior HC08_V2.0\lib\hc08c\src\Start08.c

En la figura 4 se observa como debe aparecer el proyecto después de agregar estos archivos

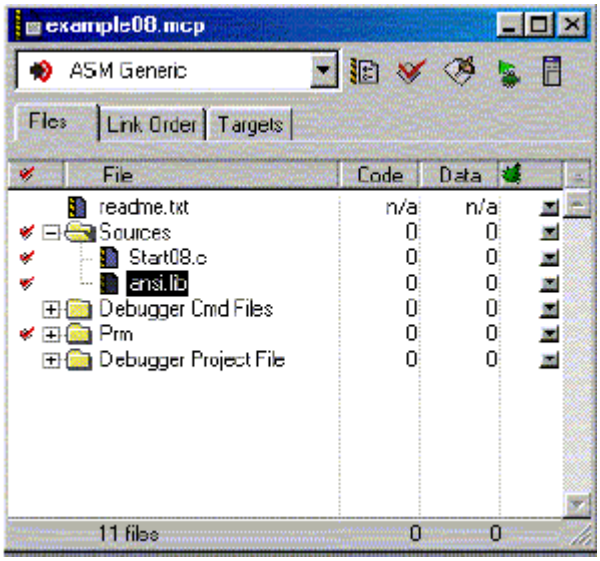


Figura 4. Archivos que se tienen que adicionar al proyecto.

Modificación del vector de interrupción de reset

Posteriormente se debe editar el archivo default.prm. Este archivo es muy importante porque define las zonas de memoria y los vectores de interrupción. En la figura 5 se indica donde esta ubicado en el proyecto. En este archivo se debe hacer lo siguiente:

Eliminar la línea : **INIT Entry**

Cambiar la línea : **VECTOR ADDRESS 0xFFFE main**

Por la línea : **VECTOR ADDRESS 0xFFFE _Startup**

_Startup es la rutina de arranque del microcontrolador, la cual está definida en **Start08.c.** La instrucción **VECTOR ADDRESS** permite definir los vectores de interrupción. Por lo tanto agregue todos los vectores de interrupción que use en su sistema. Más adelante se ilustrara con detalle la implementación de las rutinas de interrupción. En la figura 6 se muestra como debe quedar el archivo

default.prm. Si lo desea puede descargar el archivo gp32.prm (http://microe.udea.edu.co/cursos/ieo-944/files/hc08/gp32.prm) que contiene información de todos los vectores de interrupción facilitando la programación de interrupciones.

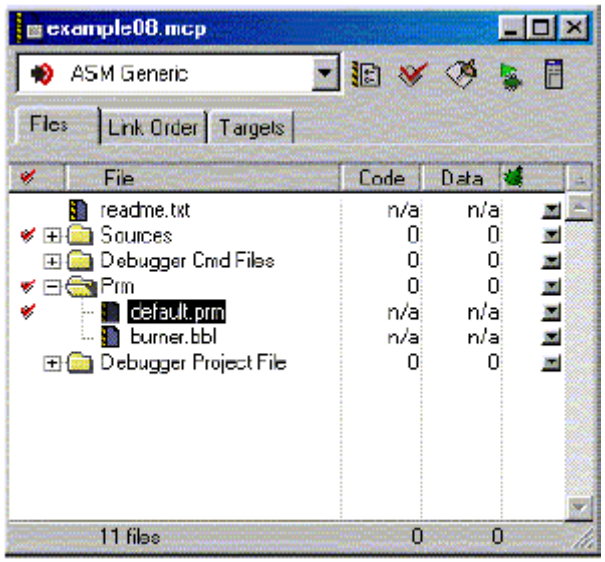


Figura 5. Localización del archivo default.prm

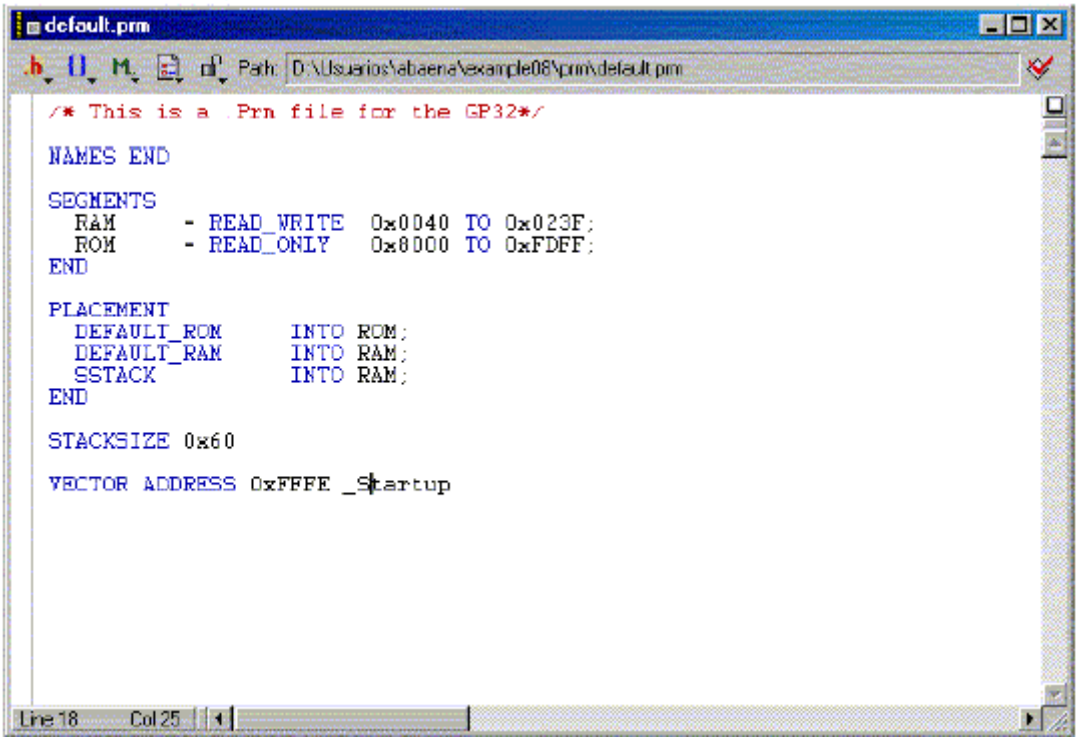


Figura 6. Archivo default.prm después de ser editado

Agregar archivos fuente al proyecto

El siguiente paso es agregar todos los archivos fuente correspondientes al proyecto. Para esto puede utilizar el menú File | New… como se muestra en la figura 7, o adicionar archivos ya existentes con Proyect | Add Files…

Como en todo programa C, debe existir una función main(). En la figura 8 se muestra una plantilla mínima para el programa.

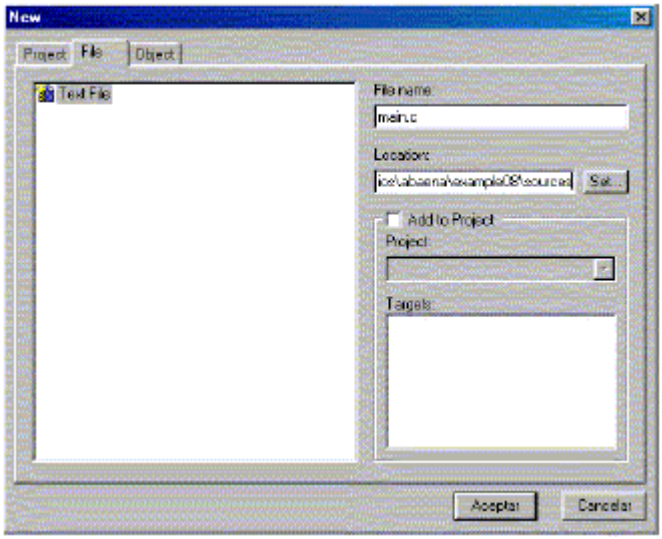


Figura 7. Crear nuevos archivos de código.

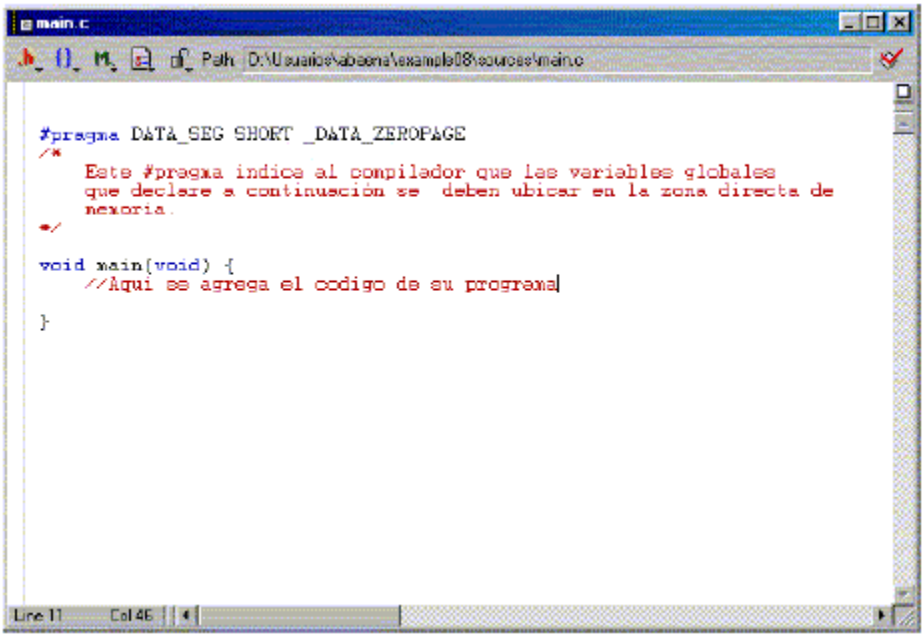


Figura 8. Función main()

Se recomienda ubicar todos los archivos fuente de un proyecto en la carpeta sources, la cual esta dentro de la carpeta del proyecto creado. Esto facilita el transporte del proyecto entre distintos computadores y evita cambios de código accidentales.

El proyecto de ejemplo es un sistema el cual prende y apaga un Led, donde los intervalos de encendido y apagado son programables por el puerto serial utilizando una aplicación tipo terminal. En la figura 9 se muestran todos los archivos fuente del proyecto en la carpeta sources y en la figura 10 el aspecto general del proyecto luego de adicionar los archivos. Estos archivos están disponibles en la página http://microe.udea.edu.co/cursos/ieo-944/files/demoHC08

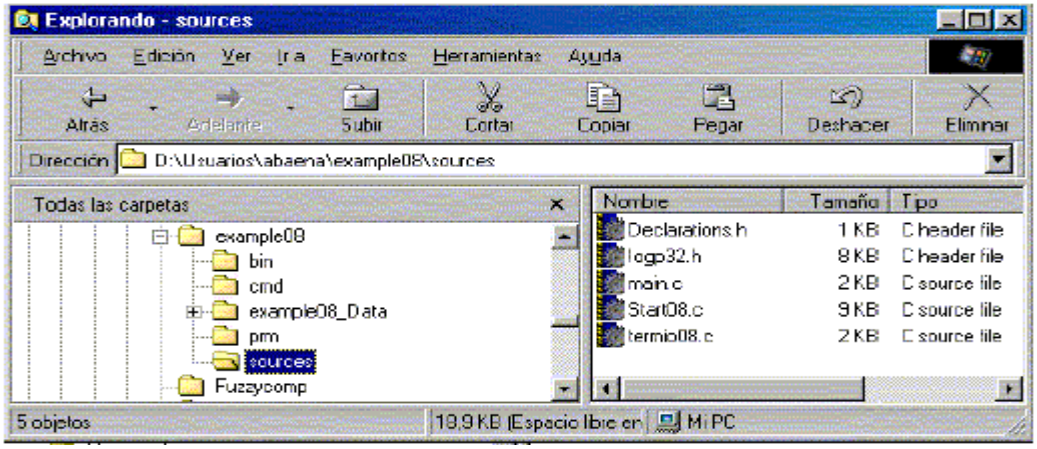


Figura 9. Archivos fuente del proyecto.

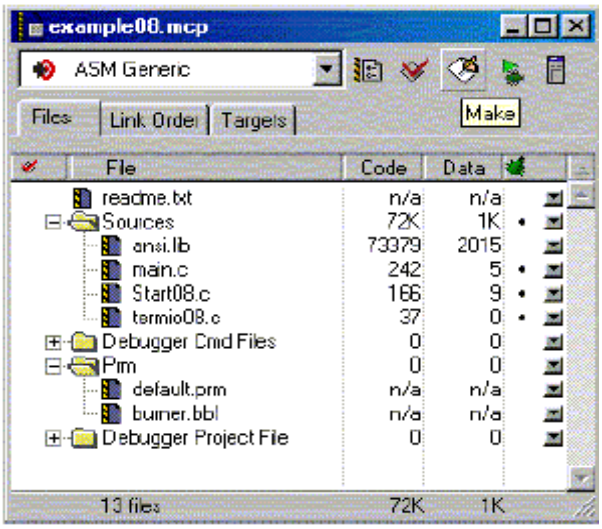


Figura 10. Archivos del proyecto de ejemplo

Un proyecto de ejemplo

En esta aplicación se utiliza una interrupción de Output Compare para medir el tiempo de encendido y apagado del LED. Para implementar la rutina de interrupción se deben realizar dos pasos.

1. Se define la rutina de interrupción como una función que no recibe parámetros y que no retorna resultados *(void)*. Justo antes de la definición de la función se debe agregar la línea

#pragma TRAP_PROC. Esto le indica al compilador que esta función es una rutina de interrupción. En la figura 11 se observa este procedimiento

```
#pragma TRAP_PROC
void OutputCompare_Isr(void){
    //Aqui va el codigo de la rutina de interrupción

}
```

Figura 11. Definición rutina de interrupción

2. El siguiente paso es agregar esta función en los vectores de interrupción, para esto en el archivo **default.prm** se agrega, por cada rutina de interrupción, la línea: **VECTOR ADDRESS [Dirección] [Nombre_función]**

Para el ejemplo anterior el archivo **default.prm** quedaría como se observa en la figura 12.

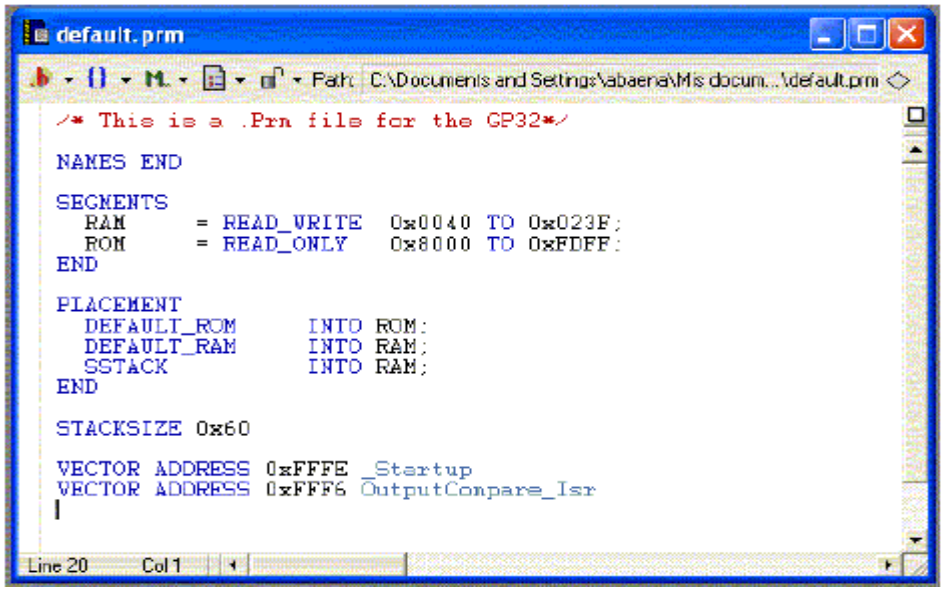


Figura 12. Definición vector de interrupciones

Para la comunicación serial con la terminal se utilizaron funciones para I/O que vienen con las librerías del *Codewarrior*. Para poder utilizar estas funciones se debe agregar al proyecto el archivo termio08.c el cual define los registros del modulo SCI y puede ser encontrado en la ruta:

\CodeWarrior HC08_V2.0\Examples\HC08\HC08 SIMULATOR\HC08 C_Calc\sources

Además, se debe asegurar que el enlazado de este archivo se de antes que el de la librería estándar ansi.lib. Para hacer esto, en la ventana Link order se arrastra con el *mouse* la librería hasta la última posición tal como se ve en la figura 13.

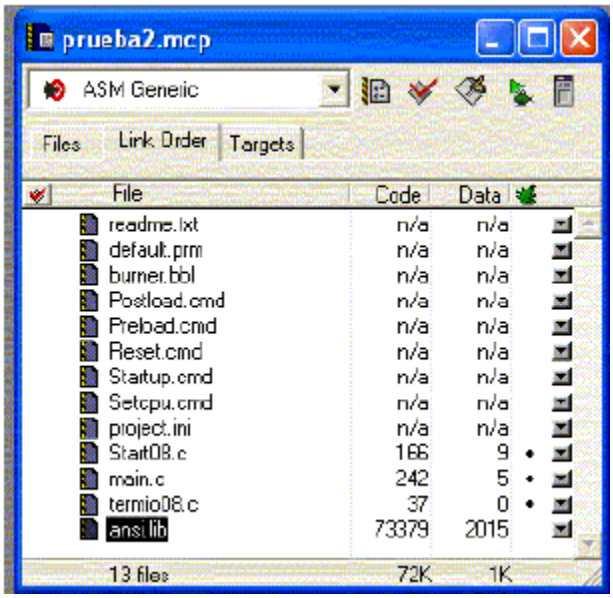


Figura 13. Orden del enlazado del proyecto

Compilación del proyecto

Después de tener todos los archivos del proyecto y actualizado los vectores de interrupción en el archivo default.prm se procede a compilar y enlazar el proyecto. Para esto seleccione

Proyect | Make.

Después de compilar los archivos CodeWarrior le dará un informe de errores, advertencias y mensajes. Si no obtiene ningún error, se puede pasar a la simulación.

NOTA: A pesar de que no haya errores de programación el enlazador muestra el error observado en la figura 14. Esto se debe a que el proyecto era inicialmente para ensamblador y no para C. Para corregir esto, se selecciona Proyect | Debug. Inmediatamente aparecerá el mensaje de la figura 15. Al presionar Yes, se recompilan todos los archivos y se abre el simulador. A partir de este momento, si se vuelve a reconstruir el proyecto con Proyect | Make, no volverá a aparecer el error mencionado.

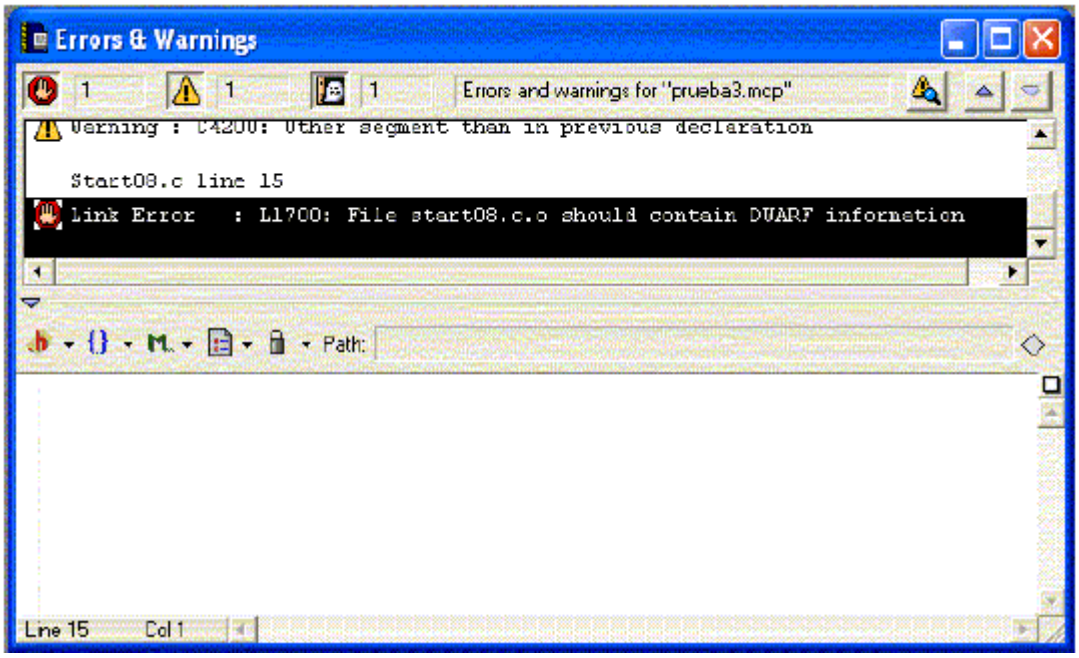


Figura 14.

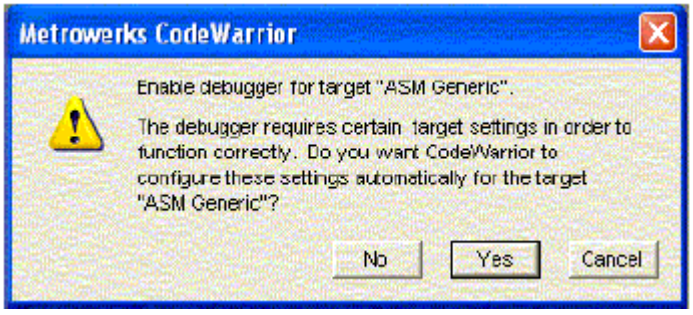


Figura 15.

Simulación del proyecto

En la figura 16, se muestra la ventana de simulación de CodeWarrior. En ella se pueden observar el contenido de los registros, memoria, variables globales y locales y el código fuente en C y en ensamblador. En la ventana Command

se pueden escribir las instrucciones de simulador de PEMICRO. Estas instrucciones también se pueden ejecutar desde el menú PEDebug.

Finalmente, solo queda grabar el programa en el microcontrolador. El archivo generado se llama AsmGeneric.S19, el cual esta ubicado en la carpeta bin del proyecto. En la figura 17 se observa el programa ya funcionando.

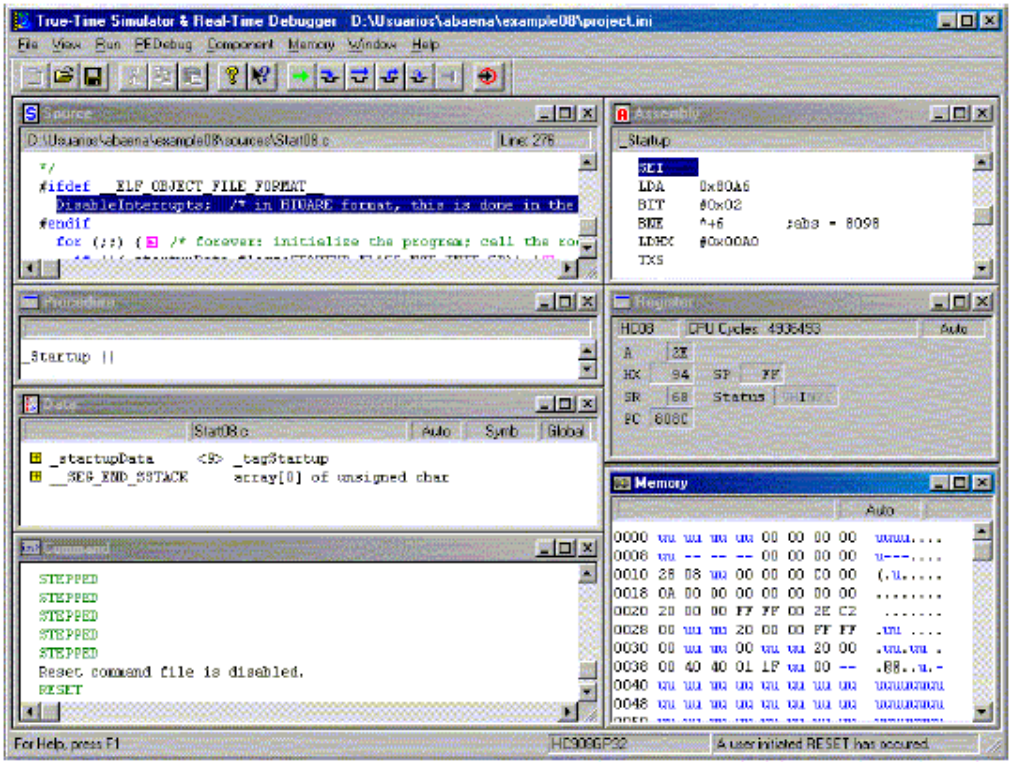


Figura 16. Sistema de simulación.

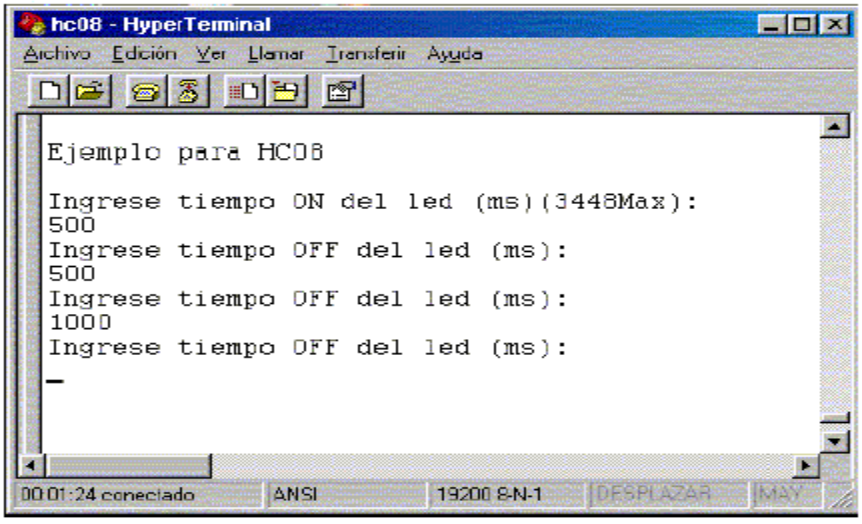


Figura 17. Programa final funcionando en el microcontrolador.

ANEXO B

ESQUEMA BASICO DEL MICROCONTROLADOR MC68HC908GP32

ANEXO C

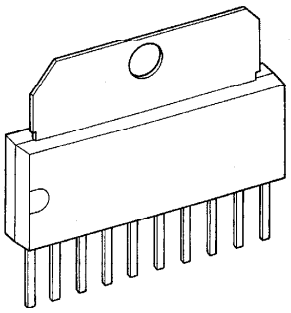
HOJA DE DATOS DEL INTEGRADO TA7288P

TOSHIBA BIPOLAR LINEAR INTEGRATED CIRCUIT SILICON MONOLITHIC

TA7288P

DUAL BRIDGE DRIVER

The TA7288P is a bridge driver that is ideal for normal / reverse switching.
This circuit offers four modes : normal rotation, reverse rotation, stop, and brake.
The output current is 1.0A (AVE.) and 2.0A (PEAK).
TA7288P has an ideal circuit configuration for VCR front tape loading and offers two types of power supply pins. One is for output, the other for control. The V_{ref} pin on the output side used to control the motor voltage facilitates motor voltage adjustment. The IC requires little input current, enabling direct connection with CMOS.



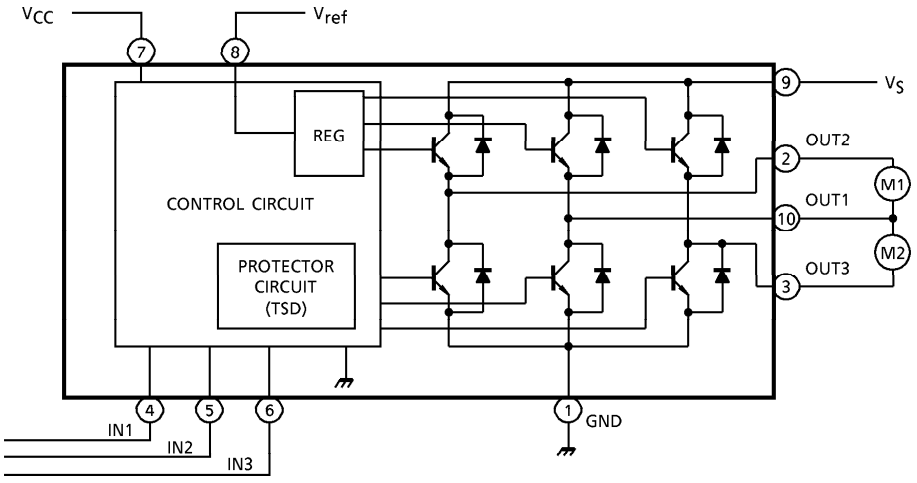
HSIP10-P-2.54

Weight : 2.47g (Typ.)

FEATURES

- 4 Modes Available (CW / CCW / STOP / BRAKE)
- Output Current Up to 1.0A (AVE.) and 2.0A (PEAK)
- Wide Range of Operating Voltage : $V_{CC\ (opr.)} = 4.5 \sim 18V$
  $V_{S\ (opr.)} = 0 \sim 18V$
  $V_{ref\ (opr.)} = 0 \sim 18V$
- Build in Thermal Shutdown, Over Current Protector and Punch-Through Current Restriction Circuit.
- Hysteresis for All Inputs.

BLOCK DIAGRAM



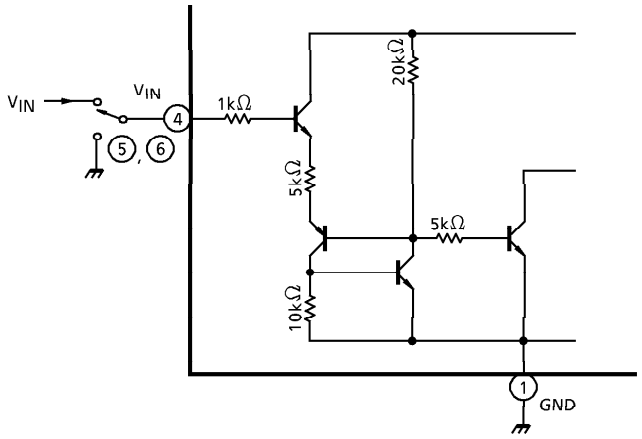
961001EBA2

● TOSHIBA is continually working to improve the quality and the reliability of its products. Nevertheless, semiconductor devices in general can malfunction or fail due to their inherent electrical sensitivity and vulnerability to physical stress. It is the responsibility of the buyer, when utilizing TOSHIBA products, to observe standards of safety, and to avoid situations in which a malfunction or failure of a TOSHIBA product could cause loss of human life, bodily injury or damage to property. In developing your designs, please ensure that TOSHIBA products are used within specified operating ranges as set forth in the most recent products specifications. Also, please keep in mind the precautions and conditions set forth in the TOSHIBA Semiconductor Reliability Handbook.

PIN FUNCTION

PIN No.	SYMBOL	FUNCTIONAL DESCRIPTION
1	GND	GND terminal
2	OUT2	Output terminal
3	OUT3	Output terminal
4	IN1	Input terminal
5	IN2	Input terminal
6	IN3	Input terminal
7	V_{CC}	Supply voltage terminal for Logic
8	V_{ref}	Supply voltage terminal for control
9	V_S	Supply voltage terminal for Motor drive
10	OUT1	Output terminal

INPUT CIRCUIT

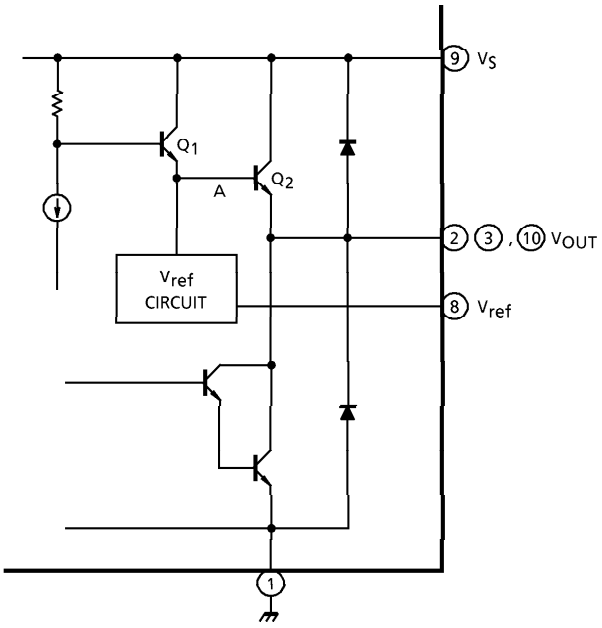


Input terminals of pin④, ⑤ and pin⑥ are all high active type and have a hysteresis of 0.7V (Typ.) 5μA type of source mode input current is required.

961001EBA2'

- The products described in this document are subject to foreign exchange and foreign trade control laws.
- The information contained herein is presented only as a guide for the applications of our products. No responsibility is assumed by TOSHIBA CORPORATION for any infringements of intellectual property or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any intellectual property or other rights of TOSHIBA CORPORATION or others.
- The information contained herein is subject to change without notice.

OUTPUT CIRCUIT



Output voltage is controlled by V_{ref} voltage. Relationship between V_{OUT} and V_{ref} is

$V_{OUT} = V_{BE}$ ($\fallingdotseq$0.7) + V_{ref}

V_{ref} terminal required to connect to V_S terminal for stable operation in case of no requirement of V_{OUT} control.

FUNCTION

INPUT			OUTPUT			MODE	
IN1	IN2	IN3	OUT1	OUT2	OUT3	M1	M2
0	0	1 / 0	L	L	L	BRAKE	BRAKE
1	0	0	H	L	∞	CW / CCW	STOP
1	0	1	L	H	∞	CCW / CW	STOP
0	1	0	H	∞	L	STOP	CW / CCW
0	1	1	L	∞	H	STOP	CCW / CW
1	1	1 / 0	L	L	L	BRAKE	BRAKE

∞ : High impedance

(Note) Inputs are all high active type.

MAXIMUM RATINGS (Ta = 25°C)

CHARACTERISTIC		SYMBOL	RATING	UNIT
Supply Voltage		V_{CC}	25	V
Motor Drive Voltage		V_S	25	V
Reference Voltage		V_{ref}	25	V
Output Current	PEAK	$I_{O\ (PEAK)}$	2.0 (Note 1)	A
	AVE.	$I_{O\ (AVE.)}$	1.0	A
Power Dissipation		P_D	12.5 (Note 2)	W
Operating Temperature		T_{opr}	−30~75	°C
Storage Temperature		T_{stg}	−55~150	°C

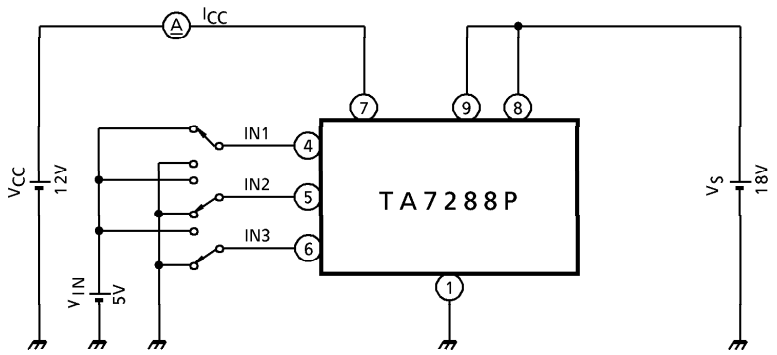
(Note 1) Duty 1 / 10, 100ms
(Note 2) Tc = 25°C

ELECTRICAL CHARACTERISTICS (Unless otherwise noted, Ta = 25°C, V_{CC} = 12V, V_S = 18V)

CHARACTERISTIC		SYMBOL	TEST CIR-CUIT	TEST CONDITION	MIN.	TYP.	MAX.	UNIT
Supply Current		I_{CC1}	1	Output OFF CW / CCW mode	—	17	30	mA
		I_{CC2}	1	Output OFF Brake mode	—	13	25	
Input Voltage	1 (High)	$V_{IN\ (H)}$	2	T_j = 25°C pin④, ⑤, ⑥	3.5	—	5.5	V
	2 (Low)	$V_{IN\ (L)}$	2	T_j = 25°C pin④, ⑤, ⑥	GND	—	0.8	
Input Current		I_{IN}	2	V_{IN} = 3.5V, Sink mode	—	5	20	μA
Input Hysteresis Voltage		ΔV_T	2	—	—	0.7	—	V
Saturation Voltage	Upper	$V_{SAT\ U-1}$	3	$V_{ref} = V_S$, V_S-V_{out}, I_O = 0.2A	—	0.9	1.2	V
	Lower	$V_{SAT\ L-1}$	3	$V_{ref} = V_S$, V_{out}-GND, I_O = 0.2A	—	1.0	1.3	V
	Upper	$V_{SAT\ U-2}$	3	$V_{ref} = V_S$, V_S-V_{out}, I_O = 1.0A	—	1.3	1.6	V
	Lower	$V_{SAT\ L-2}$	3	$V_{ref} = V_S$, V_{out}-GND, I_O = 1.0A	—	1.8	2.5	V
Output Voltage		$V_{SAT\ U-1'}$	3	V_{ref} = 10V, V_{out}-GND I_O = 0.5A	10.7	11.0	11.8	V
		$V_{SAT\ U-2'}$	3	V_{ref} = 10V, V_{out}-GND I_O = 1.0A	10.4	10.7	11.5	V
Leakage Current	Upper	$I_{L\ U}$	—	V_S = 25V	—	—	50	μA
	Lower	$I_{L\ L}$	—	V_S = 25V	—	—	50	
Diode Forward Voltage	Upper	$V_{F\ U}$	4	I_F = 1A	—	2.2	—	V
	Lower	$V_{F\ L}$	4	I_F = 1A	—	1.4	—	
Reference Current		I_{ref}	2	V_{ref} = 10V, Source mode	—	5	30	μA

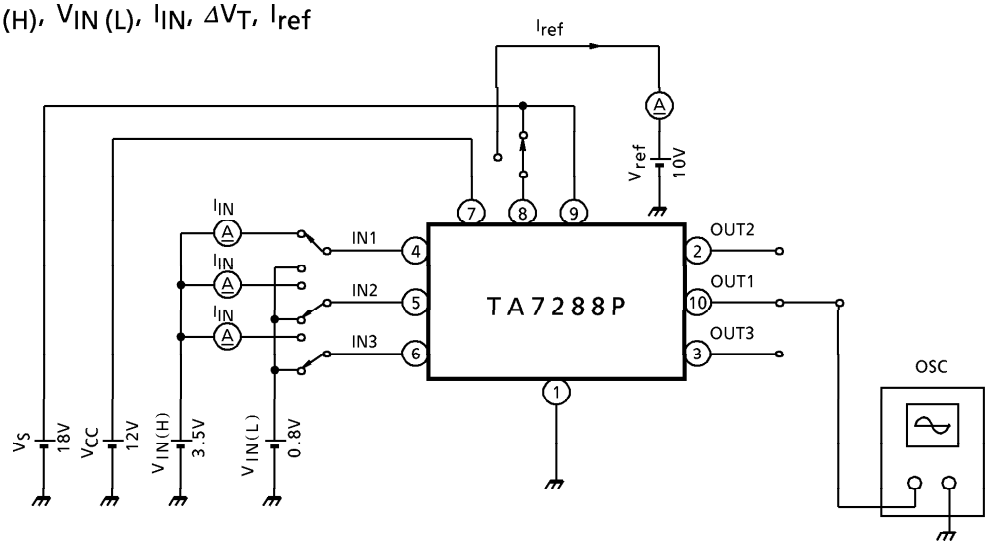
TEST CIRCUIT 1

$I_{CC1, 2}$



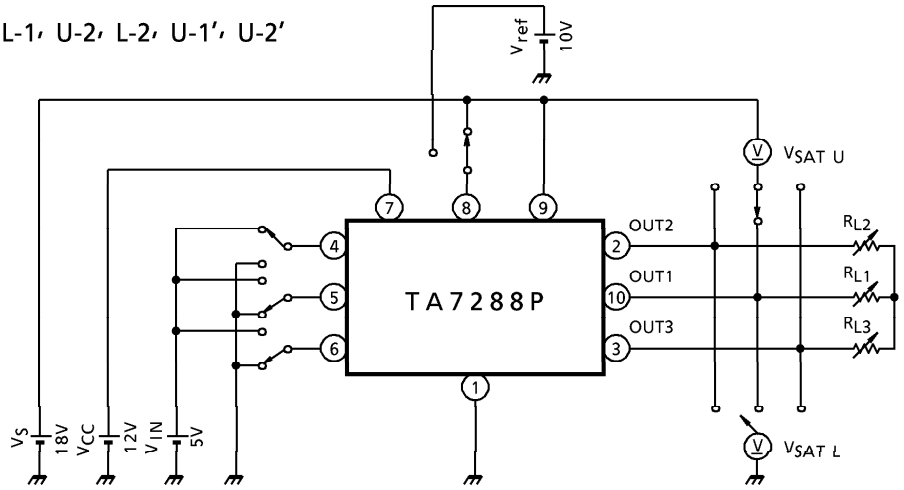
TEST CIRCUIT 2

$V_{IN\,(H)}$, $V_{IN\,(L)}$, I_{IN}, ΔV_T, I_{ref}



TEST CIRCUIT 3

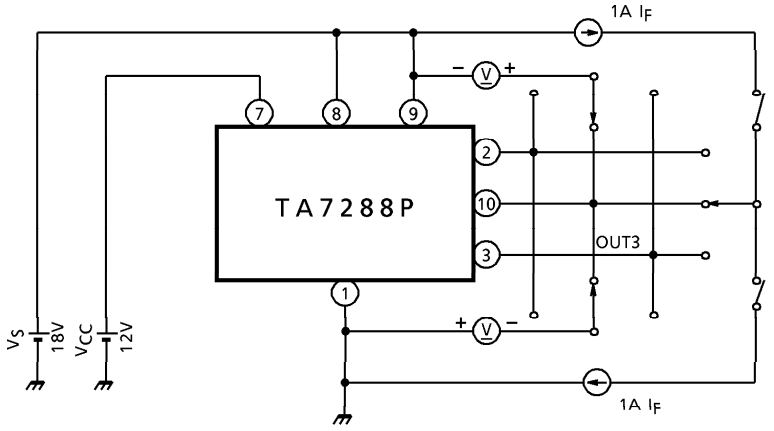
V_{SAT} U-1, L-1, U-2, L-2, U-1', U-2'



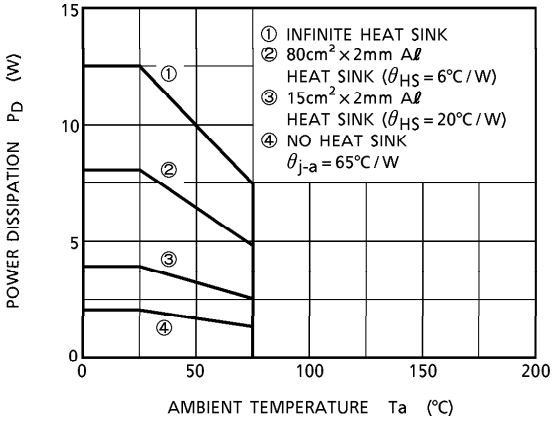
I_O calibration is required to adjust specified values of test conditions by R_{L1}~R_{L3}.

TEST CIRCUIT 4

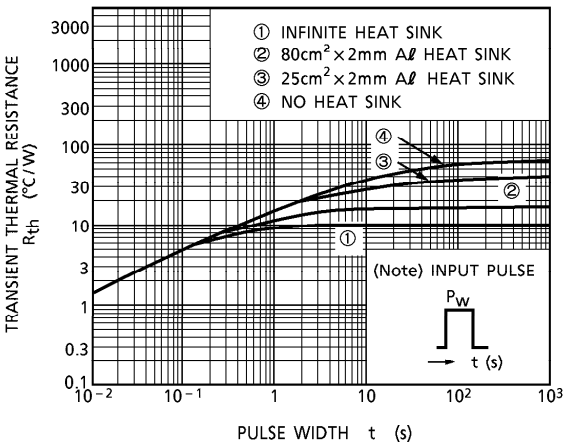
V_F U, L



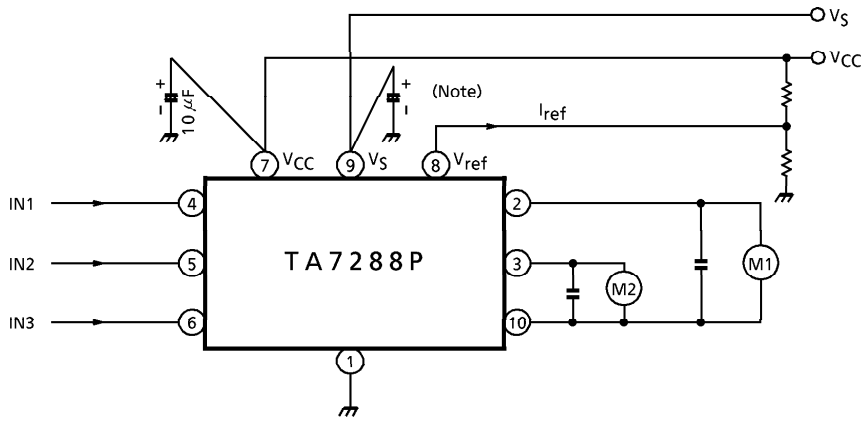
P_D – Ta



PULSE WIDTH (t) – TRANSIENT THERMAL RESISTANCE (R_{th})



APPLICATION CIRCUIT



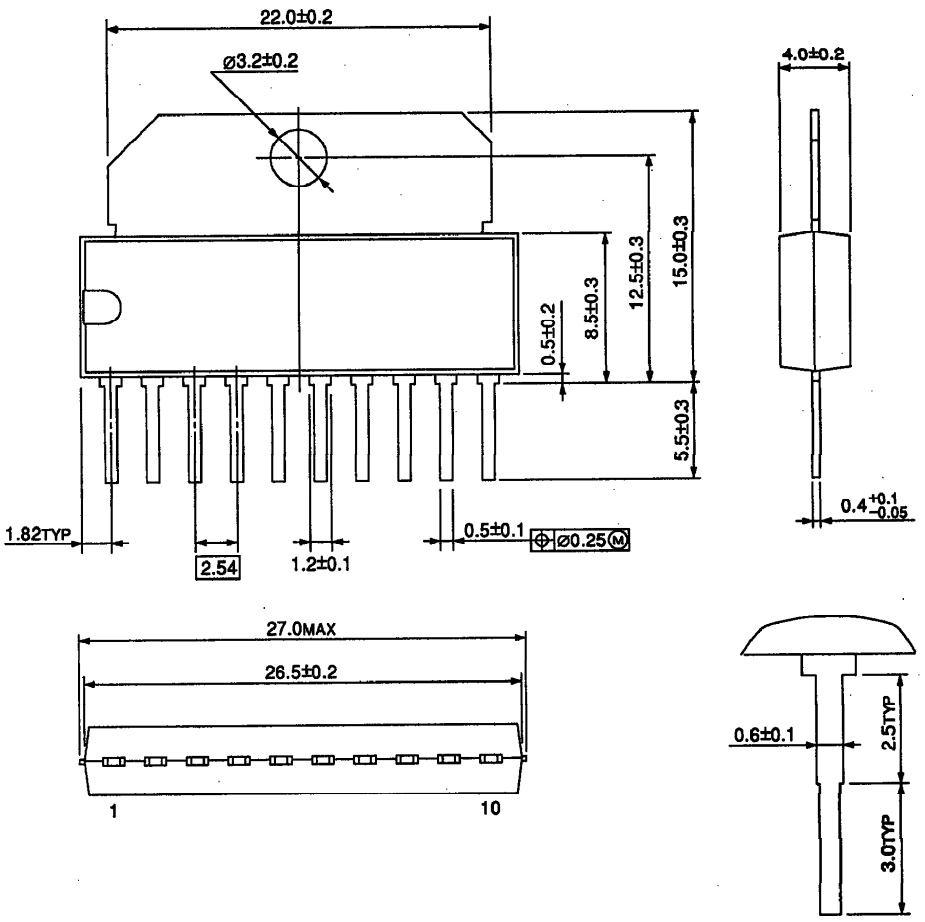
(Note 1) Connect if required

(Note 2) Utmost care is necessary in the design of the output line, V_S and GND line since IC may be destroyed due to short-circuit between outputs, air contamination fault, or fault by improper grounding.

OUTLINE DRAWING

HSIP10-P-2.54

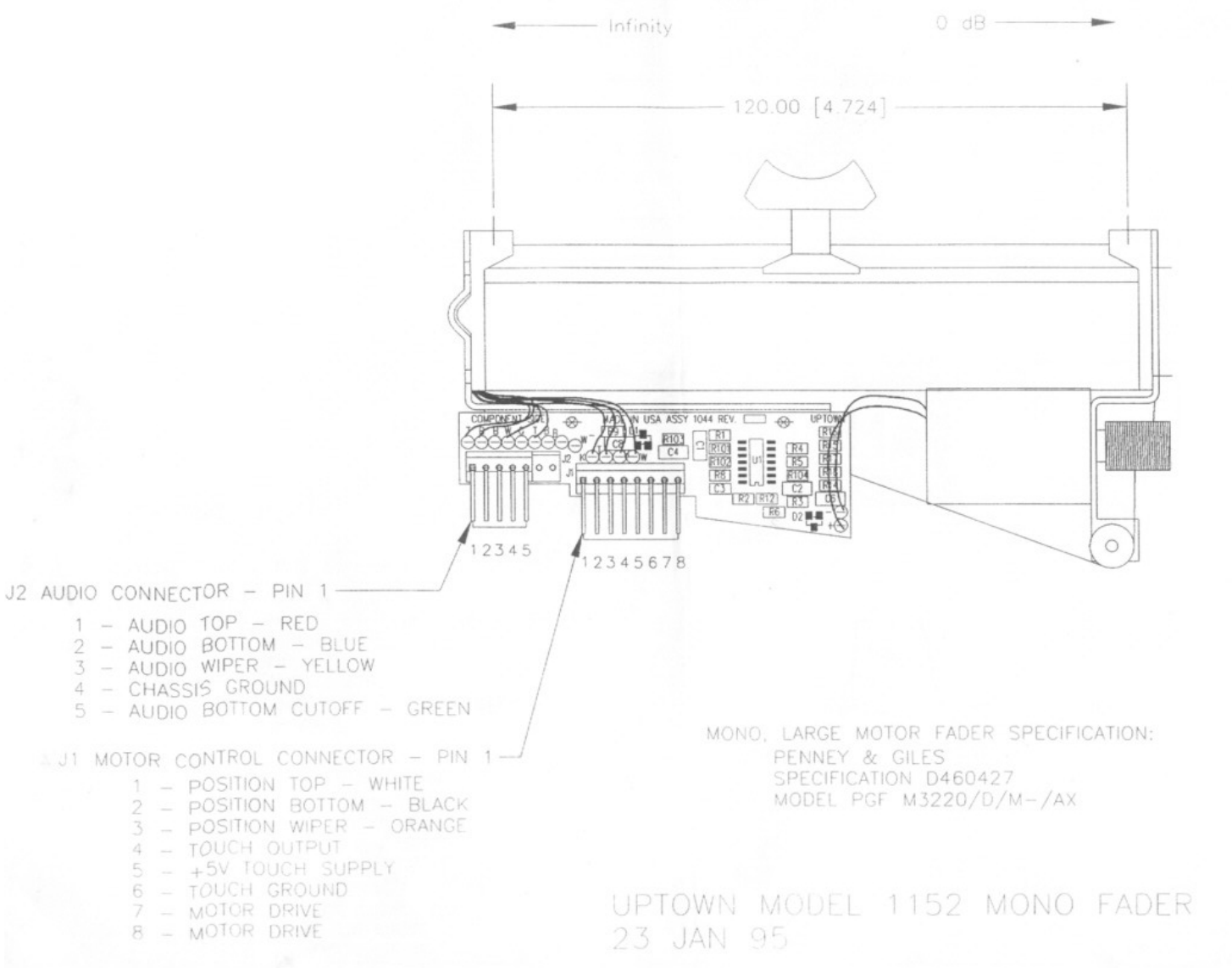
Unit : mm



Weight : 2.47g (Typ.)

ANEXO D

ESQUEMA BASICO DEL FADER D460427 Penny&Giles.


Infinity
0 dB
120.00 [4.724]
12345
12345678
J2 AUDIO CONNECTOR – PIN 1
1 – AUDIO TOP – RED
2 – AUDIO BOTTOM – BLUE
3 – AUDIO WIPER – YELLOW
4 – CHASSIS GROUND
5 – AUDIO BOTTOM CUTOFF – GREEN
J1 MOTOR CONTROL CONNECTOR – PIN 1
1 – POSITION TOP – WHITE
2 – POSITION BOTTOM – BLACK
3 – POSITION WIPER – ORANGE
4 – TOUCH OUTPUT
5 – +5V TOUCH SUPPLY
6 – TOUCH GROUND
7 – MOTOR DRIVE
8 – MOTOR DRIVE
MONO, LARGE MOTOR FADER SPECIFICATION:
PENNEY & GILES
SPECIFICATION D460427
MODEL PGF M3220/D/M-/AX
UPTOWN MODEL 1152 MONO FADER
23 JAN 95

ANEXO E

CÓDIGO EN LENGUAJE C++ DE FUZZY DINAMIC UNIT Y SUS LIBRERÍAS

```
/* ====================================================================*
 * Este programa utiliza don entradas analogo/digital y el TIM          *
 * para hacer un control con logica difusa, sobre un motor de un fader  *
 * motorizado, los canales de conversor son multiplexados en el tiempo  *
 * --------------------------------------------------------------------*
 * Archivo: tesis.c
 * Ultima modificacion:AGOSTO1 2003


 * Por: Jose Ricardo Zapata Gonzalez
 * ====================================================================*/


#pragma DATA_SEG ZERO_PAGE             // Variables definidas en la Pagina 0

/************librerias incluidas ***************/
#include "IOgp32.h"
#include <hidef.h>
#include <inout.h>
#include <stdio.h>
#include <terminal.h>
#include <termio.h>
#include <math.h>


/****************** definicion de funciones***********/

void COV_Init(void);
void error(void);
void fuzzification(void);
void inferencia(void);
void defuzzification(void);
void actuador(void);
void TERMIO_Init(void);
void PWM_Init(void);

/***************        Defines       ***************/

#define      Clr_Output_on_Compare     0x18
#define Prescaler_by_1                 0x00

const signed int v[2][5] = {{-255,-128,0,128,255},{-128,-64,0,64,128}};
                                       //matriz que dice donde esta los centros
                                       //de las funciones de membresia, donde el
                                       //primer vector es e(t) y el otro de(t)/dt
```

```
unsigned int f_ac[2][5] = {0};          //matriz con el valor de certidumbre de
                                        //cada funcion de
                                        //membresia.el error y la derivada,
const signed int v[2][5] = {{-255,-128,0,128,255},{-128,-64,0,64,128}};
                                        //matriz que dice donde esta los centros
                                        //de las funciones de membresia, donde el
                                        //primer vector es e(t) y el otro de(t)/dt

unsigned int f_ac[2][5] = {0};          //matriz con el valor de certidumbre de
                                        //cada funcion de
                                        //membresia.el error y la derivada,


signed int entrada[3] = {0};            //vector donde se almacena el valor del
                                        //error e(t) y el
                                        //cambio en el error de(t)/dt

unsigned int conv1[5]={0};              //donde guardo los valores de conv1 para
luego promediarlos
unsigned int conv2[5]={0};              //donde guardo los valores de conv2 para
luego promediarlos

unsigned int conv1pro=0;                //donde se almacena el valor promedio de
la conversion 1
unsigned int conv2pro=0;                //donde se almacena el valor promedio de
la conversion 2

unsigned int  ratio=1;                  //variable donde se almacena el ratio [1 a
                                        //10]

unsigned int referencia = 0;            //variable donde se guarda la referencia
unsigned char cual=0;                   //  para saber cual conversion esta
                                        //haciendo


unsigned char i=0,j=0,s=0;              //variables para ubicacion de los arreglos

const      signed      int      reglas[5][5]={{255,255,255,128,0},{255,255,128,0,-
128},{255,128,0,-128,-255},{128,0,-128,-255,-255},{0,-128,-255,-255,-255}};

                                        //esta es la matriz de la reglas, para la
                                        //inferencia

signed int salida[2][4]={0};            //en esta matriz se almacena el valor de
                                        //la certidumbre en
                                        //la funcion de membresia de la salida,
                                        //arriba se almacena
                                        //el valor de la certidumbre,y abajo se
                                        //almacena el centro de la funcion.

signed int final=0;                     //valor de la salida del proceso de la
                                        //logica difusa.
signed int num=0,den=0;                 //numerador y denominador


#define MODULO      0x2800              //este es el modulo para determinar la
                                        //frecuencia del PWM que es 120 Hz

unsigned int threshold=80;              //valor de threshold
unsigned int  accion=0;                 //valor donde se almacen el valor del
                                        //ancho del pulso
```

```
unsigned char direccion=1;              //esta  variable  se  hace  uno  si  la
                                        //direccion es hacia arribo si la varible
                                        //es cero la direccion es hacia abajo
unsigned char retardo=0;                //para  hacer  un  retardo  entre  accion  y
                                        //accion
unsigned char cu1=0;                    //para saber cuantas conversiones1 van

unsigned char cu2=0;                    //para saber cuantas conversiones2 van


/************************************MAIN*************************************
*/

void main (void){
      TERMIO_Init();                    //inicializacion de funciones
      COV_Init();
      PWM_Init();

      CONFIG1 = 0x01;                   //deshabilitar el  COP
      CONFIG2 = 0x01;                   //SCI usando el reloj externo, Osc En
                                        //@STOP

      DDRD = 0xFF;                      //Configuracion PORTD
                                        //como salida solo el pin 5
      PTD = 0;                          /* PORTD apagado */

      EnableInterrupts;                 //habilitar todas las interrupciones


    TERM_WriteString("\r\n Entrar el valor del threshold entre [20 240] \r\n");
    threshold=INOUT_ReadInt();
    INOUT_WriteInt(threshold);          //para verificar que valor de threshold se
                                        //entro

    TERM_WriteString("\r\n Entrar el valor del Ratio [0 1 2 3 4 5 6 7 8 9 10]
\r\n");
    ratio=INOUT_ReadInt();
    INOUT_WriteInt(ratio);


      while(1)
            {     conversion_ok();          // ir a la funcion conversion, toma
                                            //los valoresde A/D

        if (listo==1)
         {
            //se halla un promedio de los valores tomados por el conversor
             listo=0;

            if(conv1pro>=threshold)         //si la entrada sobrepasa el punto
                                            //de threshold
            {
                  referencia= ((conv1pro-threshold)/ratio)+threshold;
            }

            else
            {
                  referencia=conv1pro;

            }

                  error();
                  fuzzification();
```

```
                    inferencia();
                    defuzzification();
                  actuador();


      }
       }
}/* END main() */


/****************inicializacion del conversor analogo digital****/
void COV_Init(void){
       DDRB = 0x00;                    //puerto b como entrada del
                                       //potenciometro

       ADCLK = 0x30;                   //Configuracion del registro ADCLK
                                       //Selecion de reloj interno (4.9152MHz/4 =
                                       //1.2288 MHz)
                                       //ADC input Clock/2 = 614400 Hz

       ADSCR = 0x40;
                                       //Configuracin del ADSCR  Selecion     AD0
                                       //Canal Conversion continua activacion de
                                       //las interrupciones del ADC

//se tiene una frecuencia de muestreo de 36141.17Hz
}
/***** inicializacion del timer para el PWM*********************/
void PWM_Init(void)
{
       T2SC  = TOIE | TSTOP | TRST | Prescaler_by_1;

       T2MOD = MODULO;                         /* Configuracion del Modulo  */
       T2CH1 = 0x02;                   /* Inicializacino con output compare */

       T2SC1 = CHIE | Clr_Output_on_Compare | TOV;


    T2SC = T2SC&~TSTOP;                /* comenzar el timer del contador */

}


/******************** LOGICA DIFUSA*****************************************/

/********funcion que halla el valor del error y de/dt*****************/
void error(void)
{
                                              // valor temporal de la variable
       entrada[0]=referencia-conv2pro;        //error igual a la refencia menos
                                              //la entrada de
       entrada[1]=entrada[0]-entrada[2];      //donde se halla la de/dt
      entrada[2]=entrada[0];                  //guardar el nuevo error, para
                                              //hallar la diferencia

}



/******************** Funcion que hace la fuzzificaion******************/
/* segun el valor del error y la derivada del error se halla cual es la
certidumbre, en las
```

```
diferentes funciones de membresia activadas, entonces estos valores se ponen en
el vector
f_ac en la parte de arriba el valor de la certidumbre de cada funcion de
memebresia, si queda
en cero, es por que no se activo la funcion,*/

void fuzzification(void)
{
       unsigned int ci,cd;

       for(j=0;j<=1;j++)
       {
              for(i=0;i<=4;i++)
              {
                     f_ac[j][i]=0;             //borrar los valores anteriores que
                                               //se tenian

              }
       }


       for(j=0;j<=1;j++)
       {
              for(i=0;i<=4;i++)
              {
              if(entrada[j]<=v[j][i])
              {
                     if (entrada[j]==v[j][i])
                     {
                            f_ac[j][i]=100;           //debido a que solo hay una
funcion activa
                            i=5;
                     }

                     else
                     {
                            ci=(-100)*(entrada[j]-v[j][i]);        //se halla la
                                               //certidumbre de la funcion de
                                               //mebresia
                            ci=ci/v[j][3];
                                               //que se encuentra a la izquierda
                            cd=(100)*(entrada[j]-v[j][i-1]);
                            //se halla la certidumbre  de la funcion de menbresia

                            cd=cd/v[j][3];
                                        //que se encuentra a la derecha

                            f_ac[j][i-1]=ci;
                            //poner el valor de la certidumbre de cada funcion

       //de membresia para poder luego sacar cuales es la

       //certidumbre a usar
                            f_ac[j][i]=cd;
                            i=5;
                     }
              }
              }
       }
}

/***************Funcion que hace la inferencia segun las reglas de la logica
difusa********/
```

```
// la funcion ubica en el vector salida, los valores del minimo entre el error y
la derivada
//del error, y se pone cual es el centro de la funcion de membresia, en la parte
de abajo del
//vector de salida, entonces tenemos en la misma columna, el valor de la
certidumbre y el
//centro de la funcion de membresia.

void inferencia(void)
{
	s=0;

	for(j=0;j<=1;j++)
	{
		for(i=0;i<=4;i++)
		{
			salida[j][i]=0;
//borrar los valores anteriores del vector salida
		}
	}



	for(i=0;i<=4;i++)
//posicion del valor de el error
	{
		if(f_ac[0][i]!=0)
	//valor de ese punto en el error diferente de cero
		{
			for(j=0;j<=4;j++)
//posicion del valor de la derivada del error
			{
				if(f_ac[1][j]!=0)
//valor de ese punto en de/dt diferente a cero
				{

					salida[1][s]=reglas[i][j];
//el centro de la funcion activa de salida es la que
//esta en la tabla de reglas
					if(f_ac[0][i]<=f_ac[1][j])
	//se mira cual es la menor entre las

	//certidumbres, y se elige la menor
					{
						salida[0][s]=f_ac[0][i];
					}

					else
					{
						salida[0][s]=f_ac[1][j];
					}
					s++;
				}
			}
		}
	}
}


/************* funcion que halla el valor final de la logica******************/

void defuzzification(void)
{
```

```
	num=0;
	den=0;
	for(i=0;i<=3;i++)
	{
		num=num+salida[0][i]*salida[1][i];	//hallar los valores del
numerador
		den=den+salida[0][i];			//hallar los valores
del denominador
	}
	final=(num)/(den);				//valor final del proceso,
(crisp)
}


/****************funcion que halla el valor del ancho del pulso en el PWM
***************/

void actuador(void)
{	unsigned int temporal;


	if(final<0)
	{
		direccion=0;
		PTD = PTD&~BIT(1);
	}
	else
	{
		direccion=1;
		PTD =PTD|BIT(1);
	}
	temporal=abs(final);
//se saca la magnitud, ya que temporal en unsigned
	accion=temporal;


	accion=accion*41;
//se multiplica la salida por el valor de escala
//por que el pwm se configura en 16bit's, no en 8
	if(temporal == 0)
	{

		accion = 0x000F;		//valor minimo del PWM
	}


	if(temporal >= 250)
	{
		accion = 0x27FC;		//valor maximo del PWM
	}


	retardo=0;				//activar el retardo de 6mseg



}


/*****************************************************************************
********************** VECTORES DE INTERRUPCION ****************************
*****************************************************************/
```

```
/**************   Interrupt Vectors    **************/

interrupt 9 void TIM2_OV_ISR (void){                // Timer 2 Overflow

      static unsigned int temp=0;

      T2SC;                                    /*  Acknowledge  de  la  interrupcion
por desborddamiento del TIM */
      T2SC =T2SC& ~TOF;                        /* lectura del T2SC y escribir 0 en
TOF, borrar la bandera*/

      PTD =PTD|BIT(0) ;                              /* prender la salida */


      temp = accion;                   /*  para  evitar  OC  de  zero  2  tiempo
soncsecutivos*/
      T2CH1 = temp;

}

interrupt 8 void TIM2_CH1_ISR (void){         // Timer 2 Channel 1


      T2SC1;                             /* Acknowledge de la interrupcion   */
      T2SC1 &= ~CHF;
/* lectura del T2SC1 y escribir un 0 en  CHF */
    PTD = PTD&~BIT(0);
/* Toggle al puerto, apagar el bit 5 */


}

interrupt 16 void ADC_ISR(void)
{
      static unsigned char temporal=0;

       temporal=ADR;
//almaceno el valor del ADR en temporal

      /*ahora tomo el dato de conversion y luego multiplexo los canales*/
      if (cual==0)
       {

            conv1[cu1]=temporal;
            cual=1;
            ADSCR=0x41;
// multiplexar canal uno del conversor
            cu1=cu1+1;
//sumar 1 al contador de conversiones

       }

      else
      {
            conv2[cu2]=temporal;
//conversion2 = al el valor de la realimentacion
            ADSCR=0x40;
//multiplexar canal cero del conversor
            cual=0;
            cu2=cu2+1;
```

```
//sumar 1 al contador de conversiones

       }

        //para hallar un promedio
        if (cu1&&cu2==5)
         {
               cu1=0;                                                    //empezar
la cuenta desde cero
               cu2=0;
               listo=1;                                                 //puede   hallar
el error y su derivada
               conv1pro=(conv1[0]+conv1[1]+conv1[2]+conv1[3]+conv1[4]);
               conv1pro=conv1pro/5;


               conv2pro=(conv2[0]+conv2[1]+conv2[2]+conv2[3]+conv2[4]);
               conv2pro=conv2pro/5;
              // conv22pro=(conv2[5]+conv2[6]+conv2[7]+conv2[8]+conv2[9]);

         }

       ADSCR = ADSCR& ( ~ADCO );                              /*comenzar        nueva
conversion */


}
```

librería termio08.c

```
/****************************************************
  Demo files: Terminal port definition
 ----------------------------------------------------
   Copyright (c) HIWARE AG, Basel, Switzerland
               All rights reserved
                  Do not modify!
 ****************************************************/

/*************************************************************************
   This example shows how to access a virtual on chip IO.
   Calls for terminal output is done via on chip SCI.
*************************************************************************/

#include <hidef.h>
#include <termio.h>


typedef struct {
   unsigned char SCC1;
   unsigned char SCC2;
   unsigned char SCC3;
   unsigned char SCS1;
   unsigned char SCS2;
   unsigned char SCD;
   unsigned char SCBR;
} SCIStruct;

/* terminal.wnd */

#define SCI_ADDR 0x0013
#define NOCHAR  -1
#define  SCI (*((SCIStruct*)(SCI_ADDR)))
```

```
int TERMIO_GetBusyChar(void) {
   /* checks if there's a character from the terminal channel            */

   if (!(SCI.SCS1 & 0x20)) return NOCHAR; /* if not returns NOCHAR       */
   return SCI.SCD;                        /* if any returns the character */
 }

char TERMIO_GetChar(void) {
  /* receives character from the terminal channel */
   while (!(SCI.SCS1 & 0x20)); /* wait for input */
   return SCI.SCD;
}

void TERMIO_PutChar(char ch) {
  /* sends a character to the terminal channel */
   while (!(SCI.SCS1 & 0x80));  /* wait for output buffer empty */
   SCI.SCD = ch;
}

void TERMIO_Init(void) {
  /* initializes the communication channel */
   SCI.SCBR = 0x00;     /* baud rate 19200 at 4.9152 MHz */
   SCI.SCC1 = 0x40;
   SCI.SCC2 = 0x0C;     /* 8 bit, TE and RE set */

}
LIBRERÍA START.C

/*****************************************************************************
  FILE        : start08.c
  PURPOSE     : 68HC08 standard startup code
  LANGUAGE    : ANSI-C / INLINE ASSEMBLER
  ----------------------------------------------------------------------------
  HISTORY
    22 oct 93         Created.
    04/17/97          Also C++ constructors called in Init().
 *****************************************************************************/

#include "hidef.h"
#include "start08.h"

/*********************************************************************/
#pragma DATA_SEG FAR _STARTUP
struct _tagStartup _startupData;    /* read-only:
                                       _startupData is allocated in ROM and
                                       initialized by the linker */


#define USE_C_IMPL 0 /* for now, we are using the inline assembler implementation
for the startup code */

#if !USE_C_IMPL
#pragma MESSAGE DISABLE C20001 /* Warning C20001: Different value of stackpointer
depending on control-flow */
/* the function _COPY_L releases some bytes from the stack internally */

#ifdef __OPTIMIZE_FOR_SIZE__
#pragma NO_ENTRY
#pragma NO_EXIT
#pragma NO_FRAME
static void near loadByte(void) {
```

```
  asm {
              PSHH
              PSHX
#ifdef __HCS08__
              LDHX    5,SP
              LDA     0,X
              AIX     #1
              STHX    5,SP
#else
              LDA     5,SP
              PSHA
              LDX     7,SP
              PULH
              LDA     0,X
              AIX     #1
              STX     6,SP
              PSHH
              PULX
              STX     5,SP
#endif
              PULX
              PULH
              RTS
  }
}
#endif /* __OPTIMIZE_FOR_SIZE__ */

#endif

extern void _COPY_L(void);
/* DESC:     copy very large structures (>= 256 bytes) in 16 bit address space
(stack incl.)
   IN:      TOS count, TOS(2) @dest, H:X @src
   OUT:
   WRITTEN: X,H */


#ifdef __ELF_OBJECT_FILE_FORMAT__
#define toCopyDownBegOffs 0
#else
#define toCopyDownBegOffs 2 /* for the hiware format, the toCopyDownBeg field is
a long. Because the HC08 is big endian, we have to use an offset of 2 */
#endif
static void Init(void) {
/* purpose:     1) zero out RAM-areas where data is allocated
                2) init run-time data
                3) copy initialization data from ROM to RAM
 */
  int i;
  int *far p;
#if USE_C_IMPL   /* C implementation of ZERO OUT and COPY Down */
  int j;
  char *dst;
  _Range *far r;

  r = _startupData.pZeroOut;

  /* zero out */
  for (i=0; i != _startupData.nofZeroOuts; i++) {
    dst = r->beg;
    j = r->size;
    do {
      *dst = 0; /* zero out */
```

```
      dst++;
      j--;
    } while(j != 0);
    r++;
  }
#else /* faster and smaller asm implementation for ZERO OUT */
  asm {
ZeroOut:     ;
             LDA    _startupData.nofZeroOuts:1 ; nofZeroOuts
             INCA
             STA    i:1                         ; i is counter for number of zero
outs
             LDA    _startupData.nofZeroOuts:0 ; nofZeroOuts
             INCA
             STA    i:0
             LDHX   _startupData.pZeroOut      ; *pZeroOut
             BRA    Zero_5
Zero_3:    ;
            ; CLR    i:1 is already 0
Zero_4:    ;
             ; { HX == _pZeroOut }
             PSHX
             PSHH
             ; { nof bytes in (int)2,X }
             ; { address in (int)0,X   }
             LDA    0,X
             PSHA
             LDA    2,X
             INCA
             STA     p                      ; p:0 is used for high byte of byte
counter
             LDA    3,X
             LDX    1,X
             PULH
             INCA
             BRA    Zero_0
Zero_1:    ;
           ;  CLRA   A is already 0, so we do not have to clear it
Zero_2:    ;
             CLR    0,X
             AIX    #1
Zero_0:    ;
             DBNZA  Zero_2
Zero_6:
             DBNZ   p, Zero_1
             PULH
             PULX                           ; restore *pZeroOut
             AIX    #4                      ; advance *pZeroOut
Zero_5:    ;
             DBNZ   i:1, Zero_4
             DBNZ   i:0, Zero_3
             ;
CopyDown:    ;

  }

#endif

  /* copy down */
  /*  _startupData.toCopyDownBeg    --->    {nof(16)  dstAddr(16)  {bytes(8)}^nof}
Zero(16) */
#if USE_C_IMPL /* (optimized) C implementation of COPY DOWN */
  p = (int*far)_startupData.toCopyDownBeg;
```

```
  for (;;) {
    i = *p; /* nof */
    if (i == 0) {
      break;
    }
    dst = (char*far)p[1]; /* dstAddr */
    p+=2;
    do {
      /* p points now into 'bytes' */
      *dst = *((char*far)p); /* copy byte-wise */
      ((char*far)p)++;
      dst++;
      i--;
    } while (i!= 0);
  }
#elif defined(__OPTIMIZE_FOR_SIZE__)
  {

  asm {
#ifdef __HCS08__
              LDHX   _startupData.toCopyDownBeg:toCopyDownBegOffs
              PSHX
              PSHH
#else
              LDA    _startupData.toCopyDownBeg:(1+toCopyDownBegOffs)
              PSHA
              LDA    _startupData.toCopyDownBeg:(0+toCopyDownBegOffs)
              PSHA
#endif
Loop0:
              JSR    loadByte  ; load high byte counter
              TAX              ; save for compare
              INCA
              STA    i
              JSR    loadByte  ; load low byte counter
              INCA
              STA    i:1
              DECA
              BNE    notfinished
              CBEQX  #0, finished
notfinished:

              JSR    loadByte  ; load high byte ptr
              PSHA
              PULH
              JSR    loadByte  ; load low byte ptr
              TAX              ; HX is now destination pointer
              BRA    Loop1
Loop3:
Loop2:
              JSR    loadByte  ; load data byte
              STA    0,X
              AIX    #1
Loop1:
              DBNZ   i:1, Loop2
              DBNZ   i:0, Loop3
              BRA    Loop0

finished:
              AIS #2
    }
  }
```

```
#else /* optimized asm version. Some bytes (ca 3) larger than C version (when
considering the runtime routine too), but about 4 times faster */
  asm {
#ifdef __HCS08__
            LDHX   _startupData.toCopyDownBeg:toCopyDownBegOffs
#else
            LDX    _startupData.toCopyDownBeg:(0+toCopyDownBegOffs)
            PSHX
            PULH
            LDX    _startupData.toCopyDownBeg:(1+toCopyDownBegOffs)
#endif
next:
            LDA   0,X    ; list is terminated by 2 zero bytes
            ORA   1,X
            BEQ copydone
            PSHX         ; store current position
            PSHH
            LDA   3,X    ; psh dest low
            PSHA
            LDA   2,X    ; psh dest high
            PSHA
            LDA   1,X    ; psh cnt low
            PSHA
            LDA   0,X    ; psh cnt high
            PSHA
            AIX   #4
            JSR  _COPY_L ; copy one block
            PULH
            PULX
            TXA
            ADD   1,X    ; add low
            PSHA
            PSHH
            PULA
            ADC   0,X    ; add high
            PSHA
            PULH
            PULX
            AIX   #4
            BRA next
copydone:
  }
#endif
  /* FuncInits: for C++, this are the global constructors */
#ifdef __cplusplus
#ifdef __ELF_OBJECT_FILE_FORMAT__
  i = _startupData.nofInitBodies - 1;
  while ( i >= 0) {
    (&_startupData.initBodies->initFunc)[i]();  /* call C++ constructors */
    i--;
  }
#else
  if (_startupData.mInits != NULL) {
    _PFunc *fktPtr;
    fktPtr = _startupData.mInits;
    while(*fktPtr != NULL) {
      (**fktPtr)(); /* call constructor */
      fktPtr++;
    }
  }
```

```
#endif
#endif
  /* LibInits: used only for ROM libraries */
}
#pragma NO_EXIT
#ifdef __cplusplus
  extern "C"
#endif
void _Startup (void) { /* To set in the linker parameter file: 'VECTOR 0
_Startup' */
/* purpose:    1) initialize the stack
               2) initialize run-time, ...
                  initialize the RAM, copy down init dat etc (Init)
               3) call main;
    called from: _PRESTART-code generated by the Linker*/
#ifdef __ELF_OBJECT_FILE_FORMAT__
  DisableInterrupts;  /* in HIWARE format, this is done in the prestart code */
#endif
  for (;;) { /* forever: initialize the program; call the root-procedure */
    if (!(_startupData.flags&STARTUP_FLAGS_NOT_INIT_SP)) {
      /* initialize the stack pointer */
      INIT_SP_FROM_STARTUP_DESC();
    }
    Init();
    (*_startupData.main)();
  } /* end loop forever */
}
```

LIBRERÍA INOUT.C

```
/*****************************************************
   inout.c - Terminal: basic I/O functions with terminalwe
 -----------------------------------------------------
   Copyright (c) Metrowerks, Basel, Switzerland
               All rights reserved
                  Do not modify!
 *****************************************************/
#include <hidef.h>
#include <inout.h>
#include <terminal.h>

#if defined(__HIWARE__) && !defined(__NO_RECURSION__)
#pragma MESSAGE DISABLE C1855 /* WARNING : C1855: Recursive function call */
#endif

static int termCh;

int INOUT_ReadInt(void) {
  /* reads an integer value from the terminal */
  Bool neg = FALSE;
  int val = 0;

  termCh = (int)TERM_Read();
  if (termCh == '-') {
    neg = TRUE; TERM_Write((char)termCh); termCh = (int)TERM_Read();
  }
  while (termCh >= '0' && termCh <= '9') {
    TERM_Write((char)termCh);
    val = 10*val + termCh - (int)'0';
    termCh = (int)TERM_Read();
  }
  if (neg) val = -val;
  return val;
```

```
}

int INOUT_TermChar(void) {
  /* returns the last character read from the terminal */
  return termCh;
}

#if defined(__NO_RECURSION__)

static void NextDigit(int val) {
  char buf[5];
  unsigned char i;

  i = sizeof(buf);
  while (val > 0) {
    i--;
    buf[i] = val % 10;
    val = val / 10;
    if (i == 0) break;
  }
  while(i < sizeof(buf)) {
    TERM_Write(buf[i] + '0');
    i++;
  }
}

#else

static void NextDigit(int val) {
  if (val > 0) {
    NextDigit(val / 10);
    TERM_Write((char)(val % 10 + 48));
  }
}
#endif

void INOUT_WriteInt(int val) {
  /* writes an integer value to the terminal */
  if (val == 0) {
    TERM_Write('0'); return;
  }
  if (val < 0) {
    TERM_Write('-'); val = - val;
  }
  NextDigit(val);
}

#if defined(__NO_RECURSION__)

static void NextDigitLong(long val) {
  char buf[10];
  unsigned char i;

  i = sizeof(buf);
  while (val > 0) {
    i--;
    buf[i] = val % 10L;
    val = val / 10L;
    if (i == 0) break;
  }
  while(i < sizeof(buf)) {
    TERM_Write(buf[i] + '0');
    i++;
```

```
  }
}

#else

static void NextDigitLong(long val) {
  if (val > 0) {
    NextDigitLong(val / 10);
    TERM_Write((char)(val % 10 + 48));
  }
}
#endif

void INOUT_WriteLong(long val) {
  /* writes a long value to the terminal */
  if (val == 0) {
    TERM_Write('0'); return;
  }
  if (val < 0)  {
    TERM_Write('-'); val = - val;
  }
  NextDigitLong(val);
}
```

LIBRERÍA TERMINAL.C

```
/*****************************************************
   terminal.c - Terminal: terminal functions
 -----------------------------------------------------
   Copyright (c) Metrowerks, Basel, Switzerland
               All rights reserved
                  Do not modify!
 *****************************************************/
#include <hidef.h>
#include <terminal.h>
#include <termio.h>


#define BS   '\010'
#define DEL  '\177'
#define CR   '\015'
#define LF   '\012'
#define ESC  '\033'

char TERM_Read(void) {
  /* reads a character from the terminal */
  return TERMIO_GetChar();
}

void TERM_ReadString(LIBDEF_StringPtr str, int maxLen) {
  /* reads a string int buffer 'str' with maxLen */
  unsigned char ch = TERM_Read();
  int len = 0;

  while (ch >= ' ' || ch == BS || ch == DEL) {
    if (ch >= ' ') {
      if ( len < maxLen-1) {
        str[len++] = ch; TERM_Write(ch);
      }
```

```
    } else if (len > 0) {
      TERM_Write(ch); len--;
    }
    ch = TERM_Read();
  }
  str[len] = 0;
}

void TERM_Write(char ch) {
  /* writes a character to the terminal */
  TERMIO_PutChar(ch);
}

void TERM_WriteLn(void) {
  /* writes a new-line to the terminal */
  TERMIO_PutChar(CR); TERMIO_PutChar(LF);
}

void TERM_WriteString(LIBDEF_ConstStringPtr str) {
#ifndef _TERMIO_HAS_PUT_STRING_
  /* writes a zero terminated string to the terminal */
  while (*str) {
    TERMIO_PutChar(*str);
    str++;
  }
#else
  TERMIO_PutString(str);
#endif

}

void TERM_Direct(TERM_DirectKind what, LIBDEF_ConstStringPtr fileName) {
  /* sets direction of the terminal */
  if (what < TERM_TO_WINDOW || what > TERM_APPEND_FILE) return;
  TERM_Write(ESC); TERM_Write('h');
  TERM_Write((char)(what + '0'));
  if (what != TERM_TO_WINDOW && what != TERM_FROM_KEYS) {
    TERM_WriteString(fileName); TERM_Write(CR);
  }
}
```

LIBRERÍA TERMIO.C

```
/*****************************************************
  Demo files: Terminal port defintion
 -----------------------------------------------------
   Copyright (c) Metrowerks, Basel, Switzerland
               All rights reserved
                  Do not modify!
 *****************************************************/

/*************************************************************************
   This example shows how to access a virtual on chip IO.
   Calls for terminal output is done via on chip SCI.
*************************************************************************/

#include <termio.h>

#ifdef __ELF_OBJECT_FILE_FORMAT__
  #define HIWAVE  1  /* only HI-WAVE can handle ELF/Dwarf object files */
#elif defined(__H8_500__)
  #define HIWAVE  0  /* H8/500 simulator not available in HI-WAVE yet */
#else
  #define  HIWAVE    1    /*  set  to  one  for  HI-WAVE,  to  zero  for  HI-CROSS
Simulators/Debuggers */
#endif

#if defined(__HC16__) && (defined(__SMALL__) || defined(__MEDIUM__))
  /* NOTE: For the HC16 CPU, this module must be compiled with command line
           option "-Cp0" telling the compiler that the data page will be page 0,
           thus enabling it to generate efficient access code for the on-chip
           I/O. */
  #pragma OPTION ADD "-Cp0"
#endif

/*  if  the  testterm.wnd  terminal  emulation  is  used.  Currently  only  the  HC12
supports
   the on-chip terminal emulation (terminal.wnd) with HI-WAVE */
#define  TEST_TERM    ((HIWAVE  &&  (!defined(__HC12__))  &&  (!defined(__XA__)))  ||
defined(__HC11__))

typedef struct {
  #if TEST_TERM
    unsigned char BAUD;
    unsigned char SCCR1;
    unsigned char SCCR2;
    unsigned char SCSR;
    unsigned char SCDR;
  #elif defined(__HC12__)
    unsigned char SCxBDH;
    unsigned char SCxBDL;
    unsigned char SCxCR1;
    unsigned char SCxCR2;
    unsigned char SCxSR1;
    unsigned char SCxSR2;
    unsigned char SCxDRH;
    unsigned char SCxDRL;
  #elif (defined(__HC16__) || defined(__M68K__))
    unsigned int SCCR0;
    unsigned int SCCR1;
    unsigned int SCSR;
    unsigned int SCDR;
  #elif defined(__H8_500__)  /* H8/532 serial communication unit */
    unsigned char SMR;    /* Serial Mode Register   */
    unsigned char BRR;    /* Bit Rate Register      */
```

```
    unsigned char SCR;    /* Serial Control Register*/
    unsigned char TxBuf;  /* Transmit Data Register */
    unsigned char SSR;    /* Serial Status Register */
    unsigned char RxBuf;  /* Receive Data Register  */
  #elif defined(__XA__)
    unsigned char S0CON;
    unsigned char S0STAT;
    unsigned char _filler[62];
    unsigned char S0BUF;
  #else
    #error "unknown architecture"
  #endif
} SCIStruct;

#if HIWAVE
  #if defined(__HC12__) /* terminal.wnd */
    #define SCI_ADDR 0x00c0
  #elif defined(__XA__) /* terminal.wnd */
    #define SCI_ADDR 0x420
  #else   /* testterm.wnd */
    #define SCI_ADDR 0x200
  #endif
#else
  #if defined(__HC11__)
    #define SCI_ADDR 0x102B
  #elif defined(__HC12__)
    #define SCI_ADDR 0x00c0
  #elif defined(__HC16__)
    #define SCI_ADDR 0xFFC08
  #elif defined(__M68K__)
    #define SCI_ADDR 0xFFFC08
  #elif defined(__H8_500__)
    #define SCI_ADDR 0xFFD8
  #elif defined(__XA__)
    #define SCI_ADDR 0x420
  #endif
#endif

#if 1 || __STDC__  /* ANSI does not allow the '@' */
  #if   defined(__HC05__)   ||   defined(__HC08__)   ||   defined(__ST7__)   ||
defined(__ST19X__)
    #define  SCI (*((SCIStruct* far)(SCI_ADDR)))
  #else
    #define  SCI (*((SCIStruct*)(SCI_ADDR)))
  #endif
#else  /* use the language extension '@': easier debugging! */
  #ifdef __XA__
    #pragma DATA_SEG DIRECT sfr
  #endif
  SCIStruct SCI @SCI_ADDR;
  #ifdef __XA__
    #pragma DATA_SEG DEFAULT
  #endif
#endif

char TERMIO_GetChar(void) {
  /* receives character from the terminal channel */
  #if TEST_TERM
    while (!(SCI.SCSR & 0x20)){}; /* wait for input */
    /* NOTE: when using simulator you may want to open the 'TestTerm' component
*/
    return SCI.SCDR;
  #elif defined(__HC12__)
```

```
    while (!(SCI.SCxSR1 & 0x20)){}; /* wait for input */
    /* NOTE: when using simulator you may want to open the 'Terminal' component
*/
    return SCI.SCxDRL;
  #elif (defined(__HC16__) || defined(__M68K__))
    while (!(SCI.SCSR & 0x40)){}; /* wait for input */
    return SCI.SCDR;
  #elif defined(__H8_500__)
    char ch;

    while (!(SCI.SSR & 0x40)) {}; /* wait for receive data register full */
    ch = SCI.RxBuf;
    SCI.SSR &= 0xBF;  /* clear RDRF bit */
    return ch;
  #elif defined(__XA__)
    while (!(SCI.S0CON & 0x01)){}; /* wait for input */
    SCI.S0CON &= 0xFE;
    return SCI.S0BUF;
  #else
    #error "unknown architecture"
  #endif
}

void TERMIO_PutChar(char ch) {
  /* sends a character to the terminal channel */
  #if TEST_TERM
    while (!(SCI.SCSR & 0x80)) {};    /* wait for output buffer empty */
    SCI.SCDR = ch;
  #elif defined(__HC12__)
    while (!(SCI.SCxSR1 & 0x80)){};  /* wait for output buffer empty */
    SCI.SCxDRL = ch;
  #elif (defined(__HC16__) || defined(__M68K__))
    while (!(SCI.SCSR & 0x100)){};    /* wait for output buffer empty */
    SCI.SCDR = ch;
  #elif defined(__H8_500__)
    while (!(SCI.SSR & 0x80)){}; /* wait for transmit data register empty */
    SCI.TxBuf = ch;
    SCI.SSR &= 0x7F;  /* clear TDRE bit */
  #elif defined(__XA__)
    while (!(SCI.S0CON & 0x02)){};  /* wait for output buffer empty */
    SCI.S0CON &= 0xFD;
    SCI.S0BUF = ch;
  #else
    #error "unknown architecture"
  #endif
}

void TERMIO_Init(void) {
  /* initializes the communication channel */
  #if TEST_TERM
    SCI.BAUD  = 0x30;     /* baud rate 9600 at 8 MHz */
    SCI.SCCR2 = 0x0C;     /* 8 bit, TE and RE set */
    SCI.SCSR |= 0x80;     /* output buffer empty */
  #elif defined(__HC12__)
    #if 0
      SCI.SCxBDL  =   52;   /* baud rate 9600 at 8 MHz */
    #else
      SCI.SCxBDL   =    1;     /* for the simulator we use this small divisor to
speed up the output. On hardware please use the real value. */
    #endif
    SCI.SCxCR2 = 0x0C;    /* 8 bit, TE and RE set */
  #elif (defined(__HC16__) || defined(__M68K__))
    SCI.SCCR0 = 55;       /**** Select baud rate 9600 */
```

```
    SCI.SCCR1 = 0x000C;   /**** 8 bit, TE and RE set */
  #elif defined(__H8_500__)
    SCI.SMR = 8;      /* 00001000: 8 bits per character, no parity, 2 stop bit,
access to buffer and IER */
    SCI.BRR = 32;   /* Baud rate 9600 */
    SCI.SCR = 0x30; /* 00110000: disable Tx and Rx interrupt */
  #elif defined(__XA__)
    SCI.S0CON  = 0x12; /* UART0 mode 0 (shift register): Baud rate = osc / 16 ;
reception enabled */
    SCI.S0STAT = 0x00; /* NB: XA SFRs IO simulation simulates only this mode. */
  #else
    #error "unknown architecture"
  #endif
}
```

ANEXO F

ARTICULO PUBLICADO EN
VIII SIMPOSIO DE TRATAMIENTO DE SEÑALES, IMÁGENES Y VISION ARTIFICIAL

PROCESAMIENTO DIFUSO DE AUDIO

Jose Ricardo Zapata Gonzalez
Tony Peñarredonda Caraballo

jrzapata@upb.edu.co, *Medellin*
dfingenierias@discosfuentes.com.co, *Medellin*

Resumen: Se presenta el diseño y desarrollo de un sistema de control difuso de tiempo real para señales de audio que permite ejercer un control inteligente sobre el rango dinámico de dichas señales.
Se trata de una aplicación novedosa de los conceptos de control inteligente en la solución de un problema de procesamiento de audio. Se pretende, de esta manera, señalar una alternativa diferente a las tradicionales en el diseño de las unidades dinámicas de audio, que mejora el desempeño y nivel de distorsión de la señal involucrada en el procesamiento.

Palabras Clave: lógica difusa, unidad dinámica de audio, microcontroladores.

1. INTRODUCCIÓN

En diferentes quehaceres tecnológicos como en el ámbito de las comunicaciones, la industria musical, el entretenimiento, etc. existe, frecuentemente, la necesidad de controlar el volumen de la señal de audio en forma automática [1]. De esta manera se evita que el nivel de salida sea excesivo para no saturar los sistemas electrónicos (como amplificadores, transmisores, conversores A/D, medios de almacenamiento, etc), para eliminar efectos desagradables como la presencia de distorsión en la señal, para mantener una presión sonora confortable sobre una concurrencia y otras muchas aplicaciones. En los estudios de grabación, por ejemplo, se utiliza para mantener bajo control la voz humana o un instrumento musical, que pueden variar por los cambios de distancia respecto al micrófono o por la dinámica misma de la ejecución. En otros casos, se emplea para excluir ruidos externos que se cuelan cuando la señal no está presente.
Para efectuar esta regulación sobre la señal, se utilizan los llamados procesadores de dinámica, los cuales se implementan con sistemas electrónicos que por su naturaleza introducen distorsión y ruido en la señal.

Este artículo describe el diseño de una unidad dinámica basada en un fader motorizado que se mueve bajo los comandos de un controlador inteligente que reside en un microcontrolador. Dicho control está basado en lógica difusa y las decisiones se toman basadas en la experiencia de un experto

humano cuyo conocimiento se encuentra incorporado en el controlador

2. UNIDAD DINÁMICA DE AUDIO

Los procesadores dinámicos actúan modificando el rango dinámico de la señal de audio, el cual consiste en la diferencia en Decibeles entre la intensidad más fuerte y la más débil de la señal (acústica, eléctrica, etc). El rango útil de los sistemas de grabación y reproducción no supera en promedio los 80 db (eléctricos análogos 80 db, magnéticos análogos 60 db, digitales 120 db). Por su parte, el rango dinámico generado por la voz o por instrumentos acústicos es alrededor de 130 db, sobrepasando el rango útil de los sistemas en mención.

Antes de que existieran los procesadores de dinámica, el control de nivel debía hacerse manualmente y el operador de la consola debía conocer con anterioridad los cambios que se suscitarían en el material de programa. De esta manera con exactitud, justo antes de que se produjera el cambio, el humano bajaba o subía la ganancia en el equipo. Si la señal se silenciaba en ciertos pasajes, la ganancia se bajaba al mínimo cerrando el paso a ruidos externos. La figura 1 detalla el proceso: el sistema auditivo detecta el volumen, el cerebro, según la experiencia adquirida y los criterios del operador coordina la mano para bajar o subir el volumen mediante el fader.

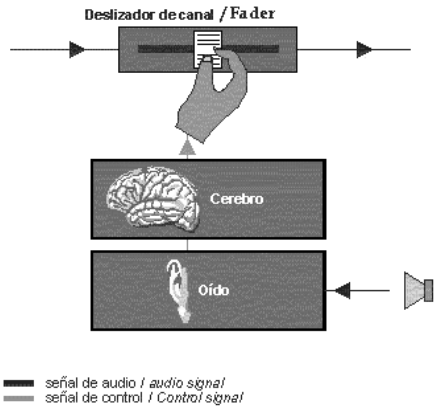


Fig. 1. Procesador "humano" de dinamica.

Este procesador humano de dinámica tiene algunas limitaciones, para comenzar, sólo puede controlar un canal, es lento y sus acciones no pueden ser exactamente repetibles. Además, el operador debe conocer de antemano la pieza musical lo que lleva a que el proceso se alargue y aumente el costo.

El sistema electrónico de procesamiento de dinámica tiene el mismo principio de funcionamiento y supera las limitaciones que presenta el procesador humano. En tales unidades dinámicas de audio, la señal de entrada se divide en dos, una de ellas se procesa a través de un sistema de ganancia controlada que normalmente es un ACV (amplificador controlado por voltaje o VCA por sus siglas en inglés). La otra parte de la señal de entrada va a un circuito de detección que actúa sobre el sistema de ganancia ACV. Para que los cambios de volumen sean graduales y no se escuchen variaciones bruscas durante los cambios del ACV, se utiliza un generador de envolvente el cual establece una rampa cuya forma permite una variación suave entre los cambios de nivel. Normalmente, se puede decidir entre detectar la señal que se va a procesar o detectar una señal externa, en este último caso se habla de señal side – Chain (cadena lateral) o key. La figura 2 ilustra el proceso:

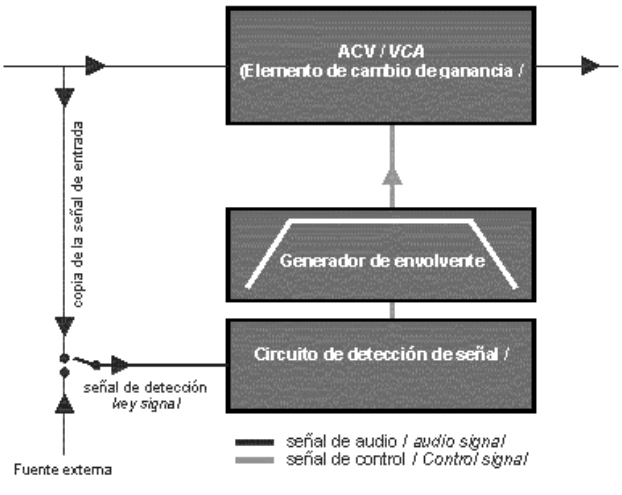


Fig 2. Procesador electrónico de dinámica.

Uno de los efectos secundarios de utilizar los ACV en la señal de audio, es la introducción de distorsión y ruido por la naturaleza misma de los componentes semiconductores que lo conforman, además, debe existir un buen circuito de detección, el cual, agrega complejidad al sistema. Los mejores sistemas de ACV resultan muy costosos y debido a ello se reservan para equipos de altas prestaciones en sonido profesional.

Existen variados tipos de unidades dinámicas, como expansores, compuertas de ruido, compresores y limitadores. El proyecto que se describe corresponde a la función como compresor.

2.1. Compresor

El compresor deja pasar la señal sin hacerle cambios siempre y cuando no supere un nivel pre establecido. El punto en el cual el compresor se activa se llama umbral (threshold), y a partir de él, se comporta como un dispositivo de ganancia variable que depende directamente de la señal de entrada, de tal forma que, si en la entrada hay una señal de intensidad que sobrepasa el umbral, este ajustará su ganancia inferior a uno, y si es una intensidad baja, se ajusta a un valor más alto, pero no superior, a uno. De esta manera se reduce el rango dinámico de la señal.
El limitador es una forma específica de compresor, donde la relación de compresión es infinita, el efecto resultante es un aplanamiento de la señal a partir del punto de umbral.
Se podría decir que comprimir es atenuar de manera suave, pero limitar es hacerlo de forma brusca. Esto es útil cuando se quieren evitar picos de voltaje que puedan dañar los equipos de amplificación, y por lo tanto se utiliza también como sistema de protección

El diagrama. de la característica Entrada vs. Salida de un compresor de audio puede apreciarse en la figura 3.

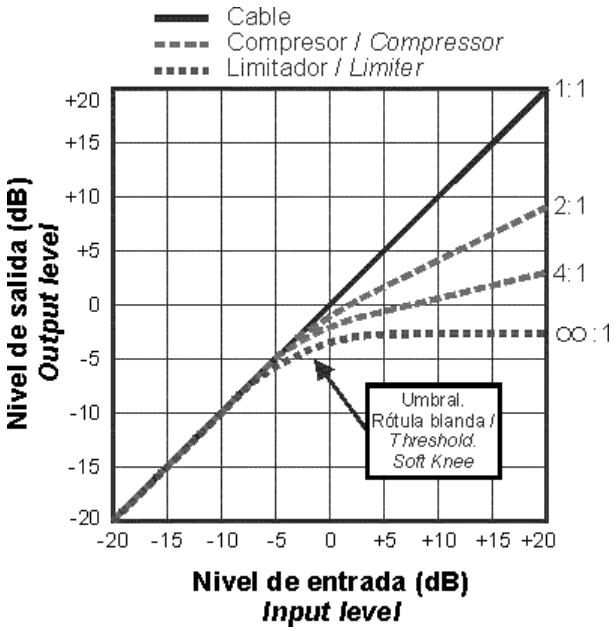


Fig 3. Característica entrada/salida del compresor de audio.

3. LÓGICA DIFUSA

La lógica difusa ha permitido el desarrollo de un tipo de controlador inteligente que imita la forma como los humanos regulan un proceso basado en su conocimiento y experiencia. Esta metodología ha demostrado su utilidad en sistemas en los que no hay linealidad o cuando no se conoce su modelo matemático.

La lógica difusa opera sobre variables lingüísticas a las cuales no se les asignan valores definidos, sino, valores que indican su grado de certeza o falsedad. Con ellas se ensamblan proposiciones que definen las reglas de control.

El controlador basado en lógica difusa desarrollado para la construcción del prototipo maneja dos entradas y una salida. El diagrama básico del controlador difuso se describe en la figura 4.

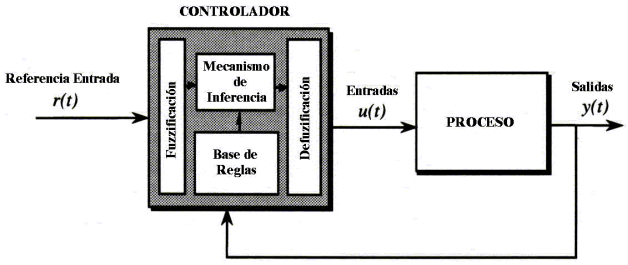


Fig 4. Diagrama básico del controlador difuso adaptado de: Passino, Kevin M. Fuzzy control. Ohio,1998. p.23.

El controlador difuso esta compuesto por 4 bloques básicos:

3.1. Fusificación o emborronamiento (Fuzzification).

Este es el bloque inicial del controlador y su función es la de transformar las variables r(t) y y(t) en variables de tipo lingüísticas. Es decir, a los valores de nivel medidos en estas señales se les asigna un grado de certeza de que pertenecen o no a un determinado conjunto. Estos conjuntos, llamados difusos, describen si los valores leídos son pequeños, medianos, grandes, etc.

3.2. Base de reglas (Rule base).

Contiene el conocimiento de un experto para tomar las decisiones de las acciones de control. Son una serie de proposiciones que describen cómo debe ser la salida dependiendo de los valores de las entradas. Todas estas reglas se establecen de acuerdo con la función que se quiera implementar.

3.3. Mecanismo de inferencia (Inference mechanism).

Toma las decisiones de control interpretando y arbitrando entre las reglas que se activan para obtener una salida con un grado de certeza.

3.4. Defusificación o concreción (Defuzzyfication).

Transforma las decisiones de tipo difuso de las etapas anteriores, en salidas de valor real para el control del sistema. La salida es un valor concreto, un número bien definido el cual alimenta la planta bajo control.

4. CONTROL DIFUSO DE UN FADER MOTORIZADO

Para desarrollar el diseño de este control se debe tener claro que el fader es un potenciómetro lineal utilizado principalmente en aplicaciones de audio.
El sistema a considerar tiene como partes principales la entrada y la salida de la señal eléctrica de audio y un motor de directa (CD). El motor cambia la posición del fader afectando el valor de la salida de audio respecto a la entrada, es decir, puede atenuarla o hacer el valor de salida igual al de entrada.
La señal de audio tomará valores entre 0 y 5 Voltios, y el motor de directa se alimentará entre 0 y 5 Voltios. Recordar que mayor voltaje mayor su velocidad de respuesta).

4.1. Entradas y Salidas del Controlador Difuso.

En este sistema el objetivo es controlar el rango dinámico de una señal de audio, por tal motivo, las variables de entrada al controlador son la salida del fader, con el fin de establecer el error respecto al nivel deseado, y la entrada de la señal de audio a la unidad dinámica. La salida del control es un nivel de voltaje y la polaridad para alimentar un motor de CD. El lazo cerrado lo configura la salida de audio del fader que realimenta al controlador difuso. Como ocurre en la mayoría de los controles clásicos (PID, adelanto – atraso, etc.). Para resumir, las entradas para el control son el error e y la derivada del error de/dt , como se puede ver en la figura 5.

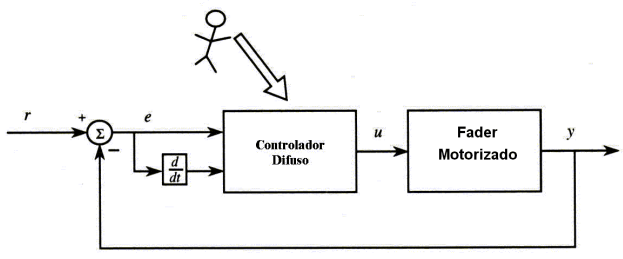


Fig 5. Controlador difuso realimentado Fuente: Adaptado de Passino, Kevin M. Fuzzy control. Ohio,1998. p.25.

4.2. Base de reglas

Las variables una vez leídas y "emborronadas" toman los siguientes valores lingüísticos::

"Negativo_ grande" representado por -2
"Negativo_ pequeño" representado por -1
"Cero" representado por 0
"Positivo_ pequeño" representado por 1
"Positivo_ grande" representado por 2

Cada una de estas variables lingüísticas se representan por unas funciones de membresía. Se escogieron funciones triangulares, dada la facilidad de implementación en software y la realización de cálculos. La programación en un microcontrolador se facilita con este tipo de variables.

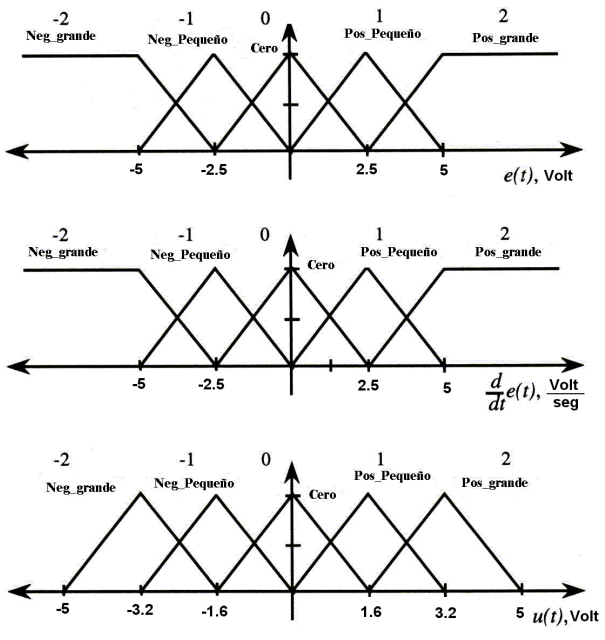


Fig 6. Espacio difuso de las variables lingüísticas

El conocimiento del experto, el cual se encuentra representado en una serie de sentencias almacenadas en una base de reglas, indica la acción a tomar sobre el fader dependiendo de los valores difusos de las entradas. Estas reglas siguen la estructura lingüística:
SI Entrada 1 Y Entrada 2 ENTONCES Salida.
Estas proposiciones se pueden codificar en forma tabular, de manera que los valores que la primera columna y la primera fila representan las variables lingüísticas de las entradas, y los valores que se encuentra en las casillas de cruce de las variables corresponden a la salida sugerida. Como el número de variables de entrada es 2, y el número de valores lingüísticos que puede tomar estas variables es 5, entonces se genera una tabla 5x5 obteniéndose 25 reglas que representarán las acciones a tomar en el sistema.

Tabla 1. Tabla de Reglas

SALIDA		Cambio en el error de/dt				
		-2	-1	0	1	2
Error e(t)	-2	2	2	2	1	0
	-1	2	2	1	0	-1
	0	2	1	0	-1	-2
	1	1	0	-1	-2	-2
	2	0	-1	-2	-2	-2

Para el proceso de concreción se utilizó el método del centroide.

5. IMPLEMENTACION DEL HARDWARE

La unidad dinámica que se implementa en este prototipo es un compresor de audio, el hardware está conformado por los componentes mostrados en el diagrama de bloques de la grafica 7.

Los módulos se desarrollaron de la siguiente manera:

El microcontrolador seleccionado para ejecutar las tareas de control en el prototipo, es el 68HC908GP32 de Motorola.

El elevador se realizó con el integrado LM358, que es un amplificador operacional doble dispuesto en configuración de comparador.

El filtro LP (pasabajos) utilizado es pasivo, debido a la sencillez de su realización y su estabilidad. Se maneja un potenciómetro en el filtro RC, el cual brinda la posibilidad de ajustarlo a las mejores condiciones de comportamiento.

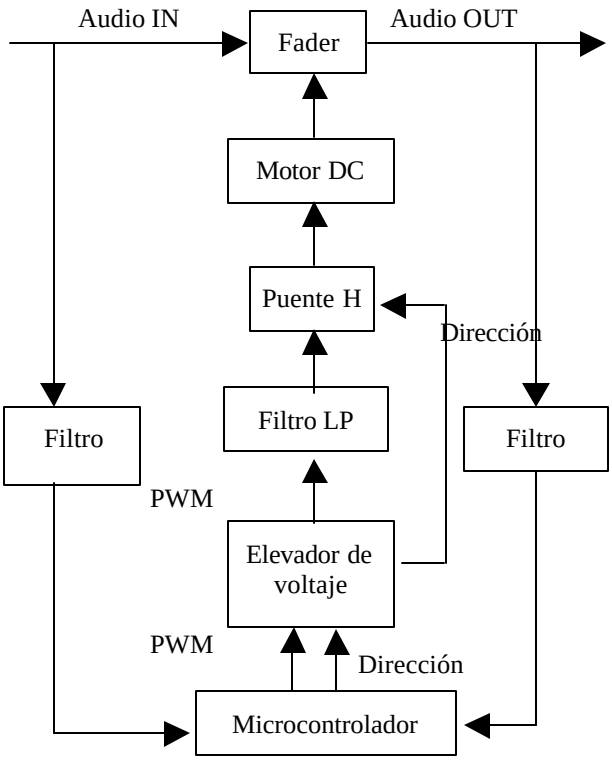


Fig 7. Diagrama de Bloques del prototipo en hardware implementado

El puente H se implementó con el integrado TA7288P, pues, además de ser popular, tiene la capacidad de manejar la potencia y dirección de giro requeridos para el motor.

El fader utilizado es de los que se pueden encontrar en las automatizaciones de consolas, posee un motor de DC incorporado lo cual simplifica el montaje en el aspecto mecánico. La velocidad del motor se controla con un PWM con frecuencia de 120HZ, pues con este valor se logran movimientos muy suaves en el motor.

6. IMPLEMENTACION DEL SOFTWARE

El software se desarrolló en lenguaje C, que proporciona flexibilidad de programación de alto y bajo nivel, además, facilita la depuración y el mantenimiento de los programas. Se trabajó con rutinas que tienen funciones bien definidas, de tal manera que un conjunto de ellas se encarga de cada parte del control difuso, otra rutina lee las entradas al microcontrolador , otra efectúa la salida PWM del mismo y una más, toma los parámetros de configuración, ratio y threshold, para el funcionamiento de la unidad dinámica (en este caso particular actuando como compresor).

La creación del software se realizó en el ambiente integrado de desarrollo CODEWARRIOR, aplicación que provee un grupo de herramientas de desarrollo de software para microcontroladores usando una interfaz gráfica (GUI).

El entorno integrado de desarrollo (IDE) permite, respecto al código:
- Editar.
- Navegar a través de las líneas de comando.
- Depurar.
- Compilar.
- Enlazar.
- Simular

El diagrama de flujo del software diseñado se aprecia en la figura 8:

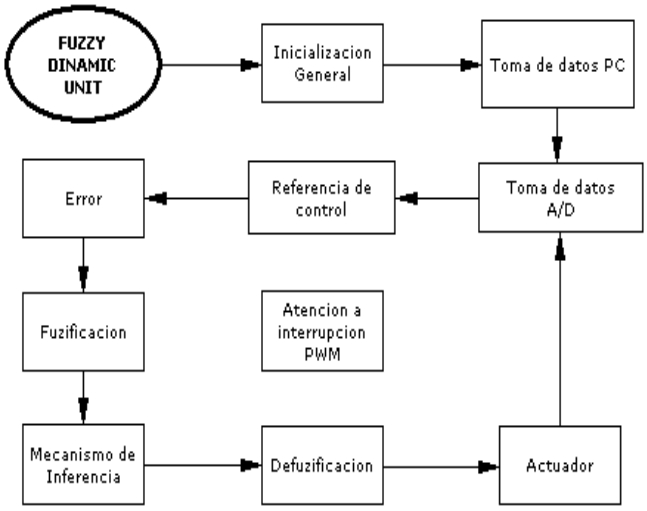


Fig 8. Diagrama de Flujo del Software.

7. RESULTADOS

Al observar el funcionamiento del prototipo, se encontró una disminución en el rango dinámico de la señal de audio cuando esta sobrepasaba el punto de threshold. Los niveles en la salida nunca estuvieron por encima del valor preestablecido por el usuario.

Una de las preocupaciones que existían era la de si mecánicamente, el fader iba a ser capaz de desplazarse con suficiente rapidez para lograr mantener el nivel deseado. Se hicieron ensayos con toda clase de señales, desde las generadas con instrumentos de laboratorio hasta música de diferentes géneros. El funcionamiento fue siempre satisfactorio aun ante transientes de gran tamaño y velocidad.

El software, según lo previsto, trabajaba en tiempo real cerrando el lazo de control, desde que se hacía el muestreo de la entrada hasta que se enviaba el comando de movimiento.

La distorsión medida a la salida de la unidad dinámica difusa era ligeramente la misma de la entrada. Esta es, sin lugar a dudas la principal bondad del diseño. El poco ruido presente durante todo el procesamiento, se debía a la construcción del prototipo y la forma como se implementó el esquema de blindaje y tierras.

8. CONCLUSIONES

El software diseñado e implementado, tiene un rendimiento muy bueno, y requiere poca memoria para su ejecución, lo que lo hace importante para un sistema embebido de tiempo real.

Se realizó un controlador difuso en un dispositivo de procesamiento de 8 bits, y se comprobó que esta resolución es suficiente. Esto se debe a que la digitalización de audio es sólo para efectos de lectura o monitoreo de la señal de entrada, más no para transformar la señal desde la entrada a la salida.

Si se tiene un mayor numero de conjuntos difusos o funciones de membresía, se puede lograr un control mas suave y mas exacto sobre el proceso, pero esto conlleva a mas operaciones a realizar en el microcontrolador, y puede afectar de manera considerable el tiempo de ejecución del control difuso.

Puesto que la señal de audio únicamente pasa por un potenciómetro de deslizamiento lineal, ésta no se afecta por la distorsión introducida por dispositivos semiconductores ni por su ruido inherente.

Se comprueba que para aplicaciones de tiempo real en el domino del audio, no necesariamente hay que recurrir a Procesadores Digitales de Señales, DSPs. La simplicidad del concepto, pese a contener técnicas avanzadas de control inteligente, permite la utilización de microcontroladores no especializados de fácil acceso al ingeniero

RECONOCIMIENTOS

Participacion en la primera competencia de diseño de proyectos de audio, Convencion 115 AES (Audio Engineering Society), new york, NY. Octubre 10 –13 ,2003.

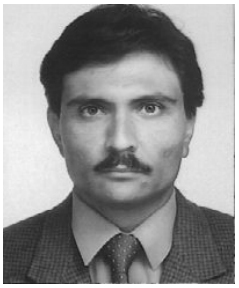
REFERENCIAS

BALLOU, Glen M. Handbook for sound engineers. The new audio cyclopedia. Carmel, Indiana, United States. Second Edition 1991.1506 p.

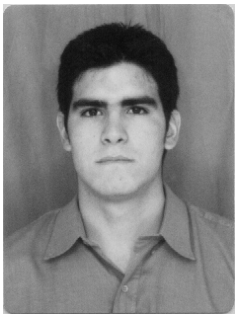
HABER, Rodolfo. PhD. Introducción al control borroso. Portafolio Consultores e.a.t. En: MEMORIAS SEMINARIO INTERNACIONAL SOBRE CONTROL INTELIGENTE DE PROCESOS. Ponencias de primer Seminario Internacional sobre control Inteligente. Medellín, Marzo de 1995.

PASSINO, Kevin M. and Yurkovich, Stephen. FUZZY CONTROL: Ohio, United States. Addison-Wesley 1998. 475p.

AUTORES



TONY PEÑARREDONDA CARABALLO. Ingeniero electrónico de la Universidad Pontificia Bolivariana (UPB). Especialista en Automática y Master en Ingeniería. Desde hace 12 años se desempeña como director del departamento de ingeniería en Discos Fuentes. Es docente universitario en temas de Control Digital y Mecatrónica. Investigador del grupo A+D (Automática y Diseño) de la UPB en las líneas de control inteligente, visión de máquina y robótica. Miembro de la AES(Audio Engineering Society) y fundador de la AES Colombia Section.



JOSÉ RICARDO ZAPATA GONZÁLEZ. nació en Medellín, Colombia, el 07 de Enero de 1981. Terminó sus estudios de Ingeniería Electrónica en la Universidad Pontificia Bolivariana en el 2002 y actualmente se desempeña como docente e investigador en esta institución en el área de Telecomunicaciones. es miembro de GIDATI (Grupo de investigacion y desarrollo en Telecomunicaciones e informatica) en la linea de redes de control. Se encuentra estudiando el Postgrado en Telecomunicaciones en la universidad Pontificia Bolivariana. Es miembro de la AES (Audio Engineering Society).

ANEXO G

ANTEPROYECTO TRABAJO DE GRADO

FUZZY DINAMIC UNIT

ANTEPROYECTO DE TRABAJO DE GRADO

PARA LA UNIVERSIDAD PONTIFICIA BOLIVARIANA

Estudiante:

José Ricardo Zapata González Cédula: 71’375.690 de Medellín
Teléfono: 2658798 E-mail: jjrzg@hotmail.com

Director:

Tony Peñaredonda C. Cédula: 91’425.195 de Barrancabermeja
Teléfono: 2657000 E-maildfingenieria@discosfuentes.com

Noviembre de 2002
Facultad de Ingeniería Electrónica
UNIVERSIDAD PONTIFICIA BOLIVARIANA
MEDELLIN - COLOMBIA

1. INVOLUCRADOS

Autor

Nombre: José Ricardo Zapata González

Facultad: Ingeniería Electrónica

Carné: 98154726

Semestre: Actualmente cursando 10º semestre

Director

Nombre: Tony Peñarredonda C.

Grado: Ingeniero electrónico U.P.B

Master

Cargo: Director Departamento de Ingeniería

Institución: Discos Fuentes

2. MODALIDAD

Descripción

El presente trabajo de grado pertenece a la modalidad de desarrollo de prototipo y/o producto, es dirigido por el ingeniero Tony Peñarredonda C.

3. TEMA DEL PROYECTO

Se trata de una aplicación novedosa de los conceptos de control inteligente en la solución de un problema de procesamiento de audio. Específicamente, se diseñará un controlador difuso para el manejo de una unidad dinámica de audio. Se pretende de esta manera mejorar el desempeño y nivel de distorsión de la señal de audio involucrada en el procesamiento.

Tipo de Trabajo	Teórico			Practico
%	Búsqueda	Estudio	Desarrollo	Desarrollo
	15%	25%	20%	40%

4. ANTECEDENTES

Como antecedentes del proyecto, están el ToolBox de Fuzzy Logic de Matlab, el proyecto de grado del profesor Carlos Peñuela de la Universidad Pontificia Bolivariana, titulado: " Lógica difusa y controladores Difusos", que consiste en la sintonía de un control PID (proporcional-integral derivativo)para el CODILUM(controlador lineal universal multivariable) con base en algoritmos difusos. En la Universidad Nacional se han presentado 2 proyectos de grado con base en algoritmos que se han usado en programas de simulación en lógica difusa, pero no trabajan en tiempo real.
También se tienen trabajos de grado con lógica difusa en tiempo real, este trabajo de grado se realizo específicamente para el CODILUM, titulado: "Control En Tiempo Real Con Lógica Difusa Para El CODILUM" el trabajo consistió en el diseño y elaboración de un software de control encargado de recibir los datos que ofrece el hardware del CODILUM (controlador lineal universal multivariable) y generar una acción de control con base en una matriz de interferencia que puede diseñarse para cualquier control deseado con la ayuda de un segundo software para diseño con lógica difusa.

El grupo de investiagacion A+D (Automática y Diseño, antes GRIAL), al cual pertenece el director de este trabajo, tiene dentro de sus líneas de investigación, el control inteligente. En este grupo se ha venido desarrollando un trabajo continuo en el tema de los controladores difusos, especialmente el FMRLC **(**Fuzzy modeling reference learning control) . Los ingenieros Manuel Betancourt, Tony Peñarredonda y César Quintero han propuesto modificaciones al esquema original de este controlador con éxito notorio y difusión internacional. Este trabajo de grado aprovecharía este bagaje investigativo para una aplicación novedosa en el área del procesamiento de audio.

5. OBJETIVOS-

5.1 Objetivo General

- Se aplicarán técnicas de los controladores difusos para el control de una unidad dinámica de audio. Se diseñará y construirá un prototipo de tal aparato.

5.2 Objetivos Específicos

- Dar a conocer el funcionamiento de una unidad dinámica de audio convencional.
- Demostrar cómo los controladores basados en lógica difusa pueden ser útiles en aplicaciones de procesamiento de audio.
- Construir un prototipo cuya principal característica será la de afectar los niveles de audio según la elección de ciertos parámetros elegidos por el usuario. Un controlador difuso se encargará del comando de un *fader* motorizado para lograr a la salida del sistema las características de audio deseadas.

6. JUSTIFICACIÓN Y BENEFICIOS

Una de las herramientas importantes en el procesamiento de audio es la unidad dinámica, que incluye varias funciones en su familia, a saber: Compuertas de ruido (*Noise gates*), compresores, limitadores, expansores, *de-essers*, etc. Estas funciones normalmente se consiguen a través de equipos *stand alone* o también incluidos en las consolas de grabación y mezcla de audio. La implementación es siempre en el dominio analógico, aunque también se pueden obtener algunas de tipo digital.

El componente principal de una Unidad Dinámica es un VCA (*Voltage Controlled Amplifier- amplificador controlado por voltage*). Este consiste en un amplificador a través del cual circula la señal y cuya ganancia cambia dinámicamente dependiendo de un comando de nivel de voltaje aplicado a un pin de control. A pesar de que se ha avanzado mucho en los niveles inherentes de distorsión y ruido de tales dispositivos, la comunidad de ingenieros de audio prefiere que tal señal de audio tenga el menor recorrido a través de cualquier circuito que se implemente. La alternativa utilizada, es un potenciómetro de acción lineal (*fader*), el cual tiene un cursor motorizado. Es un solución muy práctica, pero se complica para el control automático.

7. ALCANCES

Un trabajo de esta naturaleza sería de gran interés para la comunidad de ingenieros de audio y los ingenieros de control. Se pretende realizar una aplicación de los controladores inteligentes en un terreno muy poco explorado por esta materia, la cual es el procesamiento de audio.

Los resultados de este trabajo sonde gran interés académico, pues sin importar cuales sean estos positivos o negativos, dejaran una experiencia acumulada en la ingeniería la cual podría servir de futura referencia en posteriores trabajos de esta naturaleza. Puesto que en la universidad solo se han desarrollado simulaciones o software acerca de los controladores difusos.

8. TABLA DE CONTENIDOS

- ANEXOS

ANEXO 1 Artículo Publicable (PAPER)

ANEXO 2 Datos del Trabajo de grado

2.1 Involucrados
2.2 Modalidad
2.3 Tema del Proyecto
2.4 Antecedentes
2.5 Objetivos
2.6 Justificación y Beneficios
2.7 Alcances
2.8 Presupuesto, financiación y recursos
2.9 Cronograma y ocupación de tiempos
2.10 Propiedad intelectual y destinación

9. PRESUPUESTO FINANCIACIÓN Y RECURSOS NECESARIOS

DETALLE FINANCIACIÓN	Participación (miles de pesos)			Exige Desembolso	
Costo Recurso Presupuestado.	Estudiante	Servicio UPB	Donación	Si (Nuevo)	No (Existe)
Fotocopias y Bibliografía.	100			100	
Internet, llamadas y fax larga distancia.	200			200	
Transporte	200			200	
Servicios de Cómputo (200h @ 4k$/h)	800			800	
Materiales y componentes *Hardware*	50			50	
Trabajo Estudiante/Asistente (700h @ 6.5k$/h)	4.550				4.550
Trabajo Asesor (40 h @ 35k$/h)			1.400		1.400
Trabajo Director (30h @ 40k$/h *Magister*)			1.200		1.200
Publicaciones (Incluidos informes a biblioteca)	100			100	
Costos Previsibles (Suma subtotal)	6.000	0	2.600	1.450	7.150
Administración e Imprevisibles (10%)	600	0	260	145	715
SUBTOTALES (Por columna)	6.600	0	2.860	1.595	7.865
TOTALES (Deben dar igual)	9.460			9.460	

La financiación de los recursos será por aporte personal.

10. CRONOGRAMA Y OCUPACIÓN

Actividades / Meses	OCT 2002	NOV	DIC	7.1. ENE 2003	Ocupación Horas
Búsqueda Bibliográfica	X	X	X		55
Formulación del proyecto	X				25
Estudio Bibliografía	X	X	X		100
Pruebas con agentes reales				X	30
Elaboración del *software y hardware.*		X	X	X	250
Pruebas y optimización				X	90
Análisis de resultados				X	50
Redacción del informe				X	70
Gestión				X	30
Horas Totales					**700**
Número de Integrantes	1	**Horas promedio por integrante**			**700**

11. BIBLIOGRAFÍA

11.1 Páginas en Internet

- University of Miami Music Engenieering/ Audio Engenieering Home page http://beethoven.music.Miami.edu/programs/mue-ean/mue-ean.htm
- Audio Engineering Society AES http://www.cudenver.edu/aes
- The world wide web virtual Library: Audio http://www.comlab.ox.ac.uk/archive/audio.html
- Acustical Society of America Homepage http://asa.aip.org/

- JBL corporation

http://www.jbl.com

- http://www.answermath.com/Spanish/esp-respuestas-matematicas.htm
- http://delta.cs.cinvestav.mx/~gmorales/ldifll/ldifll.html
- http://personales.ya.com/casanchi/mat/difusa01.htm
- http://www.monografias.com/trabajos6/lalo/lalo.shtml
- http://www.doctorproaudio.com/index.html

11.2 Revistas

- AES: Journal of the Audio Engeneering Society. New York, NY: Audio Engeneering Socity. SIN: 0004-7554.

11.3 Libros

- SAPOSHKOV Mijaíl A. Electroacústica. Ed Reverte. 1983. Biblioteca personal.
- L.L.Beranek. Music, acoustics and architecture. Ed Jhon wiley and sons,1962. Biblioteca UPB 729.2 B37
- Lawrence E. Kinsler. Fundamentos de acustica. Grupo Noriega editores, 1992. Biblioteca UPB. 534 F85
- Leslie L. Doelle, Acoustics in arquitectural design. National research council, 1965. Biblioteca UPB. 729.29 D53
- Sound system design (reference manual) JBL CORPORATION. Biblioteca personal
- L.L Beranek, Olson F Harry. MUSIC, PHYSICS AND ENGENERING. 1967. Dove Publications. Biblioteca U de A. 620.23 M175.

- Rettinger Michael. Acoustics Room design and noise control. Ed chemical publishing co, 1968, Biblioteca Discos Fuentes.
- Peñuela Salazar, Carlos Eduardo. Logica difusa y controladores difusos. Medellin, 1996. Trabajo de grado(ingeniero electronico): universidad pontificia bolivariana. Facultad de ingenieria electrónica.
- Colonia Cuevas, Alejandro. Control en tiempo real con logica difusa para el CODILUM. Medellin , 1997. trabajo de grado (ingeniero electronico): universidad pontificia bolivariana. Facultad ingenieria electrónica.
- Nilsson, Nils J. Inteligencia Artificial una nueva síntesis. Ed McGrawhill, 1997. Biblioteca personal.
- Passino, Kevin. Fuzzy Control.Ed. Addionson-Wesley, 1998. Biblioteca personal.

12. PROPIEDAD INTELECTUAL Y DESTINACIÓN DEL PROYECTO: "FUZZY DINAMIC UNIT"

Los derechos morales de autor corresponden al Director y al autor y a toda persona que a criterio de éste, haga aportes originales en los avances y en el resultado final del proyecto.

El graduando y el director serán autores de todas las publicaciones nacionales e internacionales que genere el proyecto. Estos a su vez determinarán cuales de los colaboradores deben ser mencionados según sus aportes al proyecto. Adicionalmente, en cualquier tipo de divulgación que se haga del trabajo se dará crédito a la Universidad Pontificia Bolivariana, a la Facultad de Ingeniería Eléctrica y Electrónica .

Derechos patrimoniales: Los derechos patrimoniales pertenecerán de manera conjunta a los creadores, en caso de generarse derechos industriales o de explotación serán repartidos únicamente entre los creadores del prototipo en proporciones definidas por el director.

Todos los participantes conocen y aceptan el "Estatuto de propiedad intelectual" de la UPB, y también el reglamento de elaboración de trabajos de grado de IEE, y se comprometen a cumplir los deberes que allí se estipulan.

El retiro podrá ser voluntario o motivado por el incumplimiento de las obligaciones de una de las partes a criterio del director.

_____

Ing. Tony Peñaredonda

c.c 91425195 de Barrancabermeja

_____

Jose Ricardo Zapata Gonzalez

c.c 71'375.690 de Medellin

MEMO DE TRABAJO PARA EL COMITÉ

FUZZY DYNAMIC UNIT

Primera revisión

Recibió: _____ Fecha:_____

Lectura y asignación de evaluador en comité #_____

Estudio preliminar por: _____ Fecha:_____

Decisión del comité en pleno:

Reprobado: ☐ Aplazado: ☐ Aprobado: ☐

Comité # _____ Fecha:_____

Firma Responsable: _____

Comentarios:_____

_____

_____

_____

_____

_____

_____

Medellín, 14 de Noviembre de 2002

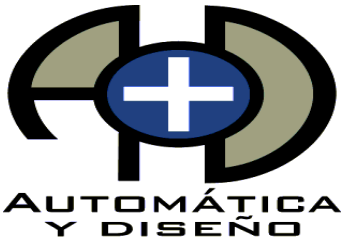
Señores:
Comité de Currículo
Facultad Ingeniería Electrónica
U.P.B

Conociendo los reglamentos de elaboración de trabajo de grado y la responsabilidad del director del mismo, a petición de él estudiante de Ingeniería Electrónica José Ricardo Zapata González, informo que le prestaré la dirección necesaria en el desarrollo del trabajo de grado titulado “FUZZY DYNAMIC UNIT”.

Cordialmente,

_____

Ing. Tony Peñaredonda
c.c 91425195 de Barrancabermeja



Medellín, Noviembre de 2002

Señores:
Comité de Curriculum
Facultad Ingeniería Electrónica
U.P.B

Respetados señores:

Con la presente doy constancia de conocer el proyecto del estudiante de Ingeniería Electrónica José Ricardo Zapata González, **"Fuzzy Dinamic Unit"**, que se encuentra avalado dentro de sus proyectos por el grupo de investigación en automática A+D.
Sin más por el momento:

_____

Ing. Julio cesar Correa
Coordinador del A+D