UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA UNIVERSITARIA DE TELECOMUNICACIÓN

TRABAJO FIN DE CARRERA

ESPECIALIDAD:Equipos Electrónicos.

TÍTULO: Estudio de la Tarjeta de Procesado de Señales de Audio AT-DSP2200 y Desarrollo de un Software.

AUTOR:José Manuel Sánchez Martín.

TUTOR:Fidel Cabrera Quintero.

MES/AÑO: Abril/1995

TRABAJO FIN DE CARRERA

ESPECIALIDAD:Equipos Electrónicos.

TÍTULO: Estudio de la Tarjeta de Procesado de Señales de Audio AT-DSP2200 y Desarrollo de un Software.

AUTOR:José Manuel Sánchez Martín.

TUTOR:Fidel Cabrera Quintero.

MES/AÑO: Abril/1995

CALIFICACIÓN:

EL TUTOR

TRIBUNAL

PRESIDENTE **SECRETARIO** **VOCAL**

ÍNDICE

MEMORIA

CAPÍTULO III. EL D.S.P.

PLIEGO DE CONDICIONES.

PRESUPUESTO.

ANEXOS.

MEMORIA

INTRODUCCIÓN

Con la realización de este Trabajo Fin de Carrera que gira en torno a la tarjeta AT-DSP2200 se ha pretendido cumplir dos objetivos básicos. El primero hacer un estudio detallado del funcionamiento de dicha tarjeta para ver las potencialidades de esta. El segundo ha sido el de desarrollar un software capaz de permitir la operatividad de la AT-DSP2200.

En cuanto a la primera parte del trabajo se ha intentado exponer de forma clara las partes más importantes de tarjeta así como sus posibles aplicaciones. La descripción del hardware se ha hecho pensando en el programador no en el diseñador, ya que la tarjeta es un producto terminado y todas las aplicaciones que de ella se puedan derivar giraran entorno al software.

La segunda parte del trabajo que como ya dije antes es un programa, que se ha realizado para suplir la inexistencia de un software básico que le diera a la tarjeta una operatividad mínima. Aunque la AT-DSP2200 viene con librerías software de propósito general no incorpora ningún programa para usuarios finales. Así con mi trabajo se ha pretendido cubrir este vacío haciendo un programa de propósito general que cubra las necesidades básicas de la digitalización de señales de audio y además dejar la puerta abierta para que en el futuro y contando con las herramientas adecuadas se puedan desarrollar aplicaciones específicas para el laboratorio de sonido.

Pensando en esto he diseñado un programa totalmente modular encajado dentro de un menú configurable y de propósito general.

La AT-DSP2200 es una tarjeta muy potente y en consecuencia necesita herramientas de programación potentes, en mi caso me he tenido que limitar a las librerías y al uso del Microsoft C, logrando con ello desarrollar una aplicación básica como ya he dicho. He tenido que renunciar a características como el procesado en tiempo real o un mayor control de todos los registros. Para poder acceder a estas ventajas hay que disponer de una herramienta adicional de programación que no se incluye en el kit inicial, este kit adicional es el llamado Developer Toolkit. Esta es sin duda la herramienta fundamental si se quiere sacar partido de las funciones avanzadas de la AT-DSP2200. Pero si además se quiere disponer de un entorno gráfico de trabajo se ha de disponer también de el LabWindows versión 2.2 o superior. El LabWindows es un entorno de programación en modo gráfico orientado hacia la adquisición y control que permite de una forma fácil y rápida ofrecer un soporte gráfico de gran calidad y vistosidad a nuestras aplicaciones. Es sin duda la combinación de estas dos herramientas las que nos permiten hacer rendir al máximo a esta potente tarjeta para procesado de señales y digitalización.

1. EL SONIDO Y EL PC

1.1 TARJETAS DE SONIDO PARA PC

Desde la aparición de los primeros PC hasta hoy estos han ido mejorando en casi la totalidad de sus características, más velocidad, más memoria, mayor resolución de vídeo, etc.. Hay algo que sin embargo no ha cambiado y esto es el sonido, desde aquel primer PC basado en un microprocesador 8086 hasta los actuales potentes Pentium, todos incorporan un simple altavoz y un timer programable como interfaz de sonido.

Al principio esto no era un gran inconveniente pues la capacidad de los primeros ordenadores no les permitía más que aplicaciones de texto y todo el sonido que necesitaban generar eran algunos pitidos de aviso, los procesadores no eran lo suficientemente potentes como para soportar cargas de trabajo intensivas. Con el paso del tiempo, poco tiempo, los microprocesadores se desarrollaron de tal forma que el PC pasó de ser poco más que una sofisticada máquina de escribir a ser una potente plataforma para realizar todo tipo de aplicaciones. Creció la memoria, aparecieron nuevos conceptos como la programación gráfica y el PC se transformó totalmente hasta convertirse en la herramienta indispensable de trabajo de la mayoría, especialmente la gente del campo de la ciencia. Se vio como el ordenador podía ser programado para que desarrollase las tareas más complejas, los fabricantes fueron añadiendo nuevas características y supliendo sus carencias. Es en este marco donde

surgen las tarjetas de sonido, al principio de 8 bits y orientadas a su uso en los programas de juegos, luego las de 8 bits estéreo, ya con aspiraciones profesionales y por último las de 16 bits y frecuencias de muestreo de 44,1 Khz o más. Con esto quedaba solucionado el problema del sonido.

El PC ya era una herramienta completa. Luego fueron apareciendo diferentes modelos de tarjetas, para diferentes aplicaciones y con diferentes características, la popular Sound Blaster y compatibles orientadas hacia el ocio o los entornos multimedia, pero de la misma forma el campo del sonido profesional también comenzó a mirar al PC como una herramienta útil, así se comenzaron a fabricar tarjetas de sonido para uso profesional que permitían almacenar en disco una canción y posteriormente editarla de forma más cómoda y rápida que la forma tradicional, también se vio la posibilidad de realizar tarjetas de propósito general. Una tarjeta que conectada al P.C permitiera convertir a este en varias herramientas y donde el límite fuera puesto por la imaginación del programador. De esta forma con estas tarjetas podríamos tener un potente analizador de espectro, realizar filtrados en tiempo real, calcular correlaciones, transformadas de Fourrier, etc. y todo con sólo cambiar el programa. Esta es sin duda la gran ventaja que ofrecen las tarjetas llamadas de sonido.

1.2 LAS TARJETAS DE SONIDO COMERCIALES.

Lar tarjetas de sonido comerciales surgieron como consecuencia del auge de los videojuegos y al principio su principal utilidad consistía en dar soporte sonoro a los mismos. El desarrollo de estas tarjetas se ha basado fundamentalmente en dos

estándares (AdLib y Sound Blaster). Estos dos fabricantes con el paso del tiempo ha ido acaparando el mercado y desarrollando productos cada vez mejores y ya no sólo orientados a los videojuegos sino al creciente mundo multimedia. Todas estas tarjetas tiene una serie de características comunes como; interfaz MIDI, entrada de micrófono, amplificador de salida y la capacidad de sintetizar sonidos. Es esta última capacidad es la que más caracteriza y diferencia a unas tarjetas de otras. Al principio la síntesis se realizaba mediante modulación de frecuencia con chips especiales pero últimamente las tarjetas más sofisticadas incorporan las llamadas tablas de ondas, que son fragmentos de instrumentos que se guardan en ROM o en ficheros en el disco duro.

Estas tarjetas también tiene la posibilidad de digitalizar las señales de audio y guardarlas en uno de los diferentes formatos de audio. Esta digitalización se realiza en formatos de 8 o 16 bits y la técnica de digitalización suele ser la codificación PCM. Las frecuencias de muestreo típicas de estas tarjetas suele ser 8, 11, 22 y 44 Khz. La primera frecuencia proporciona una pobre calidad de sonido, equivalente a los sistemas de avisos públicos, presentes en los aeropuertos. Con 11 Khz el sonido se parece al de una línea telefónica, con 22 Khz se logra una calidad equivalente al de una cinta de casette y con 44 Khz y 16 bits se logra una calidad como la de los CD de audio.

Para reducir la cantidad de información a almacenar en el disco duro, se emplean métodos de compresión. Uno de estos métodos es similar al que se usa cuando se duplica un disco duro. Mediante un algoritmo se reduce la cantidad de

datos, buscando secuencias similares y valores continuados. Esta compresión puede realizarse mediante un programa o con por hardware (integrados especializados). Independientemente del método que se use este debe ser capaz de realizarse en tiempo real, es decir mientras se realiza la adquisición. Existen en el mercado pocos formatos para compresión y entre estos podemos destacar los siguientes:

- El ADPCM (Adaptative Pulse Code Modulation), este método se basa en una compresión con pérdidas, esto es, se reduce, en una pequeña proporción la calidad sonora al realizar la compresión/descompresión. En lugar de almacenar cada valor de la muestra, se anota únicamente la diferencia entre cada muestra y la siguiente. Para señales que no cambien bruscamente, se logra una adecuada calidad sin necesitar gran volumen de datos. Este sistema resulta muy adecuado para la voz humana. El mayor problema del ADPCM es que hay varios subformatos que resultan incompatibles entre sí, y ninguno de ellos está directamente soportado por el driver de Microsoft para los ficheros .WAV de Windows. El integrado SoundPort de Analog Devices, AD1848, presente en muchas tarjetas que soportan Windows Sound System, el Business Audio Compaq, y el CD-I (Compact Disc-Interactive) incorporan compresión de sonido ADPCM que se está conviertiendo en un estándar.

- Otro sistema de codificación es el conocido como Ley-A, es un método de codificación no lineal para almacenar una señal digital de audio que utiliza 13 bits de resolución, pero cuyo almacenamiento ocupa sólo 8 bits. El método Ley-μ, que también es no lineal, opera con 14 bits, pero también con almacenamiento de

tamaño de 8 bits. Estos dos métodos son igual de rápidos y mucho más rápidos que el ADPCM. La fórmula para realizar la comprensión siguiendo la Ley-μ es

$$Y = \frac{\log(1+\mu x)}{\log(1+\mu)} \text{ para } x \geq O$$

donde:

Y: es la magnitud de salida.

x: es la magnitud de entrada

μ: es un parámetro definido para determinar el grado de comprensión.

El valor de O corresponde a la amplificación lineal.

La fórmula para realizar comprensión siguiendo la Ley-A es la siguiente:

$$Y = \begin{cases} \frac{Ax}{1+\log A} \text{ para } O \leq |x \leq| \frac{1}{A} \\ \frac{1+\log(Ax)}{1+\log A} \text{ para } \frac{1}{A} \leq |x| \leq 1 \end{cases}$$

En este caso A es el parámetro que determina el grado de comprensión.

Como ya dijimos otra característica de estas tarjetas es el interfaz MIDI. El formato MIDI contiene órdenes para generar sonidos más que los sonidos en sí mismos. El hardware será capaz de interpretar estas órdenes generando el sonido a través de los métodos de generación que antes hemos mencionado. La conexión MIDI de la tarjeta permite enviar las órdenes para que sea un dispositivo externo (como un sintetizador de alta calidad) el encargado de interpretaras y así producir mejor calidad de sonido que la proporcionada por la tarjeta. Los sintetizadores MIDI son capaces de trabajar con 24 o 32 voces (notas) simultáneas, frente a las 9 o 20 que se sintetizan con el método de síntesis FM. Eventualmente la tarjeta puede recibir órdenes MIDI

provenientes de un teclado y encargarse de almacenarlas en un fichero. Los ficheros MIDI resultan muy compactos ya que no contienen los sonidos, sino la codificación de las notas e instrumentos que los producen. Las órdenes MIDI contiene información de las notas (como una partitura) y el instrumento que debe interpretarlas, así como detalles adicionales de la escala a introducir y eventualmente los ritmos a introducir.

1.3 SÍNTESIS DE SONIDOS

La mayor parte de las ondas sonoras tiene una variación periódica en el tiempo cuya amplitud sigue una forma de tipo sinusoidal. Almacenar en forma digital este tipo de forma de onda requiere una gran cantidad de datos, lo mismo que generar una onda senosoidal a partir de una información digital. Sin embargo las matemáticas acuden en ayuda de esta transformación. Diversos estudios numéricos indican que una onda senosoidal puede ser construida por la suma de un cierto número de ondas digitales (cuadradas) cuya frecuencia sean múltiplos impares (frecuencias 3n, 5n, 7n,...) de la frecuencia básica que se desea y con amplitudes decrecientes. Realmente cuando no hay un sonido puro, este contiene no simplemente una frecuencia única, si no un conjunto de ellas, que por lo general son múltiplos unas de otras. Estos son los denominados armónicos. Aunque de forma no perceptible aisladamente, los armónicos de una señal dan realce y profundidad a una señal. cuantos más armónicos se incluyan para reproducir una forma de onda, más fiel es al sonido original será la reproducción.

Hay dos parámetros básicos para caracterizar una señal sonora; su frecuencia de repetición y su amplitud. Cuando la amplitud no es constante en el tiempo se define un valor de envolvente, esto es, la curva que engloba la amplitud a lo largo del tiempo. A su vez una señal sonora se caracteriza por varios parámetros de la envolvente: ataque, caída, mantenimiento. Cada instrumento musical, o mas bien cada familia de ellos, presenta unas características típicas. Así los instrumentos de cuerda, como la guitarra, presentan un ataque fuerte, un mantenimiento escaso y una caída prolongada. Un instrumento de viento por el contrario tendrá un mantenimiento prolongado y un ataque y caída bruscos. Controlando tanto los generadores de frecuencia discretos como un adecuado ajuste de la envolvente, se logra simular los sonidos por los instrumentos musicales y otras fuentes sonoras.

Teniendo en cuenta los factores anteriores, hay dos formas básicas de producir y sintetizar, una señal analógica a partir de datos o procesos digitales. La primera es la síntesis FM (Modulación de frecuencia), que utiliza la combinación de varios generadores de frecuencia para emular (sintetizar) de la forma más aproximada posible la voz o instrumento que produce. Para simular el timbre de un instrumento, se utiliza un generador de ondas senosoidales (operador) que modifica el funcionamiento de otro. Los chis tipo OPL2 presentan dos operadores, mientras que el moderno OPL3 contiene cuatro operadores, lo que proporciona mejor calidad acústica.

La generación Wave Table (Tabla de ondas) permite crear sonidos con calidad profesional. En lugar de simular por síntesis los sonidos, se utiliza una librería de

muestras digitalizadas (con alta calidad) grabadas a partir de los instrumentos reales. Se realiza una corta grabación de cada nota básica de cada instrumento (se eligen unos 128 instrumentos en la mayoría de los caso) y se almacenan en ROM sobre la propia tarjeta, o en ficheros en el disco duro. Para producir una melodía se "unen" las notas individuales para formar la orquesta deseada. El resultado presenta una gran riqueza sonora, muy cercana al instrumento original.

1.4 DIFERENCIAS BÁSICAS ENTRE LA AT-DSP2200 Y LAS TARJETAS DE SONIDO COMERCIALES.

Una vez que hemos visto como funcionan las tarjetas de sonido comerciales podemos apuntar algunas diferencias con respecto a la AT-DSP2200. Quizás la principal diferencia es que la AT-DSP2200 "no es una tarjeta de sonido". Está afirmación conviene matizarla, digo que no lo es porque no ha sido diseñada para dotar de sonido al PC ni para proporcionarle un interfaz MIDI. La AT-DSP2200 ha sido creada para trabajar "junto al PC", para ser usada como una herramienta multipropósito para el tratamiento y análisis de las señales cuyas frecuencias se encuentren dentro de la banda de audio. Por esto tiene características similares a las demás tarjetas, programada adecuadamente puede incluso emular casi todas las funciones, pero también puede realizar tareas que le son exclusivas. La AT-DSP2200 no ha sido pensada sólo para una función sino que su diseño abierto y las herramientas de programación le permiten realizar prácticamente todas las acciones en el campo del sonido profesional. Sólo por poner un ejemplo de la versatilidad podemos citar lo siguiente. Supongamos que necesitamos un analizador de espectro en tiempo real. En

lugar de invertir varios cientos de miles de pesetas en comprar uno, simplemente realizamos un programa y nos construimos uno a medida. Pero supongamos que además necesitamos un dispositivo de filtrado en tiempo real como el que usan los radioaficionados para limpiar el sonido en recepción. Basta con programar otra vez la AT-DSP2200 y ya tenemos dos instrumentos.

Podemos concluir que si bien las tarjetas de sonido comerciales y la AT-DSP2200 tiene características comunes, son distintas en sus objetivos.

1.5 LOS FORMATOS DE AUDIO

Existen en el mercado numerosos formatos para almacenar el sonido en el interior del PC, podemos decir que cada fabricante tiene su propio formato. El uso extendido del Windows ha hecho que determinados formatos se hayan impuesto sobre otros, de forma que en la actualidad el formato para muestras digitales más extendido en el mundo del PC es el formato .WAV compatible con Windows. A continuación se indican los formatos más comunes del mercado.

- Formato **.AIF:** Formato de sonido estándar de Apple Computer.
- Formato **.AVI:** Formato de fichero que contiene información de Audio y Vídeo entrelazada, utilizada por Microsoft Vídeo Windows.
- Formato **.MID:** Formato de fichero estándar de MIDI.
- Formato **.MOD:** Formato estándar de Media Visión Pro Audio Spectrum.
- Formato **.RL2:** Formato de fichero estándar de sonido AdLib.

- Formato **.RMI:** RIFF MIDI (Resource Interchange File Format MIDI). Formato de fichero estándar de sonido de Microsoft MIDI. Puede incluir un dibujo, una etiqueta y un texto para descripción.
- Formato **.SND:** Formato de fichero estándar de sonido de NeXT Computer.
- Formato **.VOC:** Formato de fichero estándar de sonido de Creative Labs (formato Sound Blaster).
- Formato **.WAV:** Formato de fichero estándar para Windows.

En la mayoría de los casos la conversión de un fichero a otro no suele ser difícil siempre y cuando se tenga la información de dichos formatos.

2.- DESCRIPCIÓN DE LA TARJETA

2.1.- DESCRIPCIÓN Y DIAGRAMA DE BLOQUES.

La AT-DSP2200 es una tarjeta del alta precisión para audio que incorpora un D.S.P. (Digital Signal Pocesador) y que se conecta en un slot libre de 16 bits de un PC AT. La AT-DSP2200 usa un DSP de AT&T, el WEDSP32C, como procesador principal. Con este chip, se pude tener una velocidad de procesamiento de 25 MFLOPS. De esta forma con el software adecuado se pueden ejecutar aplicaciones en tiempo real. Aplicaciones tales como filtrado y análisis espectral, la AT-DSP2200 puede sustituir costosos equipos de proceso en tiempo real. Además el potente Procesador Digital puede ser usado para realizar cálculos complejos de forma mas eficiente que el Microprocesador del P.C. La Ultra Rápida potencia de cálculo del DSP hace que la AT-DSP2200 sea ideal para el procesamiento de grandes arrays de datos. Podemos decir en definitiva que la AT-DSP2200 es en realidad una tarjeta de adquisición de datos para audio y un potente ordenador especializado en el proceso digital de señales.

Las principales Características de la AT-DSP2200 son las siguientes:

1. Incorpora un Procesador Digital de Señal con las siguientes características:
    - Velocidad de operación 50 Mhz.
    - 25 MFLOPS/12.5 MIPS.
    - Instrucciones de 24 bits y 24 bits de direcciones.

- Instrucciones de dos y tres operandos.
- Zero-overhead looping.
- Opera con 32 bits de coma flotante y con 16 y 24 de coma fija.
- Cuatro Acumuladores de 40 bits.
- Convierte a o desde el formato IEEE-745 de punto flotante en una sola instrucción.
- Puerto serie de E/S a de 25 Mbits/seg con interrupciones y DMA a memoria local.
- Puerto Paralelo de E/S de 16 bits con interrupciones y DMA a memoria local.
- Tecnología CMOS de bajo consumo.

2. Memoria local de 384 Kwords (1 Word=Palabra de 32 bits) con cero estados de espera.
3. Soporte de interrupciones.
   - De la AT-DSP2200 a la CPU del PC.
   - De la CPU a la AT-DSP2200.
4. Soporte de DMA
   - De la memoria de la AT-DSP2200 a o desde la memoria del PC.
   - De la memoria de la AT-DSP2200 a o desde la placa de E/S.

La AT-DSP2200 tiene dos canales de entrada de 16 bits y los muestrea simultáneamente con un sobremuestreo de 64 veces (64-time oversampling), los conversores A/D son del tipo Delta Sigma, que incorporan un filtro antialising digital para una adquisición de precisión. Los dos canales de salida usan un filtro digital anti-

imagen con un sobremuestreo de 8 veces y los convertidores D/A son del tipo Delta Sigma con un sobre muestreo de 64 veces.

La frecuencia máxima de conversión tanto en entrada como salida es de 51.2 Khz.

La AT-DSP2200 incorpora un bus RTSI (Real -Time System Integration) propiedad de National Instruments, fabricantes de la tarjeta, que incluye un enlace de datos serie para conectarse a o desde otras placas de National Instruments o con otras AT-DSP2200, siendo posible la sincronización del muestreo entre varias placas AT-DSP2200. Además por el bus RTSI la AT-DSP2200 puede recibir una petición de interrupción que puede usarse como una señal de disparo.

La AT-DSP2200 esta pensada para digitalizar o sintetizar señales con un ancho de banda de hasta 22 Khz con lo que cubre perfectamente todo el espectro de audio. Los filtros anti-aliasing en la entrada y anti-imagen en la salida aseguran una gran precisión en la adquisición y reproducción de señales.

La AT-DSP2200 esta dotada de osciladores a cristal para generar todas las frecuencias estándar de audio digital, estas frecuencias se muestrean en la tabla 1.

Tabla 1. Frecuencias de muestreo de la AT-DSP2200

X1 36.864 Mhz		X2 16.9344 Mhz	X3 19.6608 Mhz
48 Khz	32 Khz	44.1 Khz	51.2 Khz
24 Khz	16 Khz	22.5 Khz	25.6 Khz
12 Khz	8 Khz	11.025 Khz	12.8 Khz
6 Khz	4 Khz	5.5125 Khz	6.4 Khz

La AT-DSP2200 se acompaña de dos herramientas que facilitan la programación de sus dos principales partes, el DSP y la parte de A/D D/A, estas herramientas que se describirán con detalle en el capítulo 4 son el interfaces de utilidades NI-DSP para DOS y LabWindows, un conjunto de librerías que permite programar directamente el DSP, y el la utilidades NI-DAQ para DOS, LabWindows y Windows que son otro conjunto de librerías que lo mismo que las anteriores pueden ser llamadas desde una aplicación de usuario y permiten realizar la adquisición de datos a alta velocidad así como generar formas de onda, crear buffers, etc. Estas últimas librerías, las NI-DAQ, vienen con interfaces para diversos lenguajes profesionales para DOS como; BASIC, Turbo Pascal , Turbo C, Turbo C++, Borland C++ y Microsoft C, para Windows incluyen interface para los siguientes lenguajes; Visual Basic, Turbo Pascal, Microsoft C con SDK y Borland C++ para Windows. Existe además de lo ya mencionado una herramienta de desarrollo adicional que no se incluye en paquete básico y que es sin duda la herramienta fundamental si se quiere desarrollar aplicaciones de alto nivel que permitan sacarle todo el partido a la tarjeta. Este paquete adicional de software llamado Developer Toolkit contiene un Compilador , un Ensamblador, un Linkador y un Simulador, con esta herramienta se puede programar directamente a la AT-DSP2200.

En resumen, podemos decir que las herramientas software que acompañan al la AT-DSP2200 son de dos tipos. Una serie de rutinas para su uso con el DSP, las NI-DSP, y otras para programar la AT-DSP2200 como una tarjeta de adquisición de datos normal.

Como ya dijimos antes la AT-DSP2200 cuenta con dos canales analógicos de entrada y dos canales analógicos de salida, pero tiene también un conector para un disparador digital externo que permite iniciar la conversión al recibir una señal digital sobre ese conector. En la figura 1 se muestra un esquema de este conector y en la tabla 2 su significado.

Figura 1. Esquema de los conectores de la AT-DSP2200

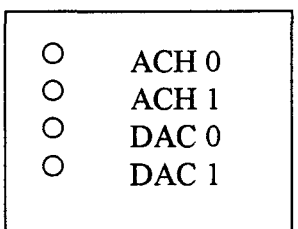


Tabla 2. Descripción del conector de señales.

Nombre de la señal	**Descripción**
ACH 0	Entrada analógica Canal 0
ACH1	Entrada Analógica Canal 1
DAC 0	Salida Analógica Canal 0
DAC 1	Salida Analógica Canal 1
EXTRTING*	Disparo digital externo

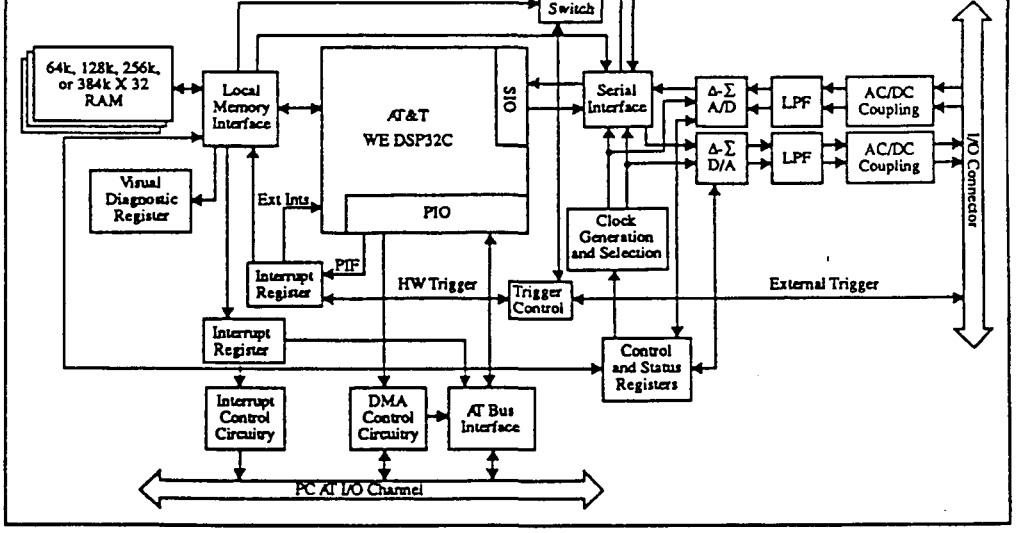
En sucesivos capítulos iremos describiendo en detalle todos las partes de la AT-DSP2200, en la figura 2 se muestra un diagrama de bloques de la tarjeta que nos servirá en el futuro como base para ir describiendo en detalle las partes más importantes.

2.2.- INSTALACIÓN Y CONFIGURACIÓN.

Antes de Instalar la tarjeta debemos comprobar que en la caja se incluyen todas las partes del kit, que son las siguientes:

- La placa AT-DSP2200.
- El manual de usuario de la AT-DSP2200.
- El software NI-DSP para DOS y LabWindows con su manual.
- EL software NI-DAQ para DOS , Windows y LabWindows, con sus dos manuales.
- Un disco de ejemplos y utilidades para la AT-DSP2200.
- Un manual de información del DSPWE32C.

Figura 2. Diagrama de bloques.



Los requisitos mínimos hardware para instalar la AT-DSP2200 son disponer de un PC AT 286 con un ranura de 16 bits tipo ISA, pero esta es una configuración desaconsejable y además imposibilita el uso de software que se ha desarrollado para la tarjeta, siendo por tanto en este caso los requisitos mínimos un PC tipo 386 o superior con una ranura de 16 bits tipo ISA libre.

Antes de proceder a la instalación dentro del PC debemos configurar la dirección base y el acoplamiento de los canales de salida. Para ello la AT-DSP2200 viene equipada con un interruptor DIP y dos junpers, uno para cada canal de salida. El acoplamiento de los canales de entrada se realiza por software. El interruptor DIP sirve para seleccionar la dirección base del canal de E/S y permite diferenciar los mandatos sobre la AT-DSP2200 de los do otras tarjetas conectadas en PC. Este interruptor actúa sobre las líneas de direcciones A5, A6, A7, A8 y A9. La AT-DSP2200 viene de fabrica con la dirección base 140 hex, esta dirección es la que he usado para conectar la tarjeta en el PC, y en la mayoría de los casos debe ser válida y funcionar correctamente, no obstante, debe verificarse que no hay ningún dispositivo conectado a las ranuras de expansión del PC que usen esta dirección, si así fuera se debe cambiar esta dirección por otra de las listadas en la tabla 2. La forma de cambiar la dirección base es la siguiente:

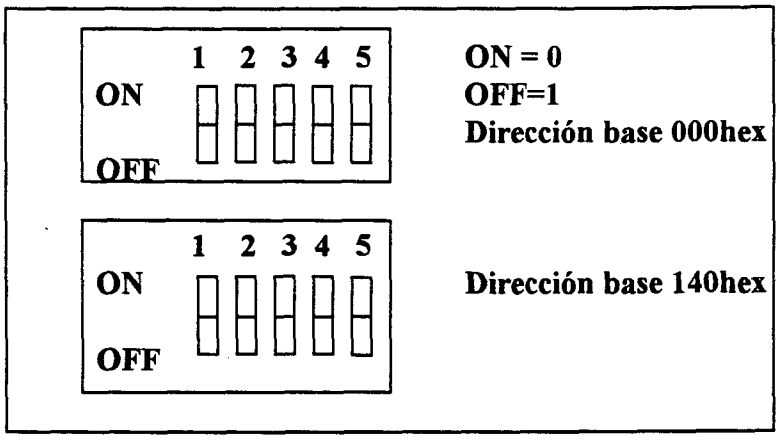
- Localizamos el interruptor DIP marcado como I1 en la figura 2, y con la punta de un bolígrafo o similar colocamos la nueva dirección base siguiendo las referencias de la tabla 3.

Tabla 3. Listado de las posibles direcciones base

Posiciones del interruptor					Dirección base de E/S (en hex)	Espacio usado por la dirección base de E/S
A9	A8	A7	A6	A5		
0	0	0	0	0	000	000-01F *
0	0	0	0	1	020	020-03F *
0	0	0	1	0	040	040-05F *
0	0	0	1	1	060	060-07F *
0	0	1	0	0	080	080-09F *
0	0	1	0	1	0A0	0A0-0BF *
0	0	1	1	0	0C0	0C0-0DF *
0	0	1	1	1	0E0	0E0-0FF *
0	1	0	0	0	100	100-11F
0	1	0	0	1	120	120-13F
0	**1**	**0**	**1**	**0**	**140**	**140-15F**
0	1	0	1	1	160	160-17F
0	1	1	0	0	180	180-19F
0	1	1	0	1	1A0	1A0-1BF
0	1	1	1	0	1C0	1C0-1DF
0	1	1	1	1	1E0	1E0-1FF *
1	0	0	0	0	200	200-21F *
1	0	0	0	1	220	220-23F
1	0	0	1	0	240	240-25F
1	0	0	1	1	260	260-27F *
1	0	1	0	0	280	280-29F
1	0	1	0	1	2A0	2A0-2BF *
1	0	1	1	0	2C0	2C0-2DF *
1	0	1	1	1	2E0	2E0-2FF *
1	1	0	0	0	300	300-31F *
1	1	0	0	1	320	320-33F
1	1	0	1	0	340	340-35F
1	1	0	1	1	360	360-37F *
1	1	1	0	0	380	380-39F *
1	1	1	0	1	3A0	3A0-3BF *
1	1	1	1	0	3C0	3C0-3DF *
1	1	1	1	1	3E0	3E0-3FF *

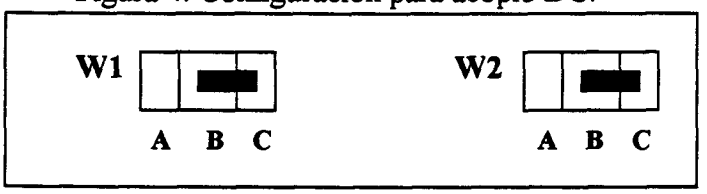
Nota: Las direcciones base de la 000 a la 0FF están reservados para el sistema y por tanto no deben usarse. Las direcciones de 100 hasta 3FF están disponibles. Las direcciones marcadas con un asterisco pueden ser direcciones reservadas para el sistema dependiendo del tipo de PC.

Figura 3. Posición del interruptor DIP.



El siguiente paso es configurar el acople de los canales de salida seleccionando entre acoplo en corriente continua DC o alterna AC. Para ello localizamos los jumper W1 y W2 para el canal cero y uno respectivamente. La forma de hacerlo se precia en las figuras 4 y 5.

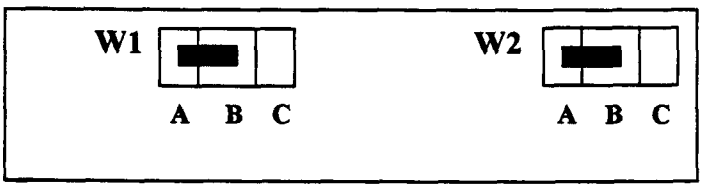
Figura 4. Configuración para acoplo DC.



Canal de salida 0 W1 B-C

Canal de salida 1 W2 B-C

Figura 5. Configuración para acoplo DC



Canal de salida 0 W1 A-B

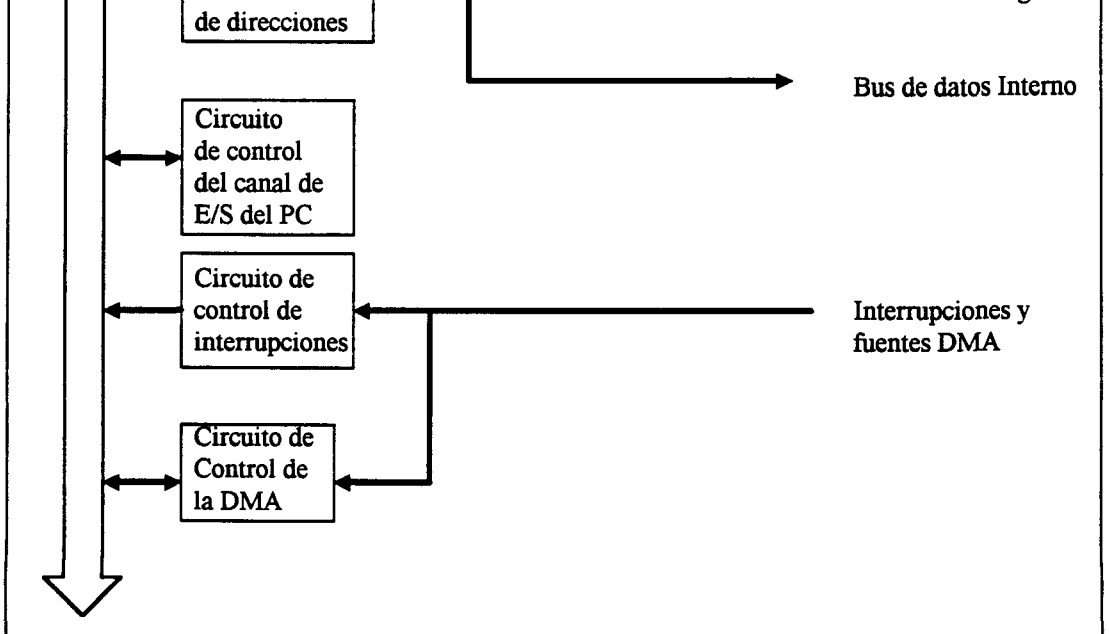
Canal de salida 1 W1 A-B

La configuración por defecto y la que yo he adoptado es el acoplo en DC.

Una vez hecho todo lo anterior podemos abrir el PC y enchufar la tarjeta en una ranura de 16 bits que se encuentre libre. Debemos procurar que a los lados de la tarjeta, siempre que sea posible, haya una ranura libre de forma que haya el máximo espacio posible entre esta y las demás tarjetas instaladas en el PC para de esta forma mejorar las características de ruido. La instalación del software se describirá el capítulo 4

2.3 CIRCUITO DE INTERFACE DE E/S CON PC.

Figura 6. Diagrama de bloques del canal de E/S con PC



La AT-DSP2200 tiene un conector tipo ISA de 16 bits para conectarse con el canal de E/S del PC. Consiste en un bus de direcciones de 24 bits, un bus de datos de 16 bits, un de control de DMA, líneas de interrupciones y algunas otras señales de control. En la figura 6 se muestra un esquema del circuito.

2.3.1 Controlador De Bus.

El controlador de bus controla el recibo y envío del los datos a y desde el PC. Por lo general el controlador de bus suele ser un chip específico que se diseña para esa función y actúa tomando el control del bus del PC o la tarjeta .

2.3.2 Decodificador De Direcciones.

Como ya dijimos antes el canal de E/S del PC tiene 24 líneas de direcciones. La AT-DSP2200 usa 10 de esas líneas para decodificar la dirección de la placa, con lo que nos da un rango de direccionamiento desde 000 a 3FF como se indicó en la tabla 3.

2^10=1024 Kbytes → 000- 3FF en hex

Así con las líneas A5 .. A9 se genera la dirección base de la placa que es la que permite habilitarla. Con las cinco líneas de mayor peso A0.. A5 se seleccionan los registros internos de la placa

2.3.3 Circuito De Control Del Canal De E/S Con PC.

Este circuito monitoriza y transmite al canal de E/S del PC señales de control y apoyo. Las señales de control identifican transferencias como lectura o escritura en memoria o como E/S y además diferencia si son de 8 o 16 bits. La AT-DSP2200 devuelve una señal de apoyo al canal de E/S del PC para indicar el tamaño de la transferencia de datos en curso.

2.3.4 Registros De Estado Y Control.

La AT-DSP2200 contiene varios registros de estado, 11 en concreto. Dos registros de 16 bits, Interrup/DMA Control y DMA TC Interrupt Clear, pueden ser usados para programar todas la interrupciones y modos de DMA de la AT-DSP2200 y para habilitar el Procesador digital. Los registros de estado de 16 bits contienen información de las señales de DMA e interrupción. Se pueden usar los otro nueve registros de estado y control para comunicarse con el puerto paralelo del chip DSP. En el capítulo cuatro dedicado a la programación se explicará con detalle cada uno de estos registros. En la tabla 4 se listan los doce registros separados en dos grupos

Tabla 4. Listado de los registros de control y estado

Nombre del registro
Grupo de Registros DSP
Parallel I/O Address Register (PAR). Registro de Direcciones de E/S del Puerto Paralelo.
Parallel I/O Data Register (PDR). Registro de Datos de E/S del Puerto Paralelo.
Error Mask Register (EMR). Registro de Máscaras de Error.

Error Source Register (ESR). Registro de fuentes de Error.
Parallel I/O Control Register Low (PCRl). Registro de Control Bajo de E/S del Puerto Paralelo.
Parallel I/O Interrup Register (PIR). Registro de Interrupciones de E/S del Puerto Paralelo
Parallel I/O Control Register High (PCRh). Registro de Control Alto de E/S del Puerto Paralelo
Parallel I/O Address Register Extended (PARE). Extensión del Registro de Direcciones de E/S del Puerto Paralelo.
Parallel I/O Data Register 2 (PDR2). Registro de Datos 2 de E/S del Puerto Paralelo
Grupo de registros de Interrupciones y DMA
Interrupt/DMA Control Register. Registro de Control de Interrupciones/DMA.
Status Register. Registro de Estado.
DMA TC Interrpt Clear Register. Registro de Borrado de Interrupciones y TC de DMA.

2.3.5 Circuito De Control De Interrupciones

El circuito de control de interrupciones encamina cualquier interrupción habilitada a la línea de interrupción seleccionada. La AT-DSP2200 puede usar una de las ocho líneas de petición de interrupción: IRQ3, IRQ4, IRQ5, IRQ9, IRQ10, IRQ 11 ,IRQ 12 o IRQ 15. Con las peticiones de interrupción , cuyas señales de salida son tri-estado, la AT-DSP2200 puede compartir las líneas de interrupción con otros dispositivos. La AT-DSP2200 genera una interrupción el los siguientes casos:

- Cuando el PDR esta lleno o vacío dependiendo de la dirección seleccionada en Interrup/DMA Control Register por el bit IN*/OUT.
- Cuando el DSP activa una línea de interrupción.
- Cuando se recibe un pulso del Terminal Counter DMA.

Cada una de esas interrupciones es individualmente habilitada o borrada.

2.3.6 Circuito De Control De DMA

Al registro PDR se le puede asignar un canal de DMA para transferencias de datos de 16 bits. Cuando la DMA está habilitada, el bit IN*/OUT esta a nivel bajo (los datos entran en la AT-DSP2200), el bit M/IO* esta a nivel bajo (la AT-DSP2200 se comporta como un dispositivo de E/S para la DMA), y el PDR esta vacío, la AT-DSP2200 envía una petición de DMA. Cuando la DMA está habilitada, el bit IN*/OUT esta a nivel (los datos saldrán desde la AT-DSP2200), el bit M/I* está a nivel bajo (la AT-DSP2200 se comporta como un dispositivo de E/S para la DMA), y el PDR está lleno, la AT-DSP2200 una petición DMA. Los canales de DMA del 5 al 7 del canal de E/S del PC están disponibles para estas transferencias. Si el bit M/IO* se pone a nivel alto (la AT-DSP2200 se comporta como un dispositivo de memoria para la DMA), la AT-DSP2200 no genera petición de DMA, pero responde a los ciclos de reconocimiento de DMA.

2.4 CIRCUITO DE INTERFACE DE MEMORIA Y DSP

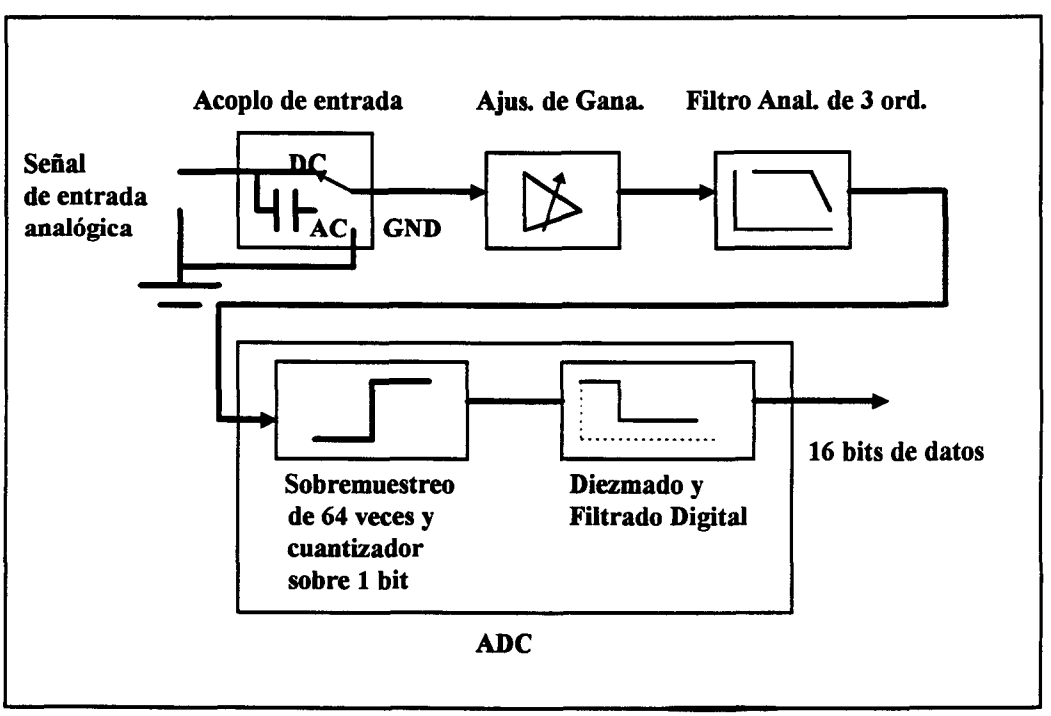
El circuito de interface de memoria y DSP conecta el DSP con la memoria de la placa, con el registro de estado y control que controla y monitoriza circuito de interrupciones del canal de E/S del PC, el circuito analógico de E/S, el circuito de interface del bus RTSI, los LEDs que indican el estado de la placa. En el capítulo

cuatro dedicado a la programación se explicará en detalle el funcionamiento de estos registros. En la figura 2 podemos ver un esquema de este circuito.

2.5 CIRCUITO ANALÓGICO DE ENTRADA

La AT-DSP2200 tiene dos canales de entrada analógicos idénticos por lo que todo lo dicho para uno servirá para el otro. En la figura 7 se muestra el esquema de uno de los canales analógicos de entrada.

Figura 7. Diagrama de bloques del Canal Analógico de Entrada



Los circuitos de entrada de la AT-DSP2200 son capaces de convertir simultáneamente dos señales con un ancho de banda de 22 Khz a señales digitales de 16 bits en complemento a dos. Ambos canales tienen un rango de tensión de entrada de ± 2.828 V o 2 Vrms de fondo de escala. El nivel máximo de señal que se puede

aplicar es ± 20 V , tanto con la placa encendida como apagada, y la impedancia de entrada es de 450 KΩ en paralelo con 65 pF. Cada uno de los canales tiene el acoplamiento seleccionable, la ganacia ajustable, un filtro analógico antialiasing, un convertidor analógico digital ADC con sobremuestreo de 64 veces y un filtro digital antialiasing.

La AT-DSP2200, está diseñada para trabajar en la banda de audio para realizar medidas y procesamiento de señales. Está equipada con tres osciladores de cristal para generar un total de 16 frecuencias de muestreo tal como se indica en la tabla 1. X1=36.864 Mhz es dividido por 768 y por 1.152, produciendo dos bases de tiempos. X2=16.9344MHz y X3=19.6608 Mhz son divididos por 386 para producir otras dos bases de tiempos. Esas bases de tiempos producen las frecuencias de muestreo más rápida de la AT-DSP2200, 51.2 khz y 44.1 Khz, cada una de esas bases de tiempos es dividida a su vez por 2, 4 y 8 para producir doce frecuencias de muestreo más, tal como se ve en la tabla1.

2.5.1 Acoplamiento De La Señal De Entrada.

La AT-DSP2200 tiene un conmutador controlado por software que permite seleccionar el acoplo entre AC o DC. Tal como se ve en la figura 7. Cuando se selecciona acoplo en DC el condensador se elimina del camino de la señal y cuando se selecciona acoplo en AC el condensador se interpone en el camino de la señal. Cuando seleccionamos acoplo DC cualquier señal de offset presente en la fuente de señal llegara hasta el convertidor analógico digital ADC. Esta configuración es la

mejor de las dos pues es la que introduce menos componentes en el camino de la señal y por tanto da mas fidelidad, sin embargo esta configuración sólo es recomendada si la fuente de señal tiene pequeños voltajes de offset, menores de 25mV, o si la fuente de señal ya posee acoplo AC. Si la fuente de señal posee una señal de offset significativamente grande o un voltaje de polarización, se debe seleccionar acoplo en AC para poder aprovechar el margen de tensión de entrada de la tarjeta. Si por ejemplo tenemos un señal de entrada senosoidal 2 V superpuesta a una continua de 1V al final tendríamos una señal variando entre +3V y -1V y como la máxima tensión es de 2.828 V la señal no sería adquirida correctamente produciéndose un recorte. Usar Acoplo AC produce una pérdida en la respuestas a las bajas frecuencias del circuito analógico de entrada debido al condensador. Así la frecuencia de corte de 3dB queda reducida a 180 Hz aproximadamente. Usar acoplo AC además produce unas pérdidas de ganancia totales respecto al acoplo en DC de aproximadamente 0.009 dB. El conmutador software además también conecta la entrada a masa , esta conexión se hace para realizar una calibración de offset.

2.5.2 Ajuste De Ganancia

La ganancia de la AT-DSP2200 es ajustable dentro de un pequeño margen. El procedimiento para ello se describirá en el apartado

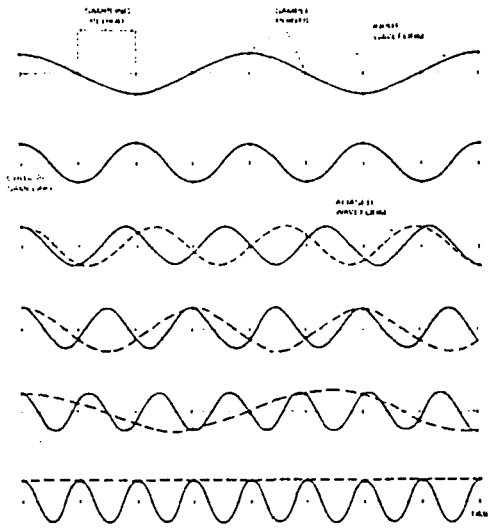
2.5.4 El Aliasing

Antes de hablar del filtro paso bajo de entrada y del ADC es necesario aclarar el concepto de Aliasing.

El Aliasing se produce cuando se incumple el teorema de Nyquist. Este teorema dice que la mayor frecuencia de una señal que vaya a se muestreada por un sistema de adquisición debe ser igual o menor que la mitad de la frecuencia de muestreo. Cuando esto no se cumple aparece el Aliasing. Si la frecuencia de la señal de entrada comienza a aumentar el numero de muestras por ciclo comienza a disminuir, esto se puede observar en la figura 8. Cuando la señal de entrada tiene el valor de la mitad de la frecuencia de muestreo hay sólo dos muestras por ciclo, lo mínimo necesario para poder interpretar una señal bipolar. Si intentamos muestrear señales de frecuencias superiores a la de muestreo el muestreador continuará produciendo muestras a una frecuencia fija lo que hará aparecer una nueva señal de la misma amplitud pero con una frecuencia menor, si la desviación de frecuencias sigue aumentando hacia arriba se van creando nuevas señales de frecuencias cada vez menores. Concretamente, si **Fm** es la frecuencia de muestreo, **Fi** es una frecuencia mayor que **Fm** y **N** es un número entero, las nuevas frecuencias creadas por el efecto Aliasing responden a la fórmula **Fn= ± N*Fm ± Fi.** Por ejemplo si tenemos una señal de frecuencia 36 Khz y la muestreamos a una frecuencia de 44 Khz tenemos que **Fn**=44-36 = 8 Khz, esta señal sería una señal "Alias" de la verdadera señal 36 Khz y sería la que obtendríamos a la salida del muestreador.

Para evitar este efecto es por lo que los circuitos de muestro deben tener a la entrada y a la salida un filtro paso-bajo que limite las señales en banda. El sistema para evitar el Aliasing en la tarjeta AT-DSP2200 se basa en este sistema.

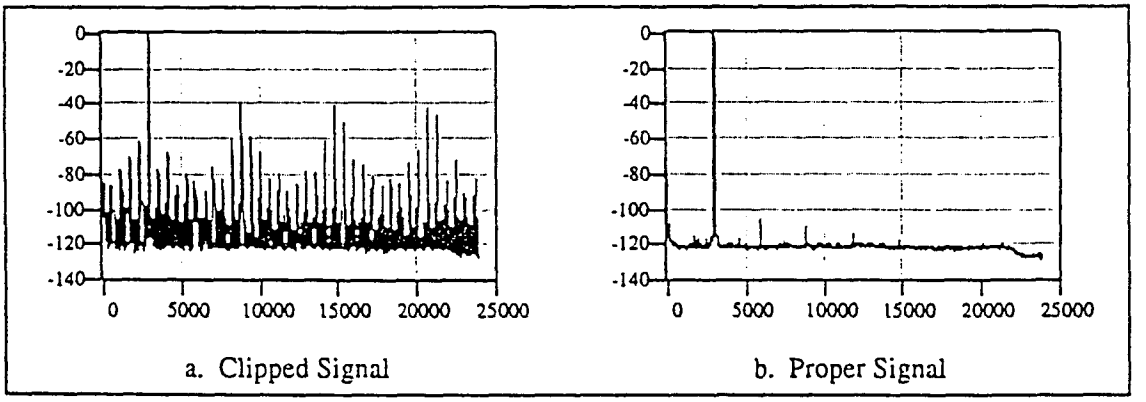
Figura 8. Ejemplo de Aliasing



Hay una forma de Aliasing que los filtros no pueden evitar. Este Aliasing se produce cuando las señales exceden del rango del ADC , a esto se le llama "Clipped ". Cuando ocurre el cliping el ADC asume que el valor entero mas cercano es el rango de la señal, que el caso de la AT-DSP2200 es de -32.768 a +32.767. Como resultado de este fenómeno casi siempre resulta un cambio abrupto en la rampa de la señal y provoca que los datos digitales de salida que representan la señal sean erróneos al tener energía en las altas frecuencias. Esta energía es distribuida a través del espectro de energía. Debido a que el "Cipping" aparece después de los filtros antialiasing, esta energía también esta dentro de la banda base de la señal de entrada. Por suerte el remedio es simple: no permitir que las señales de entrada excedan de 2 Vrms. En la figura 9 se muestra el espectro de una señal senosoidal de 2,962 Khz digitalizada a 48 Khz y con una amplitud de 2.1 Vrms y de 2.0 Vrms. El promedio de la señal a la Distorsión Armónica Total (THD) más el ruido es 35 dB para el caso en que se

produce el "Clipped" y 92 dB para el caso en que la amplitud de la señal es de 2.0 Vrms. Figura 9-a y 9-b.

Figura 9. Señales con Clipped y sin Clipped.



a. Clipped Signal

b. Proper Signal

2.5.4 Filtro Paso Bajo Analógico De Tercer Orden.

EL filtro Analógico está situado antes del muestreador analógico que opera a una frecuencia 64 veces mayor que la frecuencia de muestreo seleccionada, 3.072 Mhz para el caso de un muestreo a 48 KHz,

La AT-DSP2200 incluye dos filtros pasabajos para evitar el aliasing. El primer filtro en el camino de la señal como se ve en la figura 7 es un filtro analógico. Este es un filtro pasabajos Butterword de tercer orden. La frecuencia de corte de este filtro es 80 Khz y el rechazo mayor de 90 dB a 3 Mhz. Debido a que la frecuencia de corte del filtro es significativamente mayor que la mayor de las frecuencias de muestro de los datos el filtro posee una respuesta extremadamente plana en el rango de frecuencias de interés y un error de fase muy pequeño.

Debido a que el ADC muestrea a una frecuencia 64 veces mayor que la frecuencia seleccionada, las componentes de frecuencia por encima de 32 veces la frecuencia de muestreo pueden producir aliasing. El filtro digital rechaza la mayoría de las frecuencias que pueden producir aliasing. Sin embargo el filtro digital no puede hacer nada con las componentes que están próximas a 64 veces la frecuencia de muestreo y sus múltiplos, 128 veces, 256 veces ,etc., ya que el filtro no puede distinguir esas componentes en la banda base. Si por ejemplo la frecuencia de muestreo es de 48 Khz y una señal tiene componentes de 24 Khz , 64 * 48 Khz es 3.072 Mhz, entonces esta señal es enmascarada (aliased) dentro de la banda de paso del filtro digital y no es atenuada. El propósito del filtro analógico entonces es eliminar las componentes próximas a los múltiplos de la frecuencia de sobremuestreo antes de que lleguen al filtro digital así como el ruido. La respuesta del filtro analógico se calcula para producir el máximo rechazo a la frecuencias "alias" mientras que su respuesta en frecuencia sea plana en la banda de paso. Como el filtro es de tercer orden, su caída es mas bien lenta. Esto significa que aunque el filtro tiene un buen rechazo de las frecuencias "alias" en altas frecuencias de muestreo, este no es tan bueno para las bajas frecuencias de muestreo. En la tabla 5 se muestran las frecuencias de muestreo y las de sobre muestreo así como el rechazo del filtro analógico. Los valores de rechazo del filtro son solo aplicables cerca de las frecuencias de sobremuestreo para una frecuencia de muestreo dada. Para frecuencias que no estén próximas a múltiplos de las frecuencias de sobremuestreo el rechazo es mayor de -85 dB.

Tabla 5. Frecuencias y rechazo de filtro digital.

Frecuencia de Muestreo	Frecuencia de Sobremuestreo	Rechazo del Filtro
51.2 Khz	3.2768 Mhz	-97 dB
48.0 Khz	3.072 Mhz	-95 dB
44.1 Khz	2.8224 Mhz	-93 dB
32.0 Khz	2.048 Mhz	-84 dB
25.6 Khz	1.6384 Mhz	-79 dB
24.0 Khz	1.536 Mhz	-77 dB
22.05 Khz	1.4112 Mhz	-75 dB
16.0 Khz	1.024 Mhz	-66 dB
12.8 Khz	0.8192 Mhz	-60 dB
12.0 Khz	0.768 Mhz	-59 dB
11.025 Khz	0.7056 Mhz	-57 dB
8 Khz	0.512 Mhz	-48 dB
6.4 Khz	0.4096 Mhz	-42 dB
6 Khz	0.384 Mhz	-41 dB
5.5125 Khz	0.3528 Mhz	-39 dB
4 Khz	0.256 Mhz	-30 dB

2.5.5. El Conversor Analógico Digital (ADC)

Tres son las técnicas que tradicionalmente se han empleado para la conversión A/D: aproximaciones sucesivas, integración y flash. Estas técnicas han sido

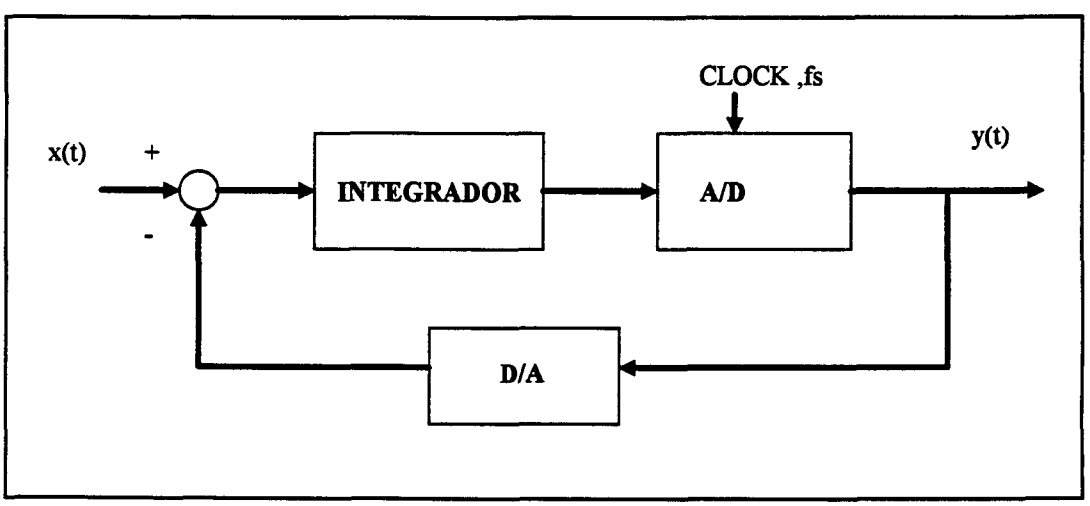
implementadas según el criterio de Nyquist-Shannon . Últimamente se ha introducido otra técnica denominada modulación Delta-Sigma. Esta forma de conversión realmente no es nueva pues tiene más de veinte años, pero hasta ahora no se había podido realizar de forma comercial por dificultades técnicas debido alto grado de integración que requiere. Esta técnica sin embargo tiene claras ventajas con respecto a las anteriores, especialmente para la digitalización de señales de audio. Una de las ventajas principales es que al usar un convertidor D/A como referencia interna está libre de errores de linealidad diferencial (DNL) que es una característica inherente al resto de los convertidores y que afecta especialmente a las señales de con niveles muy bajos en las cuales el ruido y la distorsión afectan directamente a la DNL. Por otro lado en los sistemas PCM lineales un error el bit mas significativo provoca una gran discontinuidad mientras que en este sistema como no hay bit MSB este error desaparece.

La Modulación Delta-Sigma pertenece a los sistemas de codificación diferencial y es una forma de PCM (*Pulse Code Modulation*) diferencial que lleva el método al extremo, empleando una frecuencia muy alta , 64 veces la de muestreo en este caso, y una codificación con un sólo bit.

Un convertidor Delta-Sigma está compuesto por tres bloques fundamentales: un amplificador diferencial, un integrador y una etapa de salida compuesta por un filtro digital. Su principio de funcionamiento se basa en el de un convertidor A/D de un bit insertado en un bucle analógico de realimentación negativa con una ganancia en lazo abierto muy grande, figura 11. El convertidor Delta-Sigma muestrea su entrada

a una frecuencia varias veces mayor que la mayor frecuencia a digitalizar. En el bucle de realimentación se encuentra un conversor D/A y a su salida obtenemos un valor mas o menos igual al fondo de escala. Esta señal se resta o suma al valor actual de la entrada analógica y se aplica el resultado al integrador. Este integrador actúa como un acumulador analógico, añadiendo su nueva tensión del nodo V1 a la primitiva tensión del nodo V2 para obtener el nuevo valor V2. EL conversor A/D, que es en realidad un comparador, compara este nuevo valor con la tensión de referencia y da un valor comprendido entre uno o cero dependiendo del resultado de la comparación. Esta operación se repite cada ciclo de reloj, dando como resultado un flujo continuo de bits cuyo valor medio sigue a la tensión analógica de entrada del convertidor.

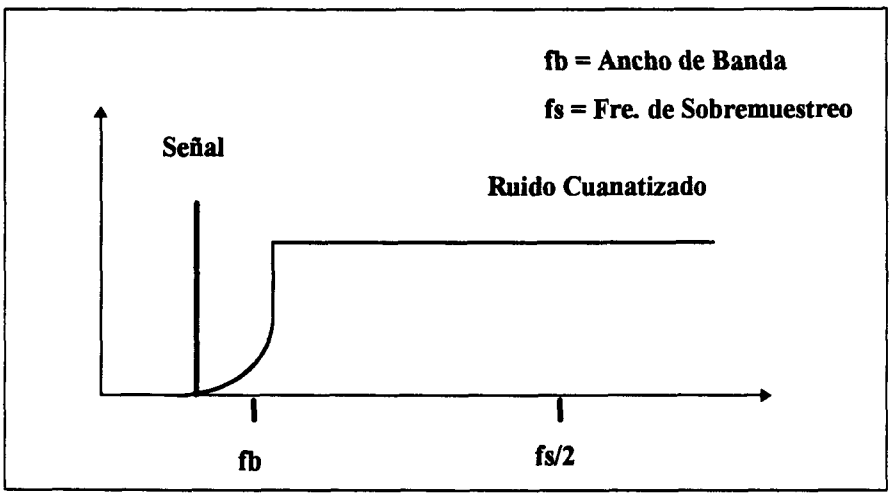
Figura 11. Diagrama de bloque de un convertidor Delta-Sigma



La cuantización de la señal en un solo bit introduce un error que podemos aproximar a una señal de ruido. Este ruido es de gran importancia y debe procurarse su eliminación ya que puede introducir una gran distorsión en la señal. Afortunadamente en los conversores Delta-Sigma este problema se soluciona por el sobremuestreo al que se somete la señal. El sobremuestreo ensancha el ancho de

banda de la señal, entonces el ruido al que nos referíamos antes, llamado normalmente ruido de cuantificación, también se distribuye a lo largo de la nueva banda de paso y llega a ser, por consiguiente, bastante menor en la banda de paso de la señal. En el caso del convertidor CS-5327-KP, con el que está equipada la AT-DSP2200 el factor de sobremuestreo es de 64 veces, así para una frecuencia de muestreo de datos de 48 Khz la frecuencia de sobremuestreo es de 3.072 Mhz, el ruido de cuantificación se reparte en una banda de paso 64 veces más grande, por la tanto el ruido residual dentro de la banda de audio es 64 veces menor al del valor sin sobremuestreo. En la figura 12 podemos ver el proceso de la señal al pasar por el conversor y como el ruido se encuentra en su mayor parte fuera de la banda de audio.

Figura 12. Salida del Modulador Delta-Sigma



La salida del modulador Delta Sigma se lleva a una etapa de filtrado diezmado, figura 13, que se encarga de eliminar todas las componentes de frecuencia fuera de la banda de interés. El diezmado incluye dos etapas de filtros FIR, FIR 1 y FIR 2, la primera misión de estas dos etapas es atenuar el ruido de cuantización, para ello FIR 1 y FIR 2 usan coeficientes de 17 y 18 bits y sus ordenes son de 27 y 28

respectivamente, además los datos son procesados con aritmética de punto fijo de 18 bits. El tercer filtro, FIR 3, conforma la banda de paso y atenúa las señales fuera de esta. Los errores de respuesta en frecuencia de la banda de paso introducidos por el modulador, el primero y segundo filtro también son corregidos por FIR 3. El rizado en la banda de paso se reduce a ± 0.001 dB desde tensión continua hasta 22 Khz. En este filtro los datos son también procesados con aritmética de 18 bits de punto fijo. A la salida los datos son truncados a 16 bits lo que contribuye a mejorar la calidad del sistema.

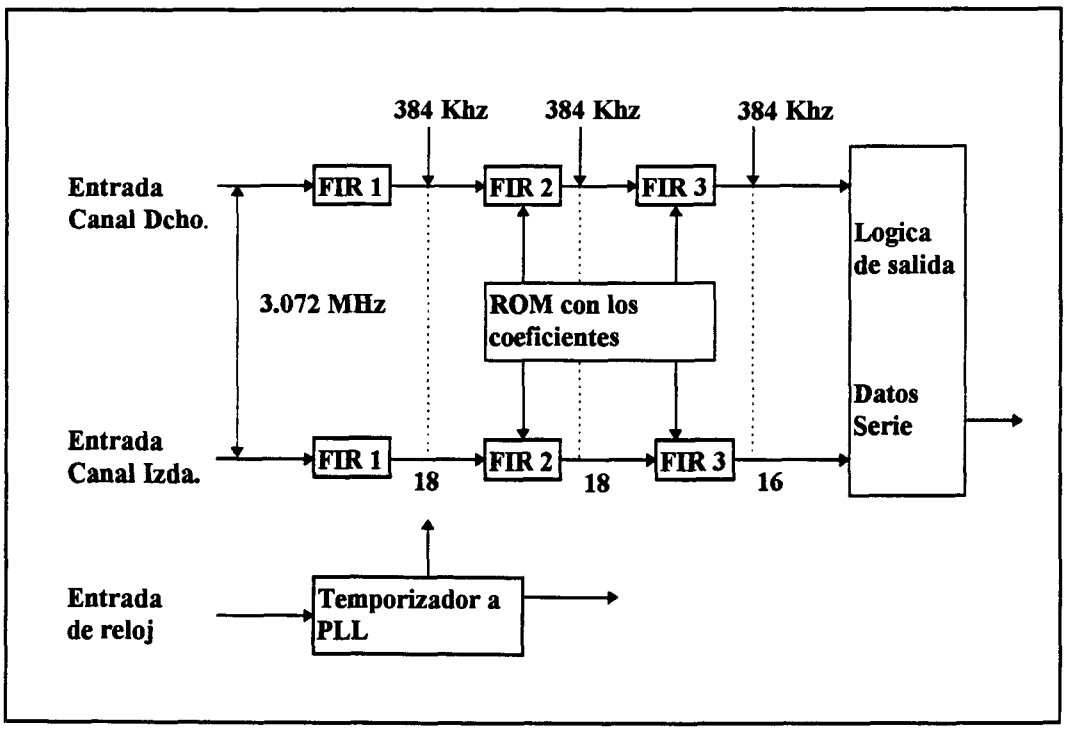
Además de lo mencionado antes FIR 1, FIR 2 y FIR 3 se combinan para proporcionar un filtrado antialising. Todas las frecuencias de entrada de 26 Khz a 3.2768 Mhz son atenuadas -85 dB.

El rango dinámico de la señal de salida del conjunto que forma el modulador Delta-Sigma es de 93 dB. El rango dinámico de un circuito es la relación entre el valor de la señal mayor y el ruido residual en ausencia de señal. En un sistema de 16 bits bipolar, la máxima señal se codifica entre los valores +32.767 y -32.768. Así para señal senosoidal el valor rms viene dado por 32.768/1.414 =23.170,475 bits menos significativos. Un canal de la AT-DSP2200 puesto a masa tiene un nivel de ruido rms de 0.5 bits menos significativos. La relación entre estas dos magnitudes es 23.170,475/0.5=46,341 o expresado en dB 93.3 dB de rango dinámico.

La salida del ADC es un flujo de datos serie empaquetados en 32 bits en complemento a dos, el canal 0 son los 16 bits mas significativos y el canal 1 los 16

bits menos significativos. 0V son representados como 0000 hex y, +2.828 corresponden a 7FFF hex (+32767) y -2.828 corresponden a 8000 hex (-32.768). Por tanto el valor máximo a fondo de escala del circuito de entrada es 2 V rms o 5.657 V pico a pico.

Figura 13. Diagrama de bloques de la etapa de filtrado y diezmado del ADC

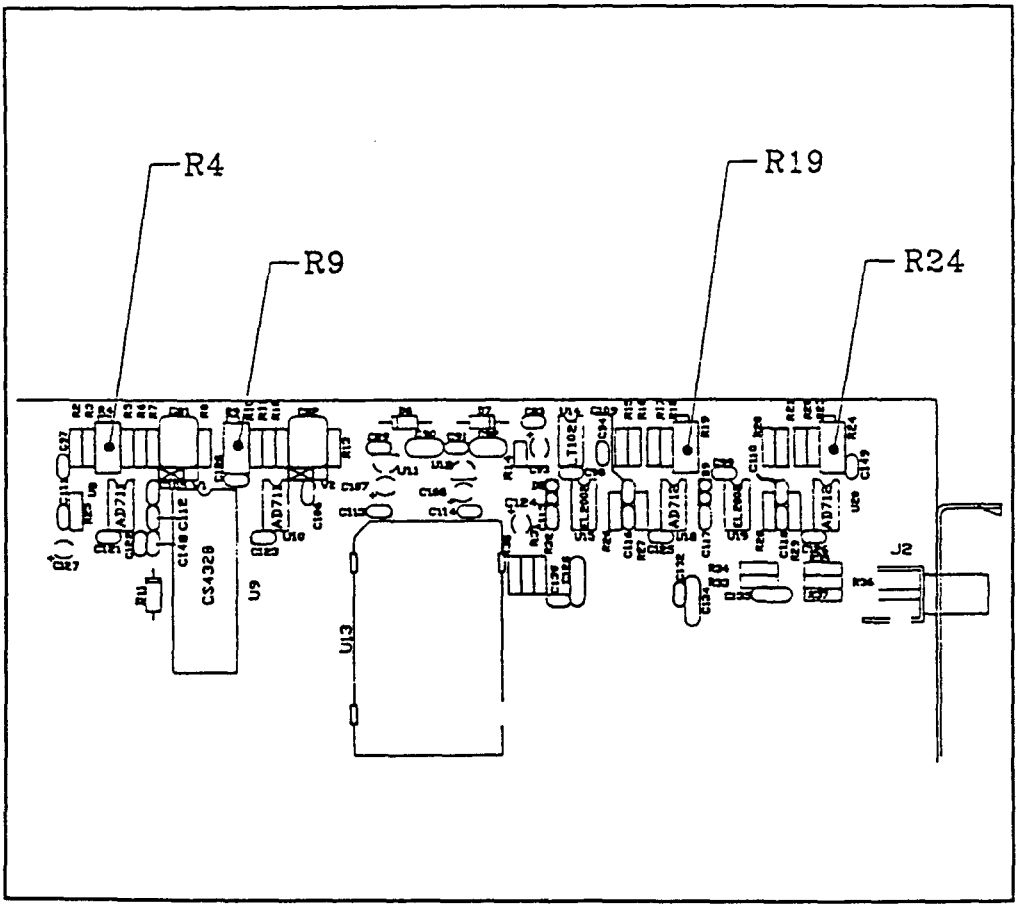


2.6 CALIBRACIÓN DE LOS CANALES ANALÓGICOS DE ENTRADA

La AT-DSP2200 tiene unos ajustes para calibrar las entradas analógicas tal como se ve en la figura 14, estos ajustes están señalados como R19 y R24. El offset de cada canal es digitalmente anulado. La ganancia de cada canal es ajustada moviendo los trimers señalados antes de forma que R24 ajusta la ganancia del canal 0

y R19 ajusta la ganancia del canal 1. Estos trimers un rango de ajuste de ganancia por canal de ± 3dB.

Figura 14. Trimers de calibración.



2.6.1 Equipos de Calibración

Para mejorar los resultados de las medidas, la AT-DSP2200 debería ser calibrada de forma que la precisión absoluta para señales AC esté dentro de el margen de ± 0.001 dB. Además de acuerdo a la norma, el equipo de calibración usado deberá

ser 10 veces mas preciso que ± 0.001 dB (≅ ± 0.01 %). Sin embargo se puede considerar aceptable equipos de calibración con dos veces esa precisión para realizar un calibrado para sistemas de adquisición, es decir con una precisión de ± 0.005 dB (≅ ± 0.05 %).

Las herramientas para calibrar la AT-DSP2200 son las siguiente:

- Una fuente de voltaje de tensión continua variable con la siguientes características.
- Precisión ± 250 μV según el estándar o ± 1.25 mV.
- Rango ± 3V.
- Resolución 100 μV.

Hasta ahora sólo hemos hablado de la calibración en continua por ser la más sencilla. La calibración en alterna requiere equipos mas caros y sofisticados difíciles de encontrar si no es en laboratorios de especializados en calibración, así por ejemplo encontrar una fuente de señal alterna y un voltímetro con una precisión de ± 0.01 dB no es nada fácil. Además la calibración con señales alternas, requiere que sea tomada una secuencia de datos muestreados y se calcule su desviación estándar. Es por esto que esta calibración no se recomienda al usuario y no viene documentada.

2.6.2 Procedimientos de Calibración de los Canales de Entrada Analógicos

Antes de proceder a la calibración de la AT-DSP2200 debe mantenerse encendido el PC con la tarjeta conectada durante al menos 10 minutos, pues el offset

y la ganancia tienen un cierta deriva con la temperatura, de esta forma es necesario que primero se estabilice la temperatura. Los ADC después del periodo de calibración de offset miden y almacenan el valor del offset en un registro interno luego restan ese valor a todas las muestras de entrada. El procedimiento a seguir para llevar a cabo una calibración es el siguiente.

- - Calibrar el offset.
- - Calibrar la ganancia.

La calibración de los canales analógicos de entrada de la AT-DSP2200 se realiza actuando sobre dos elementos fundamentalmente; el offset y la ganancia. El offset para cada canal es anulado digitalmente. En cuanto a la ganancia se puede variar en un margen de ± 6 % (± 0.5 dB) para cada canal

2.6.2.1 Calibración del Offset

El offset debe calibrarse siempre primero pues la calibración de la ganancia depende del offset. El offset es anulado digitalmente de forma automática en lugar de hacerlo con potenciometros, como la ganancia, lo que sin duda da una mayor precisión. La calibración se puede hacer de dos formas, usando las librerías NI-DAQ o de forma manual creando una rutina para ello, a continuación se explicará ambas formas.

La calibración programando directamente los registros es mas engorrosa pero es imprescindible si no se dispone de las librerías NI-DAQ. El resultado final que producen y la forma interna en que se realiza la calibración es la misma. A continuación se muestra el algoritmo para la calibración.

1. Escribir la frecuencia de muestreo en el Registro de Configuración de E/S Analógicas.
2. Poner el bit AICAL* del Registro de Configuración de E/S Analógicas a nivel alto o bajo dependiendo de la referencia elegida para el offset. Si la calibración del offset se realiza tomando como referencia la señal AICAL* debe ponerse a nivel bajo, pero si por el contrario se usa una referencia externa, el bit AICAL* debe ser puesto a nivel alto.
3. Poner el bit DPD del Registro de Configuración de E/S Analógicas a nivel alto.
   Poner el bit DPD del Registro de Configuración de E/S Analógicas a nivel bajo. Esto hace que se inicie un ciclo de calibración. Como ya dijimos antes la calibración del offset dura 4096 intervalos de muestreo (85.3 ms a 48 Khz) durante el cual el bit Dcal del Registro de estado es puesto a nivel alto. Este bit pasa a nivel bajo cuando termina la calibración, de esta forma este bit puede ser usado para testear el fin de la calibración.
4. Cuando la calibración ha terminado hay que poner el bit AICAL* a nivel alto, esto hace que el conmutador de entrada se conecte a las señales externas.

El funcionamiento normal del ADC no comienza inmediatamente después de la calibración del offset, pues el filtro digital toma 72 intervalos de muestro como

periodo de asentamiento, así pues, el primer dato válido considerando una frecuencia de muestro de 48 Khz no estará disponible hasta 86,8 ms después de que comience la calibración del offset.

Si se dispone de las librerías NI-DAQ, la rutina DSP2200_Calibrate que pertenece al grupo de funciones de calibración y configuración facilitará mucho el proceso de calibración. Esta rutina lleva acabo una calibración del offset de los canales analógicos de entrada o salida de la AT-DSP2200. El capítulo dedicado a la programación se explicará con detalle el funcionamiento de esta función.

Ahora mostrare un listado escrito en C y que usa la función DSP2200_Calibrate para realizar la calibración.

```
#include <stdio.h>
#include "nidaqdos\msc_ex\nidaq.h" /* fichero de definición de funciones*/
#include "nidaqdos\msc_ex\nidaqerr.h" /*fichero de definición de códigos de error*/
#define SLOT 1 /*indica el slot donde se conecta la AT-DSP2200*/
void main(void)
{
 int modo,RefADC, estado;
modo =3; /*calibración de los canales de entrada y salida*/
RefADC=0; /* usa como referencia la masa analogica*/
estado=DSP2200_Calibrate(SLOT,modo,RefADC);
if (estado==0)
```

```
        printf("La calibración se ha producido correctamente\n");
else
        printf("Se ha producido un error con valor %d consulte el manual\n");
}
```

Una vez ejecutado el programa se inicializa el ADC y comienza un ciclo de calibración. Este ciclo dura 4096 periodos de intervalos de muestreo (128 ms a 32 Khz).

T(sg) = [(1/fs)* Tp]

fs = Frecuencia de muestreo.

Tp = Periodos de intervalo de muestreo.

2.6.2.2 Calibración de la Ganancia.

El ajuste de la ganancia de cada canal de entrada se ajusta aplicando una tensión de entrada. El valor de dicho voltaje debe ser de 2.5 V, que corresponde con una lectura del ADC de 28.963 o 7123 hex. Los pasos para un correcto ajuste son los siguientes.

1. Conectar la fuente de voltaje de 2.5 V al canal que se quiera ajustar la ganancia.
2. Seleccionar acoplamiento DC. Esto se puede hacer usando la rutina MAI_Coupling de la librería NI-DAQ o programando directamente el Registro Configuración de E/S Analógico, poniendo a nivel bajo el bit AC/DC*. Si no se ha

modificado el acoplamiento no es necesario hacer esto puesto que la configuración por defecto de la AT-DSP2200 incluye acoplo en DC.

3. Medir las tensiones de entrada del canal que queremos ajustar y ajustar el trimer adecuado hasta que se mida un valor de 28.963 o 2.5 V.

A continuación se muestran dos ejemplos de programación, mediante programación directa de los registros. y usando las librerías NI-DAQ.

Para programar los registros para el ajuste de la ganancia sin usar las librerías NI-DAQ es necesario disponer de la herramienta llamada Devolp Tools Kit, que contiene con un ensamblador y un linkador. Así aquí sólo señalaremos los puntos básicos que hay que seguir para realizar es calibración.

1. Configurar el puerto de entrada serie.

Los ADC de la AT-DSP2200 están continuamente muestreando a la frecuencia programada en los bits CLKAD<3..0> del Registro de Configuración de E/S Analógico. Para configurar el puerto serie hay que escribir el dato 30CC0 hex en el registro IOC. Esto configura el puerto serie de E/S para 32 bits de datos, el primer dato es el MSB, sin DMA y usando un Clocks externo.

2. Leer el resultado la conversión.

El resultado de la conversión es obtenido el registro IBUF. Antes de este registro hay que testear el estado del buffer para ver si se ha completado una

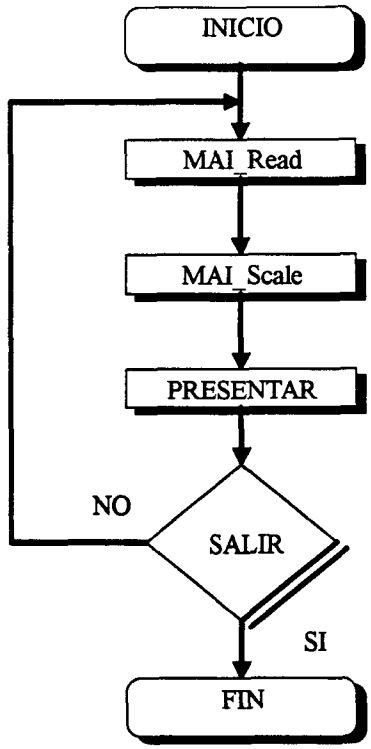
conversión. Para comprobar si se ha completado una conversión hay que hacer lo siguiente.

a.- Testear el flag IBE.

b.- Si el flag IBE es uno, se repite lo anterior, si el IBE es cero leer IBUF para obtener el resultado.

Los datos de entrada son almacenados en el registro IBUF, que es un registro de 32 bits, la parte alta del registro (16 bits mas significativos) contiene el dato del canal cero ACH0 y la parte baja (16 bits menos significativos) el canal uno ACH1. Cada lectura en el IBUF borra el contenido de este preparándolo para la siguiente lectura.

Ahora veremos el mismo proceso , ajuste de la ganancia de los canales, pero usando las librerías NI-DAQ. Como ya dijimos en el apartado de Ajuste do offset el método elegido es independiente del resultado final. Se mostrará a continuación un programa escrito en C que hace uso de las librerías NI-DAQ para ajustar la ganancia.En la página siguiente podemos ver el diagrama de flujo de este procedimiento.



Como se observa en el diagrama de flujo no se actúa sobre la configuración de la AT-DSP2200, esto es así puesto que se toman los valores de configuración por defecto que son:

- Acoplo en DC en los dos canales de entrada.
- Selección de los dos canales de entrada.
- Frecuencia de muestreo 32 Khz.

La explicación detallada de las funciones que intervienen en el programa que se lista a continuación se realiza en el capítulo cuatro.

Código del programa en C.

```
#include <stdio.h>
#include "nidaq.h"      /* definición de funciones */
#include "nidaqerr.h"  /* definición de codigos de  codigos de error*/
#define  AC 0          /* NI-DAQ DOS value for AC coupling */
#define  DC 1          /* NI-DAQ DOS value for DC coupling */
#define SLOT 1         /*indica el numero de slot donde está la placa*/

void main(void)
{
int err0,DatosLeidos[2];
double voltios[2];
  printf("\nEste programa realiza una medida cada vez que se pulsa una tecla");
  printf("\nPulse ESC para terminar");

  printf("\n\n\t\t  CH0     CH1\n\t\t");
  do
  {
    err0 = MAI_Read (SLOT, DatosLeidos);
    if (err0) break;
    err1 = MAI_Scale (SLOT, 1L, DatosLeidos, voltios); /*realiza la conversión a
                                                                    voltios*/
    if (err1) break;
    for (i=0, j=0; i<2; i++)
```

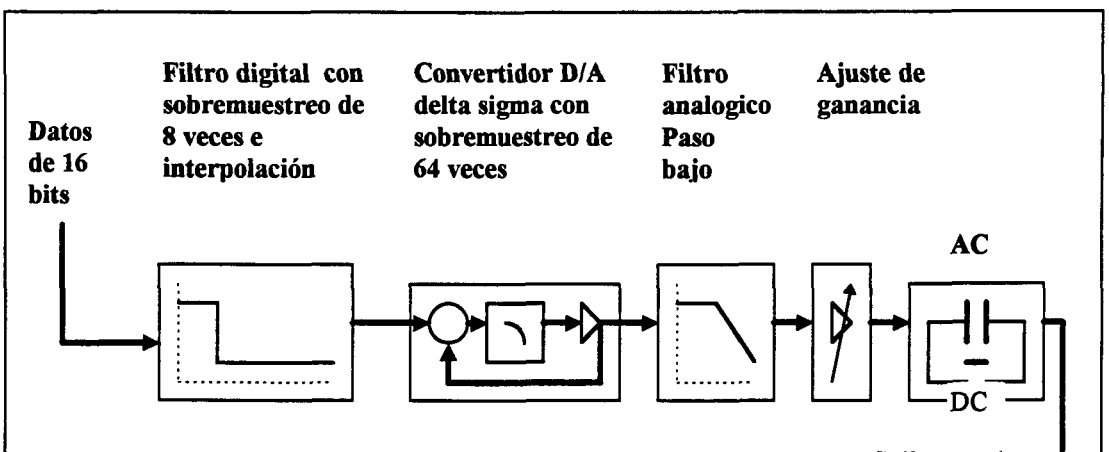
```
    {
      if (canales[j] == i)
      {
        printf("%+4.3f ",voltios[j]);
        j++;
      }
      else
        printf(" ----- ");
    }
    for (i=0; i<2; i++) printf("\b\b\b\b\b\b\b\b");
  }
 while (getch() != 27);
}
```

2.7 CIRCUITO ANALÓGICO DE SALIDA

La AT-DSP2200 tiene dos canales analógicos de salida. El circuito que forma las salidas analógicas, figura 15, convierte simultáneamente los 16 bits de datos en complemento a dos de cada canal a una señal analógica con ancho de banda limitado. El nivel máximo de ambos canales tiene un rango de ± 2.828 V o 2 V rms. Las frecuencias de muestreo y sus bases de tiempos están listados en la tabla 1 y son las mismas que la de los canales analógicos de entrada. Cada canal de salida tiene un filtrado anti-imagen formado por dos filtros, uno analógico y otro digital, un circuito

de ajuste de ganancia y unos puentes para seleccionar el acoplamiento entre AC o DC.

Figura 15. Diagrama de bloques del circuito analógico de salida

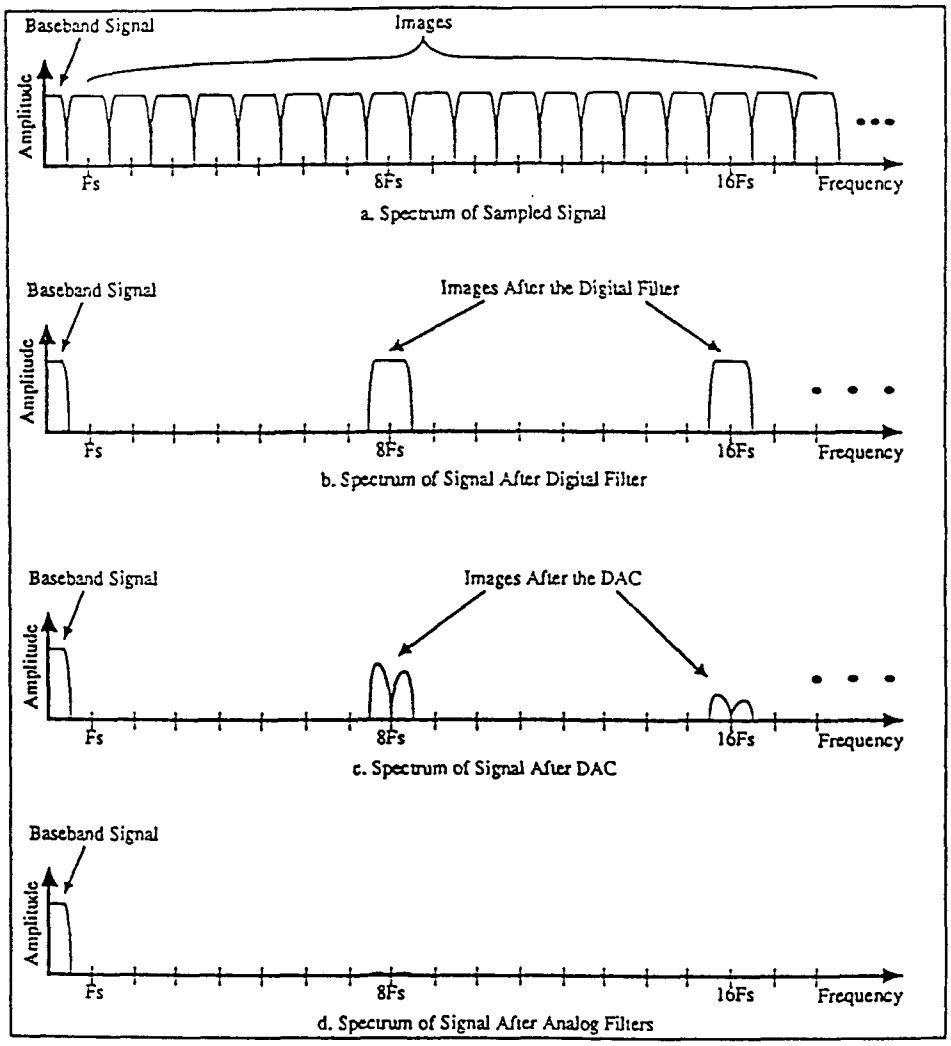


2.7.1 Filtrado Antiimagen de la Señales de Salida, Parte Analógica Y Digital

Una señal muestreada se repite a lo largo del espectro de frecuencia. Esa repetición comienza a partir de la mitad superior de la frecuencia de muestreo (Fs) y , al menos en teoría continua a través del espectro hasta el infinito, tal como se muestra en la figura 16a. Esta repetición de la frecuencia de muestreo crea una serie de armónicos o frecuencias imágenes en los múltiplos enteros de Fs con dos bandas laterales. Este espectro sin filtrar no debe aplicarse a un equipo de audio. Aunque las frecuencias superiores a 20 Khz están fuera del margen audible, producirían el bloqueo del equipo se conecte a continuación de la AT-DSP2200, como por ejemplo un amplificador, y daría lugar a productos de intermodulación que si serían audibles.

Para evitar todo esto la AT-DSP2200 lleva acabo un filtrado de las frecuencias imágenes en dos etapas usando para ello primero un filtro digital y luego un filtro analógico. Primero los datos son digitalmente remuestrados a ocho veces su valor original, interpolación . Después, un filtro digital de fase lineal elimina la mayoría de las frecuencias que están sobre la mitad de la frecuencia original y envía los datos a ocho veces su frecuencia al convertidor digital analógico (DAC),tal como se ve en la figura 16b. El proceso del DAC realiza un filtrado suplementario debido a la retención que se produce en este. Este filtrado tiene una respuesta como "sin x /x", que tiene ceros en las frecuencias múltiplos de ocho veces la de muestreo, figura 16c. Por último el filtro analógico situado después del DAC se en carga de eliminar los restos de las frecuencias imágenes figura 16d. El filtrado analógico se realiza con dos filtros analógicos. El primero es un filtro analógico de capacidad conmutada de quinto orden y su frecuencia de corte se escala con la frecuencia de muestreo con una relación aproximada de 0.52 veces la frecuencia de muestreo. Este filtro tiene una respuesta Butterword en el cuarto polo y un polo extra en 1.04 veces la frecuencia de muestreo aproximadamente. El segundo filtro es un Butterword de segundo orden con una frecuencia fija de corte de 80 Khz . La función principal de ese filtro es eliminar las frecuencias imágenes del filtro de capacidad conmutada con sobremuestreo de 64 veces.

Figura 16. Espectro de la señal a la salida del DAC



2.7.2 El Convertidor D/A Delta Sigma

El convertidor Delta-Sigma de la AT-DSP2200 funciona de la misma forma que el ADC sólo que de forma inversa. Los datos digitales pasan a través del filtro digital, como se dijo en el apartado anterior, y luego "entran" en el modulador Delta-Sigma. En el ADC el modulador Delta-Sigma es un circuito analógico que convierte

señales analógicas de alta resolución a datos digitales de alta velocidad de un bit, mientras que en el DAC el modulador Delta-Sigma es un circuito digital que convierte datos digitales de alta resolución a datos digitales de un bit de alta velocidad. Como en el ADC, el modulador traslada el ruido de cuantización fuera de la banda de la señal. Los datos digitales pasan después a un DAC. este DAC solo puede tener dos valores analógicos de forma que es totalmente lineal. La salida de DAC tiene un gran cantidad de ruido de cuantización en las altas frecuencias que como se describió antes está cerca de los múltiplos de ocho veces la frecuencia de muestreo. Los dos filtros analógicos eliminan este ruido de cuantización.

2.7.3 Procedimientos de Calibración de los Canales de Salida Analógicos

Antes de proceder a la calibración de la AT-DSP2200 debe mantenerse encendido el PC con la tarjeta conectada durante al menos 10 minutos, pues el offset y la ganancia tienen un cierta deriva con la temperatura, de esta forma es necesario que primero se estabilice la temperatura. El procedimiento a seguir para llevar a cabo una calibración es el siguiente.

- Calibrar el offset.
- Calibrar la ganancia.

La calibración de los canales analógicos de salida de la AT-DSP2200 se realiza actuando sobre dos elementos fundamentalmente; el offset y la ganancia. El offset

para cada canal es anulado digitalmente. En cuanto a la ganancia se puede variar en un margen de ± 6 % (± 0.5 dB) para cada canal

2.7.3.1 Calibración del Offset

El offset debe calibrarse siempre antes de la ganancia pues este depende de la misma. Como ya dijimos en el apartado anterior el offset es anulado digitalmente en lugar de usar potenciometros, la forma de hacerlo es la siguiente. Cuando comienza un ciclo de calibración mediante una orden software, independientemente de método elegido, el filtro digital y el modulador Delta-Sigma se resetean, el reloj del convertidor D/A es sincronizado con las señales de control internas del DAC, y el ADC fuerza las salidas analógicas a cero, corrigiendo internamente cualquier error de offset para las futuras señales de salida. Para anular el offset en los canales analógicos de salida, igual que con los canales analógicos de entrada, se puede usar las librerías NI-DAQ o escribir directamente en los registros. El programa en C mostrado en el anteriormente sirve para calibrar tanto los canales de entrada como los de salida. Para realizar la calibración escribiendo directamente en los registros hay que seguir los siguientes pasos.

1. Seleccionar la frecuencia escribiendo en el Registro de E/S Analógicas.
2. Poner a nivel alto el bit AOCAL del Registro de Configuración de E/S Analógicas.
3. Poner a nivel bajo el bit AOCAL del Registro de configuración de E/S Analógicas. Esto inicia un ciclo de calibración. La calibración del offset tarda 1024 intervalos de muestreo (21.3 ms a 48 Khz) durante el cual el bit DCal del registro de estado

se pone a nivel alto. El bit Dcal se pone a nivel bajo al terminar el ciclo de calibración. de forma que puede ser usado como bit de bandera para ver cuando ha terminado la calibración.

La operación normal del DAC comienza normalmente 72 intervalos de muestreo después de ciclo de calibración (1.5 ms a 48 Khz), por lo que el primer dato no estará disponible hasta después de 22.8 ms después de la calibración. El retardo de 1.5 ms es debido al tiempo de asentamiento del filtro digital.

Los Canales de salida de la AT-DSP2200 tienen una característica llamado "mutue", silenciamiento, de la que no habíamos hablado hasta ahora. Los dos DACs de los canales salida pasan al modo mute si reciben 4096 ceros consecutivos. En modo mute las salidas son fijadas a masa y el ruido cae de 92 dB a 120 dB del fondo de escala. Desde que se reciba un dato distinto de cero, el DAC instantáneamente comienza su operación normal. El modo mute se diseñó para eliminar el ruido de fondo cuando no se genera ninguna forma de onda. El modo mute hace, sin embargo, que haya una ligera diferencia entre el offset normal cuando escribimos ceros y el offset del mute. Como resultado el DAC tiene dos offset, uno para el modo mute y otro en operación normal. La diferencia de ambos es normalmente menor de 1 mV.

2.7.3.2 Calibración de la ganancia.

La ganancia de los canales analógicos de salida de la AT-DSP2200 se puede ajustar de forma independiente con una variación de ± 0.5 dB. Para realiazar este ajuste hay que escribir determinados valores en el convertidor D/A y medir los

resultados con una herramienta adecuada, como ya se habló en el apartado referente a la calibración de los canales de entrada. El ajuste se hace mediante los potenciometros R4 y R9 señalados en la figura 14 y tal como se describe en la tabla 6. Los Pasos seguir son los siguientes.

Tabla 6.

Canal	**Potenciometro**
0	R4
1	R9

1. Conectar un voltímetro a la salida del canal que se quiera ajustar.
2. Seleccionar acoplo en DC.
3. Escribir el valor 0 (0000 hex) en el DAC que va ser calibrado y 1 (0001 hex) en el otro DAC. Cuando se escribe, siempre se hace en ambos DAC a la vez. Al canal izquierdo van los 16 bits de mayor peso y al canal derecho los 16 bits de menor peso de la palabra de 32 bits. Así si estamos calibrando el canal izquierdo, escribimos 00000001 hex, y si estamos calibrando el canal derecho, escribimos 00010000 hex. No debemos escribir 0 en los dos canales cuando calibramos la ganancia, pues el DAC automáticamente después de 4096 periodos de muestreo en los que haya ceros en ambos canales pasa a "mute", silencia la salida, y esto produce un ligero offset.
4. Leer el voltímetro y apuntar el resultado. El voltaje debe estar comprendido entre 0V y 3 mV. si no es así ajustar otra vez el offset.

4. Escribir el valor 28963 (7123 hex) en el DAC del canal que va ser calibrado y cualquier otro valor en el otro canal. Ajustar los potenciometros hasta que en el voltímetro se lea 2.5000 V mas el valor del resultado del paso anterior.

El proceso descrito anteriormente se puede hacer de dos maneras con ayuda de las librerías NI-DAQ, o programando la AT-DSP2200 para usar los canales de salida analógicos. Para hacerlo de esta última forma es necesario disponer del paquete de software adicional llamado Develop Tools Kit. Ahora veremos un ejemplo de como hacer el ajuste de ganancia haciendo uso de las librerías NI-DAQ y luego un algoritmo de como se hace programando la AT-DSP2200 con el Devloop Tools Kit.

Programa que permite la calibración de la Ganancia de los canales analógicos de salida usando la librerías NI-DAQ.

```
/* Este programa permite introducir un valor por el teclado y sacar su equivalente en
*/
/* tensión por el canal seleccionado 0-1*/
#include <stdio.h>
#include "nidaq.h"
#include "nidaqerr.h"
void main(void)
{
 int boar_code=18,volts0, volts1 ;
 Init_DA_Brds(1,&boar_code); /*inicializa la AT-DSP2200*/
 printf("\nEscriba en el canal que desee calibrar la ganancia y ponga el otro a cero :");
 printf("\nIntroduzca el voltaje de salida, valor digital, para el canal cero: ");
```

```
scanf("%d",&volts0);
printf("\nIntroduzca el voltaje de salida,valor digital, para el canal uno: ");
scanf("%d",&volts1);
DSP2200_Calibrate(1,2,0); /*Ejecuta la calibración de offset*/
AO_Write(1,0,volts0);  /*escribe en el canal cero*/
AO_Write(1,1,volts1);  /*escribe en el canal uno*/
}
```

Algoritmo para escribir en los canales analógicos de salida un dato para la calibración de la ganancia:

En general, los DACs de la AT-DSP2200 deberían se actualizados continuamente a la frecuencia especificada por los bits CLKDA<3..0> del registro de configuración de E/S analógicas. En los casos en los que solo se desee escribir un solo valor a los DACs, como en este caso, es preciso que mantengan el valor de tensión para poder ser medido. Si se escribe un dato en el buffer del puerto serie del chip DSP, el dato es repetidamente enviado a los DACs por el puerto serie hasta que se escribe un nuevo valor. De esta forma podemos medir el dato y realizar el ajuste de la ganancia. Los pasos a seguir son los siguientes:

1. Configurar el puerto serie. Para ello hay que escribir 30CC0 (en hex) en el registro de control de E/S, IOC. Esta palabra configura el puerto serie de E/S para 32 bits de datos sin DMA y usando reloj externo.
2. Escribir el dato en el conversor D/A. Para ello hay que escribir el número que queramos convertir en el registro OBUF, pero antes hay que testear es estado de

ese registro para saber cuando esta el buffer listo para aceptar un nuevo dato, esto se hace de la siguiente forma:

- Testear el flag OBF.
- Si el flag IBF esta a uno ir al paso anterior. Sino leer el registro para obtener el resultado.

2.7.4 Acoplo AC/DC y Circuito De Salida.

Después del filtro analógico la AT-DSP2200 tiene unos puentes para seleccionar el tipo de acoplamiento entre AC o DC. La razón de esto está en que algunos dispositivos con los que la AT-DSP2200 puede conectarse requieren uno de los dos tipos de acoplo mencionados antes, al acoplo en continua podría por ejemplo en algunos equipos interferir en la polarización en continua.

Cada canal de salida tiene una resistencia de 51 Ω en serie con la salida. Esta resistencia protege a la tarjeta de cortocircuitos y ayuda a estabilizar el circuito de salida cuando se carga con cargas capacitivas o cables muy largos. La AT-DSP2200 ha sido diseñada para ser conectada a cargas con una impedancia de 10 KΩ o más. Sin embargo las especificaciones del circuito de salida han sido realizadas con cargas se 2 KΩ. Conectar cargas menores de 2 KΩ produce distorsión especialmente cerca del fondo de escala. Por ejemplo, conectar una carga de 2 KΩ provoca una caída de -0.21 dB con respecto al nivel sin carga. Las Salidas están protegidas contra cortocircuitos de forma que se puede conectar cualquier carga, si embargo el

resultado de la distorsión con algunas no será aceptable, especialmente con grandes señales. La AT-DSP2200 entrega aproximadamente ± 14 mA a la salida.

2.8 CIRCUITOS DE DISPARO.

LA AT-DSP2200 tiene tres circuitos de disparo que permiten comenzar una secuencia de adquisición y que son los siguientes:

- Disparo software.
- Disparo Digital o Analógico por los el conector EXTTRG* o sobre uno de los canales analógicos de entrada.
- Disparo recibido sobre el Bus RTSI recibido de otra placa AT.

2.8.1 Disparo Software

Esta es la forma más común de iniciar un adquisición. Un disparo software se da cuando programamos la adquisición para que comience cuando ocurra un evento de este tipo como por ejemplo la pulsación de una tecla o similar.

2.8.2 Disparo Digital Sobre El Conector Exttring*

La línea de disparo digital (EXTTRING*) es bidireccional, lo que significa que puede ser usada tanto como salida como entrada de disparo. La especificaciones de esta línea de disparo son:

Rangos de los valores máximos de entrada. + 5.75 con respecto a DGND.

- 0.5 V con respecto a DGND.

Especificaciones de la entrada digital (referidas a DGND).

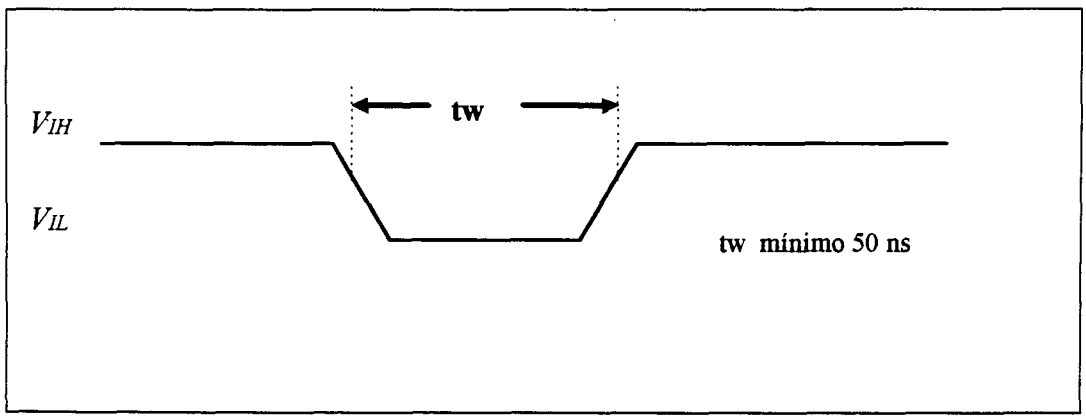
$_{IH}$ Tensión de entrada para nivel alto	2 V mínimo
$_{IL}$ Tensión de entrada para nivel bajo	0.8 V máximo
I_{IH} Carga de corriente de entrada, de tensión de entrada para nivel alto	10 μA máximo
I_{IL} Carga de corriente de entrada, tensión de entrada para nivel bajo	-10 μA máximo

Especificaciones de la salida digital (referidas a DGND)

$_{OH}$ Voltaje de salida para nivel alto	2.4 V mínimo.
$_{OL}$ Voltaje de salida para nivel bajo	0.5 V máximo.
I_{OH} Corriente de salida para nivel alto	-3.2 mA
I_{OL} Corriente de salida para nivel bajo	24 mA

La línea de disparo digital se puede programar para se disparada por flanco de subida o por flanco de bajada de la señal de entrada, esto se selecciona por software. El disparo digital una vez que realizado desencadena un suceso hasta el final, es decir hay que hacer un disparo por suceso. En la figura 15 vemos un ejemplo los requerimientos de tiempo cuando un disparo por flanco de bajada. En el capítulo cuatro, dedicado a la programación se explicará detalladamente como se programa un disparo hardware.

Figura 15. Requerimientos de tiempo para la señal EXTTRING*.



2.8.3 Disparo Analógico

La AT-DSP2200 también puede iniciar un ciclo de adquisición mediante un disparo analógico. Para ello se usan indistintamente las entradas ACH0 o ACH1. La sección del canal que hará de circuito de disparo se hace por software. Cuando programamos un disparo software podemos indicar el canal y el nivel mínimo de umbral para que se produzca este, indicadolo en voltios de -2.282 V a +2.282V. Esta facilidad permite que fijemos un nivel mínimo de tensión y si la señal que vamos a adquirir no lo sobrepasa no se almacena nada, evitando así llenar el dispositivo de almacenamiento con datos inútiles.

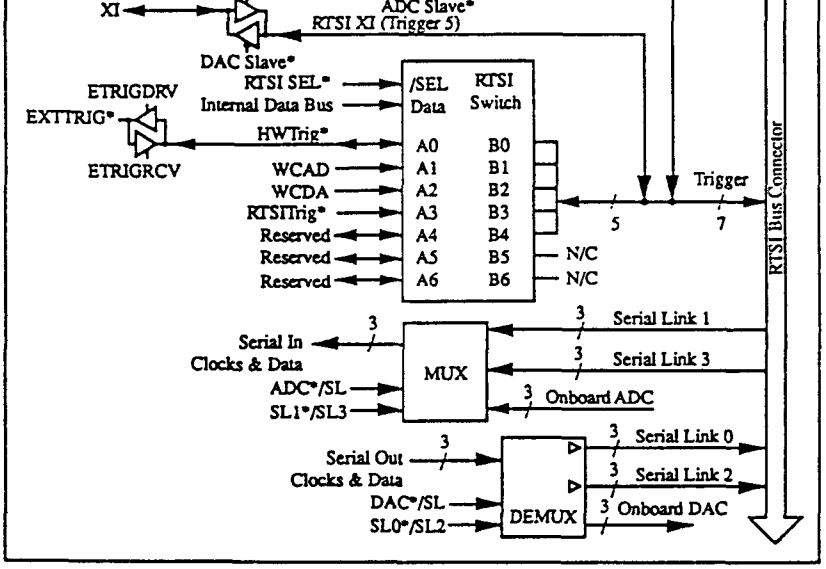
En el caso de seleccionar dos formas de disparo para desencadenar un suceso la primera de las dos que se produzca es la que se atiende como disparo. El capítulo de programación se explicara la forma de programar los disparos analógicos.

2.8.4 Disparo Sobre El Bus RTSI

La AT-DSP2200 también se puede disparar mediante el trigger del bus RTSI lo que permite la sincronización de varias AT-DSP2200 aumentando el numero de canales.

2.9 CIRCUITO DE INTERFACE DEL BUS RTSI

Figura 16. Diagrama de bloques del bus RTSI.



La AT-DSP2200 incluye un bus RTSI de National Instruments. Este bus tiene una línea de reloj, siete líneas de disparo y cuatro enlaces serie de datos. Mediante este bus se pueden interconectar toda la serie de placas de National Instruments que lo posean y compartir datos entre ellas. Este bus no da al exterior del PC ya que como dijimos esta pensado para conectar a las placas que se encuentren dentro del mismo PC. La programación del bus RTSI se explicará en el capítulo dedicado a la programación, en la figura 16 podemos ver un diagrama de bloques de este bus.

En la figura 16 podemos observar los principales bloques del RTSI, los drivers de reloj, el conmutador RTSI y los cuatro enlaces de datos serie. Estos elementos son los encargados de rutear a y desde la AT-DSP2200.

El conmutador RTSI es un circuito integrado hecho a medida por National Instruments que actúa como un conmutador de barras de 7x7. Las señales B<4..0> están conectadas a las cinco líneas de disparo del bus RTSI. La sexta línea, B5, es usada como un reloj maestro de forma que múltiples placas AT-DSP2200 puedan actualizar sus DACs a la vez. La séptima línea de disparo del bus RTSI, B6, se usa en la AT-DSP2200 en conjunto con otra señal para sincronizar los relojes de otras placas para conversión múltiple. Las señales A<3.0> están conectadas a cuatro señales en la placa. Las señales A4, A5 y A6 están reservada. El conmutador RTSI puede conectar cualquier señal A<0..6> a una o mas señales de B<6..0>. Con esta capacidad, hay gran flexibilidad para interconectar señales de cualquier serie de placas AT compartiendo las señales del bus RTSI.

En la placa AT-DSP2200, cuatro señales son conectadas a las señales A<3..0> del conmutador RTSI.. La señal HWTrig* es el disparo hardware usado para generar una interrupción externa a el chip DSP32C, el procesador digital, para disparar una operación de adquisición y puede ser originada desde el conector EXTTRING* o desde el bus RTSI La señal WCDA es de actualización del reloj del convertidor D/A. La señal RTSIrig* es generada por el registro de control de E/S analógicas controlado por el WE DSP32C y es usada para testear el funcionamiento des trigger hardware de la AT-DSP2200. La Seña RTSITrig* se usa además para enviar un disparo hardware común generado por por una sola operación sofware a múltiples placas AT-DSP2200 conectadas a traves del bus RTSI.

Además de todo esto el conmutador RTSI. de la AT-DSP2200 puede usar los enlaces serie de datos para recibir o transmitir datos entre la AT-DSP2200 y otras tarjetas de la serie AT que posean este bus.

2.10 MAPA DE REGISTROS

En este apartado se decribe el mapa de registros de la AT-DSP2200. En el caso de usar las herrameintas de programción a bajo nivel conviene conocer bien este mapa para saber como debemos comunicarnos con la tarjeta. Pero si como en mi caso no se usa o no se dispone de la herramienta de programción a bajo nivel no es necesario el conocimiento preciso del mapa de registro.

Los registros se clasifican en dos grupos que son:

- El mapa de registros del canal de E/S.
- El mapa de registros del DSP.

El primer grupo contiene los registros necesarios para comunicar el PC con el DSP mediante el puerto paralelo de DSP. En la tabla 7 se muestran estos registro y su dirección vista desde el PC. En el fichero cabecera dsp2200.h viene una definición de estos para poder ser incluidos en un programa.

Tabla 7. Mapa de registros del canal de I/O

Nombre del Registro	Dirección (hex)	Tipo	Tamaño
Grupo de registros del DSP			
PAR	Dirección Base+00	Lectura/Escritura	16 bits
PDR	Dirección Base+02	Lectura/Escritura	16 bits
EMR	Dirección Base+04	Lectura/Escritura	16 bits
ESR	Dirección Base+06	Lectura/Escritura	8 bits
PCR1	Dirección Base+07	Solo Lectura	8 bits
PIR	Dirección Base+08	Lectura/Escritura	16 bits
PCRh	Dirección Base+0A	Lectura/Escritura	8 bits
PARE	Dirección Base+0B	Lectura/Escritura	8 bits
PDR2	Dirección Base+0C	Lectura/Escritura	16 bits
Grupo de registros de Interrupción/DMA			
Registro de control de Interrupción/DMA	Dirección Base+10	Solo escritura	16 bits
Registro de estado	Dirección Base+10	Solo lectura	16 bits
Registro de Borrado de la interrupción Tc DMA	Dirección Base+12	Solo escritura	16 bits

Ahora comentaremos el significado de cada uno de estos registro que como dijimos pertenecen al puerto paralelo del DSP.

- Registro de Dirección de E/S Paralelo (PAR).

  Es un registro de 16 bits que es cargado con los 16 bits de menor peso de la dirección de la DMA del chip.

- Registro de Datos de E/S del Puerto Paralelo (PDR).

El PDR es un registro de 16 bits que puede ser leído o escrito por el DSP o por el PC y se usa para transferir datos entre ambos.

- Registro de Mascara de Error (EMR).

  El EMR es un registro de 16 bits que puede ser usado para enmascarar o desenmascarar las condiciones de parada del DSP producidas durante la ejecución del programa.

- Registro Fuente de Error (ESR)

  El registro ESR contiene el estado de alguna condiciones de error que pueden ocurrir en el chip DSP. El permiso para recibir esas condiciones es determinado por el registro EMR. El ESR solo puede ser leído por el PC y se borra después de cada lectura, También se borra cuando se produce un Reset en el DSP.

- Registro de Control Bajo de E/S del puerto paralelo (PCRl)

  El PCRl controla y monitoriza el estado del puerto paralelo del WE DSP32C.

- Registro de Interrupciones de E/S del Puerto Paralelo (PIR).

  El PIR es un registro de 16 bits que puede ser escrito o leído por el PC y por el chip DSP.

- Registro de Control Alto de E/S del Puerto Paralelo (PCRh).

  El PCRh controla y monitoriza el estado del puerto paralelo de chip DSP.

- Extensión del Registro de Direcciones de E/S del Puerto Paralelo (PARE)

  El PARE es un registro de 16 bits que se caga con los 8 bits MSB de la DMA del chip.

- Segundo Registro de Datos de E/S del Puerto Paralelo (PDR2)

El PDR2 es un registro de 16 bits que puede ser leído o escrito tanto por el PC como por el chip.

El grupo de registro de Interrupciones/DMA está formado por tres registro que se encargan de controlar y monitorizar el circuito de DMA e Interrupciones. El Registro de control de Interrupción/DMA controla las interrupciones las funciones DMA de la AT-DSP2200. El registro de estado nos indica el estado del circuito de interrupciones. El Registro de Borrado TC DMA se usa para borrar el contador de final de DMA si esto ocurre.

El segundo grupo de registros esta formada por el mapa de registros del DSP tal como es visto por WE DSP32C. Estos registros son de 32, 16 y 8 bits y están formados por el grupo de memoria, el grupo de Registros de E/S Analógicas, el grupo de Registros del Bus RTSI y un grupo llamado Miscelánea. en la tabla 8 podemos ver las características principales de dichos registros.

Tabla 8. Mapa de registros del DSP.

Nombre del Registro	Dirección (hex)	Tipo	Tamaño
Grupo de Memoria			
Memoria interna del Chip	000000 17FFFF	Lectura/Escritura	32 bits
Memoria Interna del Chip Banco 1	FFE000 FFE7FF	Lectura/Escritura	32 bits
Memoria Interna del Chip Banco 2	FFF000 FFFFFFFF	Lectura/Escritura	32 bits
Grupo de Registros de E/S Analógicos			
Registro de configuración de E/S Analógicas	300000	Solo Escritura	16 bits
Registro de Estado	300000	Solo Lectura	8 bits
Grupo de Registros del Bus RTSI			
Registro de Control de Enlace de Datos Serie	400000	Solo Escritura	16 bits
Registro de Desplazamiento del Conmutador RTSI.	600000	Solo Escritura	8 bits
Registro Strobe del Conmutador RTSI	600004	Solo Escritura	8 bits
Grupo de Registros Miscelánea			
Registro de Interrupciones AT	200000	Solo Escritura	16 bits
Registro de Diagnóstico Visual	500000	Solo Escritura	8 bits

- Grupo de Memoria.

  El grupo de memoria se usa por el chip como una memoria local por el DSP. La memoria es direccionada como bytes, palabras o como datos de tipos long y cada localización actúa coma un registro de datos de escritura/lectura de 32 bits. El mapa de memoria es el mismo para todas las versiones de AT-DSP2200, pero la cantidad de memoria en la placa es distinta para cada versión. En la tabla 9 se ve el mapa de memoria para el modelo de la AT-DSP2200 con 384Kword.

Tabla 9. Mapa de memoria para la AT-DSP2200 versión 384 Kword.

Rango de Memoria (dirección en Hex)	**Banco de Memoria**
000000 - 07FFFF	Banco 0
080000 - 0FFFFF	Banco 1
100000 - 17FFFF	Banco 2

- Grupo de Registros de E/S Analógicas.

  Los dos registros que pertenecen a este grupo controlan el circuito de entradas y salidas analógicas y se usan para monitorizar el estado del conversor ADC el DAC y la calibración de offset. El Registro de Configuración Analógico de E/S controla la frecuencia de muestreo , el acoplo de entrada (AC, DC, o GND), la calibración de offset del DAC, el modo de operación del DAC y un trigger que puede ser enviado mediante el Bus RTSI para sincronizar múltiples conversiones. El Registro de Estado indica el estado del ADC y el DAC, los ciclos de calibración y el muestreo y la actualización de los relojes.

- Grupo de Registros RTSI.

  Estos registros permiten programar al conmutador RTSI para rutear señales sobre las líneas del bus RTSI a y desde la AT-DSP2200, sincronizar muestreos, disparar varias placas y habilitar o inhabilitar los enlaces de datos serie.

- Grupo de registros Misceláneo.

  Los dos registros que forma este grupo de registros envían interrupciones al PC e indican el estado del los diodos de la placa. Estos diodos tiene un propósito de funcionar como depuradores del programa.

3. EL D.S.P.

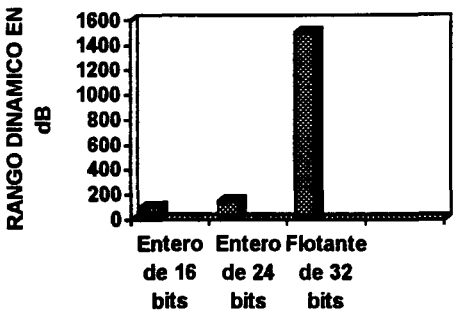
3.1 INTRODUCCIÓN A LOS D.S.Ps

Antes de nada tenemos que plantearnos una pregunta, ¿qué es un DSP ?, ¿ Cómo podemos reconocer a un Procesador Digital de Señal de otros microprocesadores?. En la actualidad la capacidad de llevar a cabo una operación de multiplicación y acumulación en un ciclo de máquina debería ser probablemente una característica que todos los DSP deberían cumplir. Normalmente el ciclo máquina de un DSP es excepcionalmente rápido entorno a los 80 ns e incluso los 35 ns. Pero no sólo esta característica de rapidez frente a los tradicionales procesadores Von Neumann es lo que caracteriza a los DSP. Las diferencias principales se basan en la arquitectura de proceso y en los métodos de programación. La arquitectura de los DSPs. facilita los procesos simultáneos a diversos niveles y en forma paralela.

Inicialmente debido a numerosos factores como el coste y complejidad de diseño las operaciones de los DSP estaba basada en una aritmética de coma fija de 16 bits y excepcionalmente 24 bits. Esto presentaba una limitación en cuanto a las variables en juego y la gama dinámica, es decir, al intervalo entre el valor máximo y el mínimo que se puede almacenar en memoria. Con 16 bits el rango dinámico es de 96 dB, lo que normalmente es insuficiente para la gestión precisa de muchos tipos de señales complejas. En la figura 1 se puede ver un gráfico donde se ve la gran

diferencia de rango dinámico entre un DSP de coma fija de 16 bits y 24 bits frente a uno de 32 bits en punto flotante.

Figura 1. Rango dinámico de un DSP de coma fija y coma flotante.



Fue AT &T la primera empresa que introdujo en el mercado el primer DSP de 32 bits en punto flotante, el DSP32C. Creado en 1985, fue diseñando, inicialmente, para el consumo interno. Los buenos resultados obtenidos en sus primeras aplicaciones, especialmente en centralitas telefónicas de conmutación digital impulsaron a AT &T a ponerlo en el mercado un tiempo después. A aquel primer DSP siguió el DSP32C, el que incorpora la AT-DSP2200, mas avanzado y rápido capaz de gestionar el direccionamiento de memoria a 24 bits. Al contrario de otros DSPs, en los de la familia DSP32 la arquitectura es poco convencional pues en ella se evita la separación entre los espacios de direccionamiento de los datos y los de las instrucciones. Arquitectura Von Neumann frente a la arquitectura Harvard. La estructura interna es muy sencilla, figura 2, con un sólo bus de datos y un sólo bus de direcciones. La escasa complejidad interna permite una adecuada gestión del tiempo compartido de los buses, de forma que se logra el acceso cuatro veces en un sólo ciclo de reloj. En el mismo ciclo se pueden realizar hasta dos transferencias de datos a y desde la memoria.

3.2 INTRODUCCIÓN AL CHIP WE DSP32C

Después de haber hablado un poco de lo que es un DSP vamos ahora a describir el DSP WE DSP32C, que es el que incorpora la AT-DSP2200 y del cual deriva su nombre.

El WE DSP32C es el último miembro de la familia de procesadores DSP de 32 bits en coma flotante de AT&T. Está diseñado para aplicaciones que requieran aritmética en punto flotante, grandes flujos de datos, grandes cantidades de memoria y bajo consumo. Estas características unido a su alta velocidad de trabajo representan los mayores avances en Procesadores Digitales de Señal en un solo circuito integrado. El DSP32C está fabricado con tecnología CMOS y encapsulado en formato PGA de 133 pines. Es totalmente compatible con el DSP32.

El DSP32C tiene dos unidades de ejecución, la unidad aritmética de control (CAU) y la unidad aritmética de datos (DAU), que se usan para llevar a cabo la ejecución de hasta 12.5 millones de instrucciones por segundo. Cada unidad tiene su propio conjunto de instrucciones. La CAU ejecuta operaciones lógicas y aritméticas con 16 y 24 bits en coma fija para lógica y control de funciones, mientras la DAU ejecuta operaciones en coma flotante de 32 bits y operaciones de conversiones de tipos de datos. La CAU puede funcionar de forma autónoma realizando transferencias de datos, control de saltos, aritmética de coma fija y operaciones lógicas en paralelo con operaciones en coma flotante con la DAU. Además la CAU genera direcciones para los operandos de la DAU. Esta forma de trabajo se puede comparar la de un

microprocesador convencional considerando la DAU como el procesador y la CAU como un coprocesador.

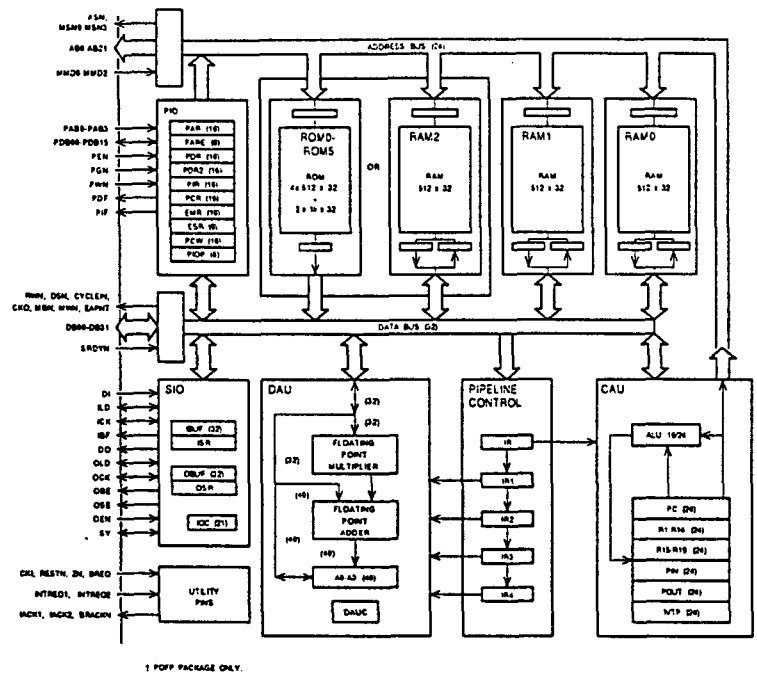
Excepto para cargar un registro, la ejecución de una instrucción de la CAU se realiza antes de que la siguiente instrucción comience. Esto simplifica el uso de la CAU para operaciones lógicas y de control. La DAU emplea cuatros estados "pipeline" para ejecutar 25 millones de cálculos por segundo en coma flotante. Esto significa que hay un pequeño número de periodos de latencia asociados con el conjunto de instrucciones de la DAU.

La DAU está configurada para realizar operaciones de multiplicación y suma y es la unidad de ejecución primaria para los algoritmos de procesado de señales. Contiene un multiplicador y un sumador en coma flotante que trabajan en paralelo para realizar operaciones de la forma a = b+c*d. Para comprender el funcionamiento del DSP32C, uno debe comprender el encauzamiento ,"pipeline", de operaciones de la DAU. La DAU emplea un sencillo ciclo "pipeline" de búsqueda-multiplicación-suma-escritura que se explicará más adelante en detalle. La DAU ejecuta una instrucción de multiplicación y una suma es cuatro pasos:

- Búsqueda de c y d.
- Multiplicación de c y d.
- Suma de b y el producto de c y d. El resultado se guarda en el acumulador.La última operación es opcional, escribir el resultado en memoria o en un puerto de E/S.

Un máximo de dos de los tres operandos multiplicación suma pueden venir de memoria.

Figura 2. Diagrama de bloques del DSP32C



LEGEND:					
A0—A3	Accumulators 0—3	ISR	Input shift register	PDR2	PIO data register 2
ALU	Arithmetic logic unit	IVTP	Interrupt vector table pointer	PIN	Serial DMA input pointer
CAU	Control arithmetic unit				
DAU	Data arithmetic unit	OBUF	Output buffer	PIO	Parallel I/O unit
DAUC	DAU control register	OSR	Output shift register	PIOP	PIO port register
EMR	Error mask register	PAR	PIO address register	PIR	PIO interrupt register
ESR	Error source register	PARE	PIO address register extended	POUT	Serial DMA output pointer
IBUF	Input buffer				
IOC	Input/output control register	PC	Program counter	R1—R19	Registers 1—19
		PCR	PIO control register	RAM	Read/write memory
IR	Instruction register	PCW	Processor control word	ROM	Read-only memory
IR1—IR4	Instruction register pipeline	PDR	PIO data register	SIO	Serial I/O unit

Los datos son transportados a través del dispositivo mediante el bus de datos de 32 bits. El bus de datos soporta cuatro accesos a memoria durante cada ciclo de instrucción: una instrucción de búsqueda, lectura de dos operandos y una escritura en memoria. Esta alta velocidad del bus de datos y la arquitectura "pipeline" del

dispositivo permiten al DSP32C buscar dos operandos de 32 bits de la memoria o del puerto de E/S, realizar una multiplicación y una suma y escribir el resultado a la memoria o a un puerto de E/S durante cada ciclo de reloj.

3.3 CICLO DE INSTRUCCIÓN.

El DSP32C tiene un ciclo de instrucción de 80 ns a una frecuencia de 50 Mhz. Cada ciclo de instrucción se divide en cuatro estado de máquina, numerados de 0-3, donde un estado es igual a un periodo de reloj, o 20 ns a una frecuencia de 50 Mhz. En una sólo ciclo de instrucción, el DSP32C puede ejecutar un ciclo de búsqueda, búsqueda de dos operandos y escritura en memoria. Cada instrucción de multiplicación suma de la DAU puede tener hasta cuatro accesos a memoria: (1) lectura en memoria (instrucción Y), (2) lectura en memoria (operando X), (3) lectura en memoria (operando Y) y escritura en memoria (operando Z). Debido a que la DAU tiene arquitectura "pipeline", esos cuatros accesos no ocurren todos en el mismo ciclo de instrucción para una instrucción dada. Figura 3a.

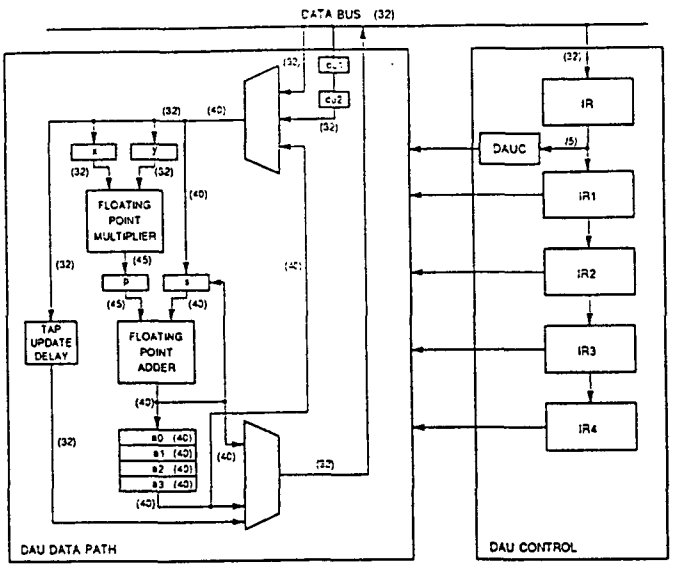
La figura 3b muestra un encauzamiento total "full pipeline" de un secuenciamiento de instrucciones multiplicación suma de la DAU. Los números indican la instrucción; las instrucciones se buscan y se ejecutan en orden ascendente. Por ejemplo, X_0 es el operando X para la instrucción número 0, y X_{-1} es el operando X para la instrucción previa al número 0.

3.4. UNIDAD ARITMÉTICA DE DATOS (DAU)

La DAU es la unidad de ejecución primaria para la ejecución de algoritmos de procesado de señal. Esta unidad contiene un multiplicador en coma flotante de 32 bits, un sumador de 40 bits en coma flotante, cuatro acumuladores de 40 bits y un registro de control de unidad de aritmética de datos (DAUC). El multiplicador y el sumador trabaja en paralelo para ejecutar 12.5 millones de cálculos por segundo de la forma (a = b + c * d). La figura 4 representa un diagrama de bloques de la DAU. La DAU transmite datos y recibe datos de otras secciones del chip a través del bus de datos interno. La DAU esta dividida en dos partes, ruta de datos y unidad de control. La ruta de datos consiste de un multiplicador en coma flotante, un sumador en coma flotante, registros, buses y conectores de bus. El multiplicador y el sumador de la DAU actúan en paralelo, produciendo cada una un resultado por ciclo de instrucción. Los cuatro registros que hacen de acumulador pueden ser leídos o escritos por el programa de control. La unidad de control de la DAU decodifica cada instrucción en señales que controlan la ruta de datos. La ruta de datos y la unidad de control de la DAU contiene cuatro etapas de encauzamiento. Así, en cualquier ciclo de instrucción, la DAU puede estar procesando cuatro instrucciones diferentes, cada una en diferentes estados de ejecución.

La DAU contiene registros de 32 bits , registros de 40 bits y buses para soportar los dos formatos de coma flotante. El formato de 40 bits proporciona ocho bits adicionales como mantisa de guarda. El figura 5 se muestran ambos formatos.

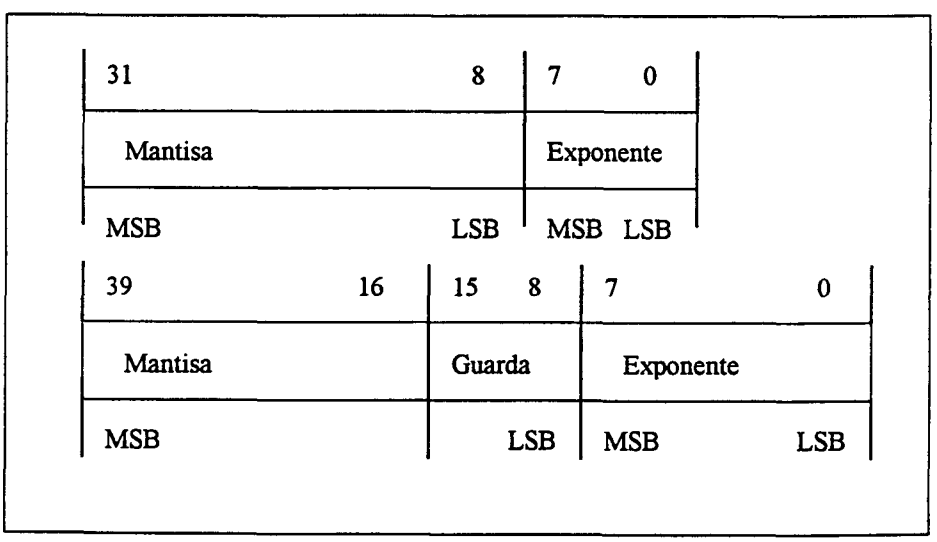
Figura 6. Esquema de la DAU.



LEGEND:

a0—a3	Accumulators 0—3	IR3	DAU instruction register (instruction cycle 3)
DAUC	DAU control register	IR4	DAU instruction register (instruction cycle 4)
du1, du2	Adder input operand delay registers	p	Product register
IR	Instruction register	s	Special function register
IR1	DAU instruction register (instruction cycle 1)	x	Multiplier input register
IR2	DAU instruction register (instruction cycle 2)	y	Multiplier input register

Figura 6. Formatos de Coma flotante.



Cuando un bus de 40 bits se conecta a un registro de 32 bits, los bits de guarda son excluidos (truncados). Por ejemplo, cuando escribimos el resultado de un

acumulador de 40 bits a memoria, el resultado es primero truncado a 32 bits. Cuando un registro de 32 bits se pone en un bus de 40 bits, los bits de guarda (bits del 15 al 8) se ponen a cero.

Cada instrucción de multiplicación de acumulación de la DAU implica tres operandos en coma flotante. Dos de esos operandos son multiplicados a la vez, y el resultado es sumado al tercero. El resultado de la suma es almacenado en el acumulador. El valor en este acumulador puede ser un resultado intermedio usado luego en otra operación de multiplicación acumulación, o puede ser el resultado final para ser almacenado en memoria o puesto en la unidad de E/S.

Las entradas del multiplicador de la DAU pueden originarse en la memoria, la entrada de puerto SIO o un acumulador. Además pueden seleccionase valores constantes como entradas al multiplicador. Las entradas del multiplicador son números de 32 bits de como flotante con una mantisa de 24 bits y ocho bits de exponente. Si la entrada al multiplicador proviene de un acumulador, los bits de guarda del acumulador son primero truncados. El multiplicador siempre proporciona una entrada al acumulador. Los valores constantes también pueden ser seleccionados como entradas del sumador. La otra entrada del sumador se puede originar en memoria o en el puerto de entrada SIO.

Los cuatro acumuladores de 40 bits a0 - a3, reducen los problemas de redondeo al asegurar 24 bits de precisión. La lógica de Post-normalización transparentemente desplaza el punto binario y ajusta el exponente para prevenir la

imprecisión del redondeo de bits cuando se suma o multiplica números en punto flotante, eliminando el error de cuantización y escalado. Todas las normalizaciones son hechas automáticamente, de esta forma los resultados del acumulador están completamente normalizado.

Las instrucciones simples como la conversión de tipos de datos se hacen en la DAU mediante hardware, de esta forma se reduce el tiempo de conversión. Los operandos paras esas conversiones pueden provenir de la memoria, del registro de datos PIO, del puerto de entrada SIO o de un acumulador. El registro DAUC se usa para configurar algunos de esos tipos de conversiones. Este registro normalmente puesto, escrito, al comenzar un programa de aplicación y rara vez es modificado durante la ejecución del programa. La DAU ejecuta conversiones de tipos de datos entre el formato de coma flotante de 32 bits del DSP32C y el formato estándar de 32 bits de coma flotante IEEE 745 y entre los formatos lineales de 8 bits y los enteros de 16 bits, 24 bits, 8bits μ-law y A-law. La DAU además proporciona una instrucción para convertir operandos de coma flotante de 32 bits a valores "semilla" usados para aproximaciones reciprocas en operaciones de división.

A continuación presentamos una descripción de como pasa una instrucción simple a través de la "encauzamiento" ,pipeline, de la DAU. Como ya dije la DAU soporta cuatro formatos de instrucciones de multiplicación-acumulación, pero para simplificar estos ejemplos solo veremos un formato.

El DSP32C ejecuta una instrucción de multiplicación acumulación del tipo, aN = aM + Y * X , en tres pasos, hay un cuarto paso opcional que es la escritura en memoria. Los tres pasos básicos son:

1. Búsqueda de X e Y.
2. Multiplicación de Y * X.
3. Suma de producto con aM

Si se ejecutan varias instrucciones de multiplicación acumulación una después de otra el DSP32C encauza automáticamente las instrucciones de forma que de forma que una instrucción es completada en cada ciclo de instrucción. En los siguientes párrafos veremos como se realiza esto.

1. aN = aM + Y1 * X1
2. aN = a M + Y2 * X2
3. aN = aM + Y3 * X3
4. aN = aM + Y4 * X4

A continuación se muestra como se ejecuta cada instrucción.

1.			*Búsqueda de X Y*
2.		*Multiplicar1*	*Búsqueda 2 de X Y*
3.	*Acumulación1*	*Multiplicar2*	*Búsqueda3 de X Y*
4.	*Acumulación2*	*Multiplicar3*	*Búsqueda4de X Y*
5.	*Acumulación3*	*Multiplicar4*	
6.	*Acumulación4*		

El programador debe darse cuenta de cuando termina una acumulación si se piensa usar el resultado de la acumulación como entrada del multiplicador. Por ejemplo, la acumulación de la instrucción 1 deberá estar disponible como entrada del multiplicador para la instrucción 4. Un resultado acumulado puede ser siempre usado como entrada al acumulador en la siguiente instrucción.

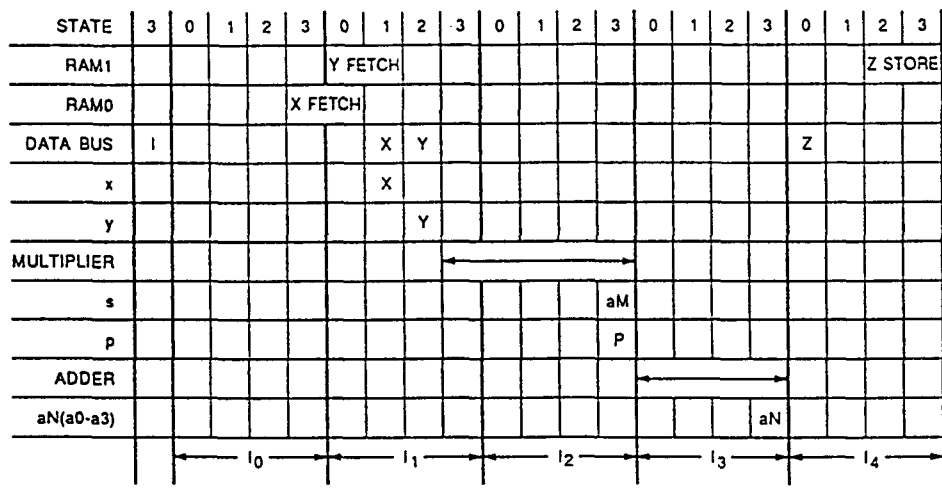
El encauzamiento ,"pipeline", del DSP32C puede ser extendido a cuatro estados, permitiendo al contenido de un acumulador la entrada del multiplicador, o a la entrada del sumador escribir en memoria o a un puerto de E/S sin necesidad de instrucciones adicionales. Esto es una ventaja cuando ejecutamos tareas como filtros adaptativos, operaciones con matrices, etc. Para ilustrar este paso adicional vemos el siguiente ejemplo, en el que vemos que se añade otra columna al diagrama de encauzamiento anterior.

1.			*Búsqueda1 de X Y*
2.		*Multiplicar1*	*Búsqueda2 de X Y*
3.	*Acumulación1*	*Multiplicar2*	*Búsqueda3 de X Y*
4. Escritura1	*Acumulación2*	*Multiplicar3*	*Búsqueda4 de X Y*
5. Escritura2	*Acumulación3*	*Multiplicar4*	*Búsqueda5 de X Y*
6. Escritura3	*Acumulación4*	*Multiplicar5*	
7. Escritura4	*Acumulación5*		
8. Escritura5			

La sintaxis de cada instrucción es Z = aN = aM + Y * X. Puede verse que la posición de memoria escrita por la instrucción 1 es la entrada del multiplicador en la instrucción 5.

La temporización interna del encauzamiento de la DAU para una instrucción simple de la forma anterior se muestra en la figura 7.

Figura 7. Temporización interna del encauzamiento para una instrucción simple



For instructions in the form of: Z = aN = aM + Y * X,

where X, Y = data reference (memory, I/O registers, a0—a3)
aM = a0—a3
aN = a0—a3
Z = data write (memory, I/O registers)
P = product
S = adder input

Una instrucción requiere cinco ciclos de instrucción (I_0 - I_1) para ser decodificada y ejecutada en la DAU. La instrucción aparece en el bus de datos en el estado 3 precediendo a I_0. Los registros X e Y son cargados con los estados 1 y 2 de

I_1, respectivamente. El contenido de esos registros son multiplicados durante I_2 y el producto es cargado dentro del registro de producto, p, en el estado 3 de I_2. El registro de entrada del sumador, s, es cargado también con el estado 3 de I_2. El contenido de los registros p y s es sumado duranteI_3 , y el resultado es cargado dentro del acumulador en el estado 3 deI_3. Los contenidos de los registros p y s son sumados durante I_3, y el resultado es cargado al acumulador en el estado 3 de I_3. Este mismo resultado es situado en el bus de datos (para ser llevado a memoria o a aluno de los puestos) en el estado 0 de I_3 si se especifica en el capo Z de la instrucción que es opcional.

La DAU soporta dos formatos de instrucciones de multiplicación acumulación llamados instrucciones *tap-update.*

$$aN = aM + (Z = Y) * X$$

En el primer formato el resultado de la operación de multiplicación acumulación en lugar de ser escrito en memoria o en los puertos, es escrito el operando Y (una entrada del multiplicador). Este debe ser escrito en el mismo ciclo que el resultado del multiplicador acumulador (estado 0 de I_4). Debido a que el operando Y aparece en el bus de datos en el estado 2 de I_1, debe se "latcheada" y mantenida durante tres ciclos de instrucción antes de comenzar la escritura a memoria o a un registro. Este retardo de tres ciclos de instrucción es realizado en el bloque de retardo *tap-update.* El otro formato de instrucción de *tap-update* tiene la siguiente sintaxis.

$$aN = (Z = Y) + X$$

Aquí, en lugar de escribir en memoria o en los puertos el resultado de una operación de multiplicar acumular, se escribe el operando Y. Debido a que el operando Y está disponible en la DAU en el estado 3 de I_2, debe ser latcheado y mantenido en la el bloque *tap-update* de la DAU hasta que se escriba en el estado 0 de I_4.

3.5 UNIDAD ARITMÉTICA DE CONTROL (CAU)

La CAU es la responsable de llevar a cabo el calculo de las direcciones, el control de salto y las operaciones lógicas y aritméticas en 16 y 24 bits de coma fija. Contiene una unidad lógica aritmética de 24 bits (ALU) que ejecuta las operaciones lógicas y aritméticas, un registro contador de programa (PC) de 24 bits y 22 registros de 24 bits de propósito general. En la figura 7 vemos un diagrama de bloques de la CAU.

La CAU tiene dos modos de operación: en uno ejecuta instrucciones de control aritmético (CA) y en el otro genera direcciones para los operandos de las instrucciones de datos aritméticos (DA). Las instrucciones CA llevan a cabo movimientos de datos, control de saltos y operaciones lógicas de 16 y 24 bits. Las instrucciones DA pueden realizar hasta cuatro accesos a memoria por instrucción, y la CAU es la responsable de generar esas direcciones usando el modo de

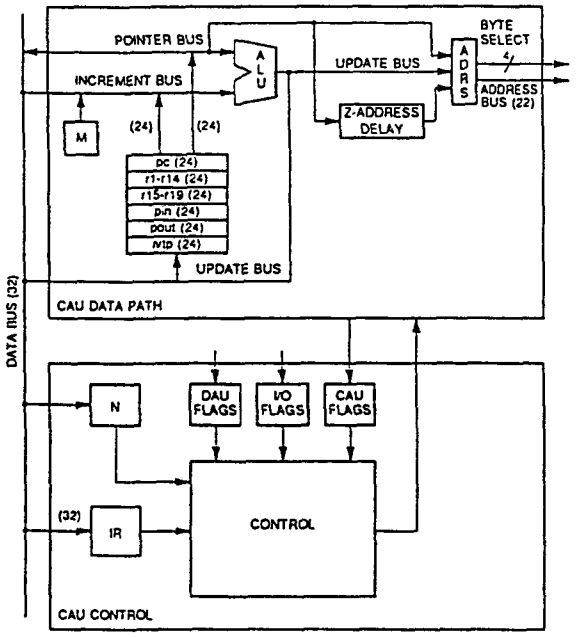
direccionamiento de registros indirectos, modo postmodificado, una dirección en cada uno de los cuatro estados de un ciclo de instrucción.

Los registros r1 a r14 se usan como registros de propósito general en las instrucciones (CA). Como punteros a memoria en las instrucciones (DA). Cuando se usan como punteros a memoria, r1 a r14 mantienen direcciones de 24 bits. Los registros r15 a r19 se usan como registros de propósito general en instrucciones CA y como registros de incremento en instrucciones en DA. Cuando se usan como registros de incremento, r15 a r19 tienen valores de 24 bits que pueden posmodificar las direcciones en los punteros a memoria. El registro r20, llamado PIN (puntero in), es usado como puntero de entrada para la DMA del puerto serie de E/S SIO, y r21, llamado POUT (puntero out), es usado como puntero de salida para la DMA del puerto serie de E/S SIO. Cuando r20 y r21 se usan como registros de propósito general, hay que tener en cuenta sus efectos sobre la DMA si se habilita la DMA del puerto serie. El registro r22, llamado IVTP (tabla de punteros de vectores de interrupción), mantiene la dirección base de la tabla de vectores de interrupción. Cuando r22 se usa como registro de propósito general, se deben tenerse en cuenta los efectos sobre las interrupciones.

Todas la instrucciones aritméticas de CA se realizan con 24 bits en complemento a dos y con operandos enteros. Si un flag, "bandera", se ve afectado por una instrucción, se calcula basándose en las reglas de operación de los flag en 16 o 24 bits, dependiendo del tamaño de la operación que se especifica en la instrucción. Por ejemplo, cuando se realizan operaciones usando aritméticas enteras de 16 bits de datos, se cargan datos de 16 bits en los 16 bits bajos de los registros de 24 bits de la

CAU, con el bit mas significativo del entero extendido a los ocho bits superiores; cuando se ejecutan operaciones con 24 bits, con flags calculados de acuerdo a las reglas de operación para flags de 16 bits; y el resultado puede ser escrito en los 16 bits bajos del registro o en una posición de memoria de 16 bits.

Figura 8. Diagrama de bloques de la CAU.



LEGEND:

IR	Instruction register	ivtp (r22)	CAU general-purpose register, Interrupt vector table pointer
pc	Program counter	M	Do-loop counter register (number of instructions in loop)
r1—r14	CAU general-purpose registers, DAU pointer registers	N	Do-loop counter register, immediate value (number of iterations of loop)
r15—r19	CAU general-purpose registers, DAU increment registers		
pin (r20)	CAU general-purpose register, serial DMA input register		
pout (r21)	CAU general-purpose register, serial DMA output register		

Algunas de las instrucciones CA pueden ser realizadas usando tres operandos distintos: dos registros fuente (rS_1, rS_2) y un registro destino (rD). Las instrucciones CA pueden además ser ejecutada condicionalmente, basándose en los flags generados

en la CAU. La figura 9 muestra el encauzamiento de la CAU para la ejecución de dos instrucciones de la forma:

rD - rS

if (eq) rD = rS1 + rS2

La primera instrucción resta el contenido del registro fuente rS del registro fuente rD para poner o borrar el flag cero (z) de la CAU. La segunda instrucción chequea el contenido de flag z y si el flag es uno (si el resultado de la operación previa fue cero), suma el contenido de rS1 y rS2 almacenando el resultado en rD. Si el flag no es puesto a uno, el contenido de rD se ve afectado por esa instrucción.

Figura 9. Temporización interna del "pipeline de la CAU. Para una instruccón CA

STATE	3	0	1	2	3	0	1	2	3	0	
POINTER BUS	PC_1		rD		PC_2		rS_1		PC		
INCREMENT BUS	+4		rS		+4		rS_2		+4		
ALU		+		—		+		+		+	
UPDATE BUS			PC_2				PC_3		?rD		
ADDRESS BUS		PC_1				PC_2				PC_3	
DATA BUS	I_0				I_1				I_2		
IR		I_0				I_1				I_2	
FLAGS					Z				?		
		← I_0 →				← I_1 →					

For instructions in the form of:

I_0: rD – rS
I_1: if(eq) rD = rS_1 + rS_2

where rD = destination register
rS, rS_1, rS_2 = source register
eq = CAU condition *equal to zero*, based on the CAU z (zero) flag
? = modification of registers and flags is conditional
PC = program counter
+ = addition operation
– = subtraction operation

Excepto para el registro cargado desde memoria que tiene una latencia de un ciclo de instrucción, la ejecución de una instrucción en la CAU se completa antes de que la siguiente instrucción comience a ejecutarse. Esto simplifica el uso de la CAU para operaciones de lógica y control.

Durante la ejecución de instrucciones DA, la CAU genera hasta cuatro direccionamientos, una dirección en cada uno de los cuatro estados de un ciclo de instrucción. En cada estado, la CAU puede sumar el contenido de dos registros: un puntero seleccionado de r1 a r14 y un incremento seleccionado de r15 a r19. La CAU emplea tres estados "pipeline":

1. Búsqueda de un registro.
2. Operación sobre los operandos buscados.
3. Almacenar el resultado en un registro.

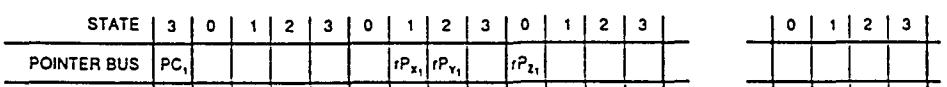
Debido al encauzamiento el puntero anterior en actualizado, mientras el puntero siguiente está siendo accedido. La figura 10 muestra el encauzamiento para la ejecución de una instrucción DA de la forma:

$$Z = aN = aM + Y * X$$

Como el resultado de la operación de multiplicación acumulación no está disponible en el bus de datos hasta el estado 0 del ciclo de instrucción I_5 y la dirección destino se calcula en el estado 1 del ciclo de instrucción I_2, la dirección es latcheada y

mantenida durante tres ciclos de instrucción antes de ser situada en el bus de direcciones. Este retardo de tres ciclos de instrucciones es realizado en el bloque llamado *Z-address delay* de la figura 7.

Figura 10. Temporización interna del "pipeline de la CAU. Para una Instrucción DA



For instructions in the form of:

$$I_1: \quad {}^*rP_Z{+}{+}rI_Z = aN = aM + {}^*rP_Y{-}{-}rI_Y * {}^*rP_X{+}{+}rI_X$$

where rP_Z, rP_Y, rP_X = pointer register for DAU, Z, Y, and X operands

rI_Z, rI_Y, rI_X = increment register for DAU, Z, Y, and X operands

PC = program counter

+ = addition operation

A = address

3.6 LA MEMORIA

El DSP32C tiene memoria interna y un interface para memoria externa. Las instrucciones y los datos pueden residir tanto en la memoria externa como en la interna, independientemente de si esta es RAM o ROM, El espacio de localización de memoria puede ser configurado en ocho modos diferentes. Independientemente de la

configuración, la primera instrucción después del reset debe estar situada en la dirección 0x000000 hex.

Internamente el DSP32C tiene 1024 palabras de 32 bits de RAM que está disponible en todas las configuraciones de memoria. Además se proporciona en el chip 4096 palabras de ROM programable mediante máscaras o 512 palabras adicionales de RAM. Así son posible dos opciones de memoria:

1. 1024 palabras de RAM y 4096 palabras de ROM.
2. 1536 palabras de RAM.

La RAM del chip es estática y no necesita refresco. La ROM puede ser grabada con un programa o con datos. Los contenidos de la ROM pueden ser codificados.

El interface de memoria externa puede direccionar directamente hasta 16 Mbytes, 8 Mbytes de palabras de 16 bits o 4M de memoria adicional sin pérdida de la velocidad de ejecución. Este interface puede además soportar cuatro estados de espera para permitir el acceso a memorias lentas, a periféricos de E/S y para permitir al arbitro de bus compartir las señales del interface de memoria externa con otros dispositivos. La memoria externa se divide en dos secciones: una zona baja (A) y una zona alta (B). El número de estados de espera para cada banco de memoria se puede configurar de forma independiente. De esta forma, se pueden mezclar memorias lentas y rápidas.

3.7 EL BUS DE DATOS

El DSP32C emplea una arquitectura Von Neumann con un sólo bus de direcciones , un sólo bus de datos y un solo espacio de direccionamiento. La arquitectura soporta múltiples búsquedas de programa y datos en un solo ciclo de instrucción usando una estructura paralela entrelazada. Las direcciones y los datos son multiplexadas en tiempo a cuatro veces la velocidad de instrucción dentro del bus de datos y direcciones respectivamente.

Los datos viajan por el DSP32C mediante el bus de datos interno de 32 bits. El bus de datos interno llega a todas las secciones del chip, y también al interface de memoria externa, que conecta con el bus de datos externo de 32 bits.

3.8 TIPOS DE DATOS

Internamente, el DSP32C tres tipos de formatos de datos: 16 bits y 24 bits en coma fija y 32 bits en coma flotante. La CAU usa los formatos enteros de 16 y 24 bits para datos en coma fija. La CAU además usa 24 bits para las direcciones. La DAU usa el formato de coma flotante para las operaciones de multiplicación acumulación.

3.8.1 Coma Flotante

La DAU usa dos formatos de coma flotante: el formato de 32 bits sin guarda y el formato de 40 bits con guarda. Este último formato se usa sólo en el interior de la DAU.

El formato de punto flotante de 32 bits proporciona un rango dinámico de mas de 1500 dB.

3.8.2 Enteros de 16 Bits

La CAU soporta tipos de datos enteros de 16 bits en complemento a dos que son usados para operaciones en punto fijo. El rango de va de -2^15 a 2^15-1. Los registros de la CAU tienen el tamaño de 24 bits. Cuando se llevan a cabo operaciones con 16 bits solo se usan los 16 bits de orden inferior de los registros.

3.8.3 Enteros de 24 Bits

La CAU también admite enteros de 24 bits en complemento a dos, usados para operaciones en punto fijo. El rango de direccionamiento de 24 bits es de -23^23 a 2^23-1. Si una instrucción de la CAU se ejecuta con 24 bits, la instrucción debe incluir el sufijo "e" en el nombre del registro. Si se escribe en memoria un valor de 24 bits, su signo se extiende a 32 bits y se carga dentro de una posición de memoria de 32 bits. Los datos enteros de 24 bits pueden ser además usados para direccionar memoria.

3.9 MODOS DE DIRECCIONAMIENTO

El DSP32C soporta varios modos de direccionamiento, inmediato, directo a memoria, directo a registro, indirecto a registro con posmodificación e indirecto a

registro con postmodificación de acarreo inverso. El resultado es un potente conjunto de formas de acceder a los datos. En las siguientes secciones se explicará cada uno de estos modos.

3.9.1 Direccionamiento Inmediato

La notación para definir este direccionamiento es N. En este modo el operando N es suministrado por la instrucción. Este modo no puede se usado por los operandos de la DAU. N puede ser un cualquiera de los dos tipos de datos enteros 16 o 24 bits.

3.9.2 Direccionamiento Directo a Memoria

La notación para definir este direccionamiento es *N. En este modo la instrucción contiene la dirección de un operando en memoria. Un direccionamiento directo no puede ser aplicado a los operandos de la DAU.

3.9.3 Direccionamiento Directo a Registro.

La notación es REG. En este modo se especifica en la instrucción un registro que contiene el operando.

3.9.4 Direccionamiento Indirecto a Registro

La notación es *rP, *rP++, *rP--, rP++rI. En este modo la instrucción contiene el nombre de un registro puntero (rP) que contiene la dirección del operando. Esos registros pueden ser posmodifcados. Este modo de direccionamiento se suele usar en algoritmos de procesado de señales en el que los operandos son organizados en tablas. El asterisco precediendo al nombre del puntero significa "dato apuntado por la dirección del registro".

- ***rP :** Esto significa, "el dato apuntado por la dirección del registro". El contenido del registro puntero rP no es alterado por la operación.
- ***rP++:** Los signos "++" indican un posincremento del registro puntero. El registro es posincrementado por 1, 2 o 4 dependiendo del tipo de dato.
- ***rP--:** Los signos "--" indican un posdecremento de registro puntero. El registro es posdecrementado por 1, 2 o 4 dependiendo del tipo de dato.
- ***rP++rI:** En este caso "++rI" indica un posincremento del registro puntero por los contenidos de un registro de incremento "el dato apuntado por la dirección en el registro rP; suma los contenidos del registro incremento rI para evaluar el registro puntero rP después de que la operación se ha completado.

3.9.5 Direccionamiento de Bit Inverso.

La notación es *rP++rIr. Este es un caso especial de direccionamiento de registro indirecto, donde la modificación del puntero registro es ejecutada por la propagación del acarreo en la dirección inversa, desde el bit mas significativo al

menos significativo. Este modo acelera las aplicaciones que requieran gran cantidad de datos como el algoritmo de la FFT.

3.9.6 Direccionamiento Relativo a PC (Contador de Programa)

La notación de este direccionamiento es pcgoto etiqueta. Esta instrucción es generada por el ensamblador y es un goto N(rH), donde rH es el PC, y N se calcula por el ensamblador para ser la distancia entre el valor PC y la etiqueta.

3.10 CARACTERÍSTICAS DE CONTROL DEL PROCESADOR.

La unidad de control del DSP32C proporciona la coordinación de las operaciones que ocurren entre los principales subsistemas del dispositivo. En este apartado se discutirán varios modos de control del procesador.

3.10.1 Unidad De Control

La unidad de control es la encargada de reconocer y priorizar las características de control del dispositivo. En la tabla 1 se ve un esquema de las prioridades cuando varios procesos requieran al procesador. Las características se listan a la derecha y en la parte superior de la tabla y la acción resultante cuando son requeridas durante el mismo ciclo de instrucción dos peticiones se explica mediante la intersección de una fila con una columna.

Tabla 1. Característica de prioridades del Procesador

	SIO DMA	PIO DMA	Interrupción	Parada	Reset
PIO DMA	Tiene prioridad la SIO DMA a menos que una PIO DMA este en marcha	-----	------	-----	----
Interrupción	Si hay una PIO DMA en marcha, la SIO DMA tiene prioridad. Si no la interrupción tiene preferencia al SIO DMA en un ciclo de instrucción	Si una SIO DMA se esta realizando, la PIO DMA tiene prioridad, sino la interrupción tiene preferencia a la PIO DMA en un ciclo de instrucción	------	-----	-----
Parada	La parada tiene prioridad. El SIO DMA es inhabilitado.	La Parada tiene prioridad, pero el PIO DMA se mantiene activa durante la parada.	La Parada tiene prioridad, la interrupción no es servida.	-----	-----
Reset	El Reset tiene prioridad	El Reset tiene prioridad	El Reset tiene prioridad	El Reset tiene prioridad	----
Ceder el Bus EMI*	El Bus es cedido; SIO DMA activa, El DSP32C debe esperar por el bus EMI	EL Bus es cedido; PIO DMA activa, el DSP32C debe esperar por el Bus EMI	El Bus es cedido; interrupción activa, el DSP32C debe esperar por el Bus EMI	El Bus es cedido y el chip se para.	El Reset tiene prioridad

* EMI= Interface de Memoria Externa.

3.10.2 La DMA

Los dispositivos externos pueden acceder a la RAM interna del DSP32C, así como la memoria externa, usando el Acceso Directo a Memoria (DMA). Las transferencias DMA pueden ocurrir entre la memoria y los puertos de E/S sin

intervención del procesador, por ejemplo, por robo de ciclo donde el procesador para durante una transferencia DMA. El chip soporta tres controladores de acceso a memoria; mediante la entrada serie, la salida serie, y por los puertos de E/S paralelos.

En el modo DMA, las transferencias de E/S serie (SIO) ocurren entre el registro del buffer de entrada (IBUF) y memoria y/o entre memoria y el registro del buffer de salida (OBUF) sin intervención del programa. En este caso los registros PIN (punteros de entrada) y POUT (punteros de salida) de la CAU actúan como punteros dedicados para las transferencias DMA.

Las transferencias DMA por las E/S paralelas son siempre iniciadas por un dispositivo externo. Usando DMA, un dispositivo externo puede descargar un programa a el DSP32C sin interrumpir la ejecución del programa. El dispositivo externo pone la dirección de memoria de comienzo para la transferencia DMA en el Registro de Direcciones Paralelo (PAR). Las transferencias DMA ocurren entre la memoria y el registro PDR (transferencia DMA de 16 bits) y entre la memoria y los registros PDR/PDR2 (transferencias DMA de 32 bits). Mientras el modo PIO DMA esta habilitado, el usuario debería tener la precaución de no usar los registros PDR/PDR2.

Si se solicita una DMA SIO y una DMA PIO durante el mismo ciclo del procesador, la DMA SIO requerida tiene una prioridad mas alta y debe ser servida primero.

3.10.3 Las Interrupciones

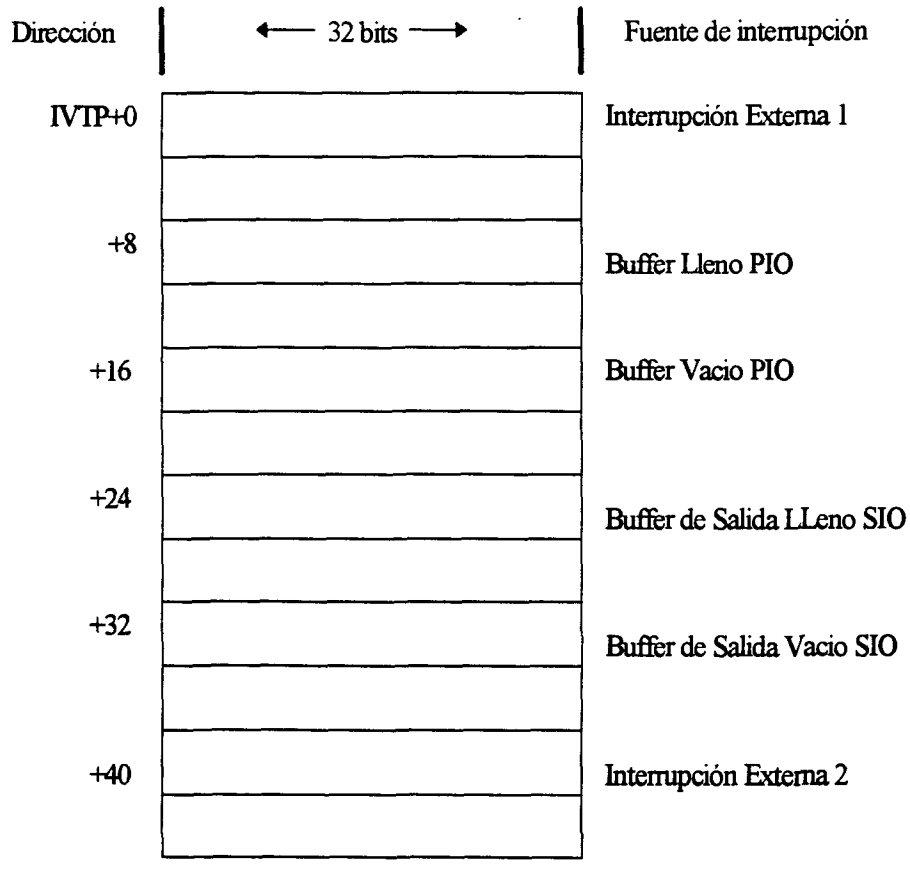
El DSP32C proporciona un solo nivel de interrupción y puede responder a seis fuentes, cuatro internas y dos externas. Las interrupciones son priorizadas y son

individualmente enmascarables mediante el campo INTER de la palabra de control del procesador (PCW) . Una tabla de vectores reasignable controla el flujo basado en la fuente de la interrupción. Las fuentes de interrupción son enumeradas en orden de prioridad descendiente.

1. Interrupción Externa Uno (INTREQ1).
2. Buffer Paralelo Lleno (PDF) - requerida cuando el registro PDR es cargado.
3. Buffer Paralelo Vacío (PDE) - requerida cuando el registro PDR es leído.
4. Buffer de Entrada Serie Lleno (IBF) - requerido cuando se pone el flag IBF.
5. Buffer de Salida Serie Vacío (OBE) - requerido cuando se pone el flag OBE.
6. Interrupción Externa Dos (INTREQ2)

En respuesta a una determinada interrupción, el DSP32C salta a la correspondiente dirección en la tabla de vectores de interrupción. La tabla de vectores de interrupción contiene ocho pares de palabras de 32 bits comenzando en una localización especificada el Puntero a la Tabla de Vectores de Interrupción (ITVP), registro r22. Antes de que una interrupción sea habilitada, el IVTP debe contener la dirección base de la tabla del vector de interrupciones. En la figura 11 podemos ver un mapa de la tabla del vector de interrupciones. Las últimas cuatro palabras del la tabla corresponden a dos fuentes de interrupción reservadas y no se usan normalmente.

Figura 11. Tabla de vectores de interrupción



3.10.4 Arbitro de Bus

El DSP32C permite a dispositivos externos compartir el bus externo de memoria mediante un protocolo de petición/reconocimiento. El bus de interface de memoria externo consiste de un bus de direcciones (AB00-AB21), bus de datos (DB00-DB31). Un dispositivo externo solicita el interface de memoria poniendo a cero lógico el pin (BREQN) . Cualquier transacción en curso es terminada antes de ceder el bus. El DSP32C reconoce la petición poniendo el bus externo de interface en alta impedancia.

3.10.5 Estados de Espera Para Memoria Externa

El control de estados de espera de memoria permite al usuario usar el DSP32C en diferentes configuraciones. La memoria externa está dividida en dos secciones para acoplar hardware de diferentes velocidades: una partición baja (A) y una partición alta (B). La palabra de control del procesador (PCW) es puesta en el programa de aplicación para habilitar o desabilitar de forma independiente el generador de estados de espera para cada partición. El contenido de esos registros es cero, uno, dos, tres , o dos o mas estados de espera (usando SRDYN) para cada partición.

3.10.6 Parada

El modo de parada permite parar el procesador bajo tres formas diferentes.

- Un valor bajo en el pin externo de Reset. RESTN
- Un bit cero del registro de control paralelo I/O. PCR[0].
- Si ocurre un error no enmascarable interno.

En el modo de parada, las instrucciones en curso se terminan de ejecutar y se vacía el encauzador "pipeline". Mientras el DSP32C esta en modo Parada, el reloj continua funcionando. La CAU sitúa una instrucción "nop" el registro de instrucciones a continuación de la instrucción donde se paro el DSP32C. El contador de programa no avanza. La unidad SIO para todas las actividades de la DMA completando las transacciones en curso. Las otras actividades SIO continúan funcionando. La unidad PIO se mantiene activa durante el modo de parada y puede

ser controlada por dispositivos externos. Mientras el DSP está parado, el bus de interface de memoria externo puede se cedido a otros dispositivos.

3.10.7 Reset

El DSP32C es reseteado cuando se detectan alguna de las dos condiciones siguientes.

- Una transición de bajo a alto de pin RESTN.
- Una transición de bajo a alto en el bit cero del registro de control de E/S paralelo, PCR[0].

Si el DSP32C es parado debido a un nivel bajo en RESTN, las condiciones posteriores no causan el reset del dispositivo. Cuando comienza una secuencia de reset la CAU sitúa dos instrucciones en sus registros de instrucción. La primera instrucción es equivalente a call 0 (r14). Esta instrucción transfiere el contenido del contador de programa al registro 14 de la CAU y borra el PC (0x000000) provocando un salto a la posición de memoria 0. La segunda instrucción es de tipo nop. Después de esta instrucción la ejecución del programa comienza con la primera instrucción que encuentre en la posición de memoria cero.

Durante el Reset, todas las transferencias DMA se paran. En la tabla 2 se muestran los registros que se ven afectados durante el reset.

Tabla 2. Registros afectados por el reset.

Registro	Valor	Comentario
r14	PC Previo	Resultado de call 0 (r14)
PC	0	Resultado de call 0 (r14)
DAUC[4]	0	Por defecto redondeo durante la conversión de float integer en la DAU
EMR[15-0]	Todos puestos	Todas las condiciones de error enmascaradas.
ESR[7-0]	Todos Borrados	Todas las condiciones de error borradas.
IOC[20-0]	Todos Borrados	No SIO DMA
PCW[15-6]	Todos Borrados	Todas las fuentes de interrupción borradas.
PCW[5-0]	Todos Puestos	Las dos particiones de memoria externa configuradas para dos o mas estados de espera (usando SRDYN) configuradas como salida.
PCR[0]	1	DSP32C en modo Run
PCR[10-1]	Todos borrados	
Condiciones de los Flags de E/S	Todos borrados	Todas las condiciones de E/S puestas a cero

3.11 REGISTROS DE CONTROL

El DSP32C tiene tres registros de control: el registro de control de la unidad aritmética de datos la palabra de control del procesador, y el registro de control de E/S. Esos registros son escritos normalmente cuando comenzamos una aplicación y rara vez son modificados durante la ejecución del programa. Con ellos se configura al dispositivo en varios modos de operación. En los apartados siguientes describiremos cada uno de ellos.

3.11.1 Registro de Control de la Unidad Aritmética de Datos. (DAUC)

El DAUC controla los tipos de conversión llevados a cabo por la DAU en la entrada y salida de datos. Este registro incrementa la flexibilidad de la DAU sin complicar las instrucciones. El DAUC se escribe con una instrucción de la forma:

dauc=VALOR

donde VALOR es un número de 5 bits. DAUC[4] se borra durante el reset y se selecciona la conversión de float a entero, la que está por defecto.

3.11.2 Registro de Palabra de Control del Procesador (PCW)

El PCW en un registro interno de 16 bits que puede ser leído o escrito directamente usando instrucciones del DSP32C. Este registro no es directamente accesible por un dispositivo externo. La PCW es usada para configurar la interrupciones, los bits controlados por el usuario del puerto PIO (PIOP), y los estados de espera externos. La tabla 3 muestra las funciones de este registro.

Tabla 3. Registro de control de la palabra del procesador.

Bits(s)	Campo	Función
0	WB	Si es 0, desabilita el generador de estados de espera para la partición de memoria externa B. Si es 1, Habilita el generador de estados de espera para la partición de memoria externa B
1	WA	Si es desabilita el generador de estados de espera para la partición de memoria externa A Si es 1, Habilita el generador de estados de espera para la partición de memoria externa A
2,3	MEMB	Estos bits seleccionan el número de estados de espera que son generados para la partición de memoria externa B. 00 - 1 Estado de espera. 01 - 2 Estados de espera 10 - 3 Estados de espera 11 - Dos o más estados de espera (controlados por la señal SRDYN)
4,5	MEMA	Estos bits seleccionan el número de estados de espera que son generados para la partición de memoria externa A. 00 - 1 Estado de espera. 01 - 2 Estados de espera 10 - 3 Estados de espera 11 - Dos o más estados de espera (controlados por la señal SRDYN)
6	PIOPL	Si es 0, los pines PIOP[0-3] son entradas, si sin 1 los pines PIOP[0-3] son salidas.
7	PIOP	Si es 0, los pines PIOP[4-7] son entradas, si son 1 los pines PIOP[4-7] son salidas.
8	MGM	Si es 0, MGM actúa como una salida de habilitación de memoria, si es uno, MGM es usado por el bus como árbrito de protocolo.

9	PEND	Si PCW[8]=0, PEND no es usado. Si PCW[8]=1, El valor lógico de ese bit es un OR con una señal interna, la cual indica que un acceso externo ha sido pedido.
15-10	INTER	Cada bit de este campo corresponde a una de las ocho fuentes de interrupción. un valor 1 en una posición habilita la correspondiente fuente de interrupción, un valor de cero las desabilita. 10 - INTREQ2. Pin de interrupción dos. 11 - OBE. Buffer de salida serie Vacío. 12 - IBF. Buffer de salida serie lleno. 13 - PDE. Datos Paralelos lleno (salida) 14 - PDF. Datos paralelos llenos (entrada) 15 - INTRQ1. Pin de interrpción 1

3.11.3 Registro de Control de E/S (IOC)

El IOC se usa para configurar el interface serie de E/S a dispositivos externos. Este registro es usado para poner las configuraciones tales como, longitud de los bits, reloj interno o externo y sincronización interna o externa. El contenido del registro IOC es escrito por el programa con una instrucción del tipo:

ioc=VALOR

donde VALOR representa un numero de 21 bits. En la tabla 4 se describe este registro.

Tabla 4. Descripción del registro IOC.

Bits(S)	Campo	Función
0	ASY	Si es 0, SY es externo. Si es 1 SY es interno. Cuando es generado internamente SY={ICK,OCK}/{256,512,1024}=IOC[BC]/(32xIOC[SLEN]).
1	BC	Si es 0, ICK es usado para derivar la carga del chip y los relojes SY. Si es 1, OCK es usado para derivar la carga del chip y los relojes SY.
2,3	SLEN	Estos bits seleccionan la relación de frecuencia de reloj de carga del chip. 00- No usado 01 - Relación = 8 10 - Relación = 16 11 - Relación = 32
4	AIC	Si es 0, ICK es externo. Si es 1 ICK es generado internamente con frecuencia de CKI/8 o CKI/24, basa en IOC[18]

5	AIL	Si es 0, ILD es externa. Si es 1 ILD es generada internamente con frecuencia de {ICK, OCK}/32=IOC[BC]/32.
6,7	ILEN	Estos bits especifican la longitud de los datos serie de entrada. 00 - Longitud de los datos serie de entrada 32 bits (antes de ILD) 01 - Longitud de los datos serie de entrada 8 bits (después de ILD) 10 - Longitud de los datos serie de entrada 32 bits (después de ILD) 11- Longitud de los datos serie de entrada 32 bits (después de ILD)
8	AOC	Si es 0, OCK es externo. Si es 1 OCK es internamente generado con una frecuencia de CKI/8 o CKI/24, basada en IOC[18]
9	AOL	Si es 0, OLD es externo. Si es 1 OLD es internamente generado con una frecuencia de valor {ICK, OCK}/32=IOC[BC]/32
10,11	OLEN	Estos bits especifican la longitud de los datos de salida serie. 00 - Usado con el bit 19 (O24)* 01 - Salida de datos serie de longitud 8 bits. 10 - Salida de datos serie de longitud 16 bits 11 - Salida de datos serie de longitud 32 bits.
12	SAN	Si es 0 borra la sanidad. Si es 1, pone la sanidad.
13-15	DMA	Estos bits controlan el acceso directo a memoria. 000 - Sin DMA 001- Entrada DMA cuando IBF esta a nivel alto. 010 - Salida DMA cuando OBE esta a nivel alto 011 - Entrada DMA cuando IBF esta alto y salida DMA cuando OBE es puesto alto 100 - Entrada y Salida DMA cuando IBF y OBF están a nivel alto. 101 - Entrada y salida DMA cuando IBF esta a nivel alto. 110 - Entrada y salida DMA cuando OBE esta puesto a nivel alto. 111 - Entrada y Salida DMA cuando IBF o OBF están a nivel alto
16	IN	Si es 0, el LSB es el primero en ser recibido durante la transmisión serie. Si es 1 el MSB el primero en entrar.
17	OUT	Si es 0, el LSB es transmitido primero. Si es 1 el LSB es transmitido primero.
18	CKI	Si es 0, el Clock SIO interno tiene el valor de CKI/8. Si es 1, el Clock SIO interno tiene el valor de CKI/24.
19	O24	Si es 0, la longitud de los datos de salida es especificada por el campo OLEN de IOC. Si es 1, OLEN=0, OUT=0 (LSB Primero), La longitud de los datos de salida es de 24 bits.
20	DSZ	Si es 0, el tamaño de los datos de entrada y de salida DMA es de 32 bits. Si es 1, el tamaño de los datos de entrada DMA se determina por el campo ILEN, y el tamaño de los datos DMA de salida es determinado por los campos OLEN y O24.

* Si O24=0, OLEN=00 no es usado

3.12 EL PUERTO DE E/S SERIE

El puerto serie de E/S (SIO) proporciona las comunicaciones serie y la sincronización del DSP32C con otros dispositivos externos. La sección de E/S serie proporciona puertos serie sincronos de doble buffer para las comunicaciones de

entrada y salida de datos digitales. Las entradas o salidas pueden ser manipuladas bajo programa, DMA o interrupciones. Las E/S serie pueden conectarse a un multiplexor por división de tiempo (TDM), codificadores serie u otros DSPs con pocos componentes adicionales.

El SIO está dividido en tres unidades funcionales: el puerto de entrada serie (SIP), El puerto de salida serie (SOP) y el generador de reloj. Existe además un registro que permite programar la configuración de esos unidades para varios modos de operación.

3.13 EL PUERTO DE E/S PARALELO (PIO).

El PIO soporta dispositivos externos de 8 y 16 bits. El PIO proporciona un interface asíncrono para los dispositivos externos. Existen dos modos de transferencia: direccionamiento directo a memoria (DMA) y E/S programadas. Usando la DMA del PIO, un dispositivo externo puede descargar o cargar un programa o datos sin interrumpir la ejecución del programa en DSP32C. Las transferencias entre el PIO y el DSP32C a memoria externa o interna pueden ser configuradas para ser realizadas con 16 o 32 bits. Usando E/S programada, el dispositivo externo y el DSP32C cooperan entre ellos para realizar la transferencia.

El PIO además soporta un mecanismo por el cual un dispositivo externo puede parar y resetear al DSP32C. El PIO monitoriza las condiciones de error del DSP32C, y el dispositivo externo puede ser notificado de lo ocurrido. Además el PIO es la

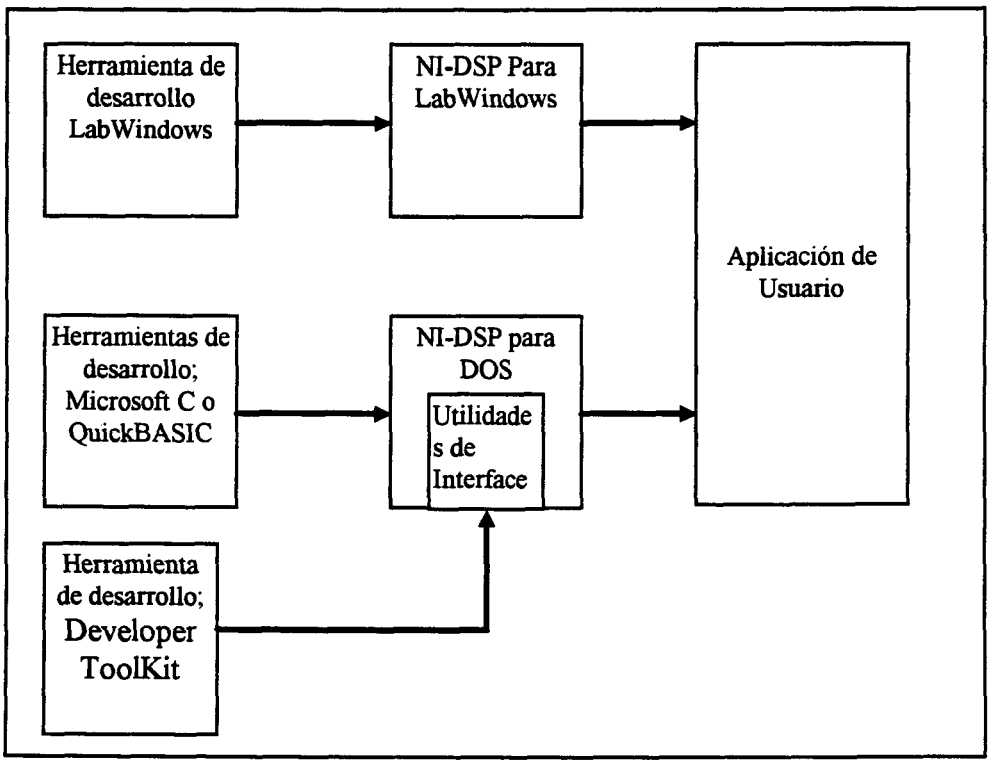
fuente de dos interrupciones. Las interrupciones del PIO indican si el PDR ha sido leído o escrito.

4. HERRAMIENTAS DE PROGRAMACIÓN PARA LA AT-DSP2200.

4.1 EL SOFTWARE NI-DSP

El paquete que contiene las NI-DSP está compuesto de dos partes; en la primera están los discos de las NI-DSP para DOS y en la segunda están los discos de las NI-DSP para LabWindows.

Figura 1. Posibles Caminos de desarrollo con las NI-DSP.



EL software NI-DSP es un herramienta que contiene una serie de Librerías de Análisis de alto nivel que se ejecutan sobre el Procesador Digital de Señal que incorpora la AT-DSP2200 y además contiene las Utilidades de Interface que son un conjunto de herramientas que junto con las Developer ToolKit permiten personalizar y crear nuevas Librerías.

Las Librerías NI-DSP vienen preparadas para ejecutarse con tres entornos de programación; Microsoft C , QuickBASIC y LabWindows. En la figura 1 se muestran las posibilidades de trabajo de las NI-DSP. Las librerías DSP que hemos nombrado están dentro de lo que se llama COFF Fichero Objeto de Formato Común. Este fichero constituye el software residente en la AT-DSP2200 el fichero COFF contiene el Kernel, rutinas de manejo de memoria, rutinas de control de ejecución, rutinas de comunicaciones, rutinas de manejo de interrupciones para la adquisición de datos, un conjunto de mas de 100 funciones de análisis. A todo esto es a lo que nos referimos cuando hablamos de las Librerías DSP.

4.1.1. Instalación de las NI-DSP Para DOS.

Para instalar la NI-DSP para DOS hay que introducir el disco uno y teclear la palabra SETUP desde la línea de comandos del DOS. La ejecución del programa SETUP da como resultado la instalación del software NI-DSP en un directorio, que puede ser el que crea el programa por defecto o uno especificado por el usuario. El programa de Instalación crea además una serie de directorios de trabajo, tabla 1.

Tabla 1. Directorios creados por el programa SETUP.

NOMBRE DEL SUBDIRECTORIO	DESCRIPCIÓN
LIB	Contiene los ficheros librería para el linkado de los programas
INCLUDE	Ficheros "include" asociados a las librerías y la utilidades.
INTERFAC	Contiene la Utilidad para Construir un interfas y los ficheros relacionados
	La Utilidad Dispatch y los ficheros relacionados.
DISPATCH	Contiene el código fuente para de los ejemplos.
EXAMPLES	

4.1.2 Instalación de las NI-DSP Para Labwindows.

Las NI-DSP para LabWindows están pensadas para ser usadas como las Librerías estándar de LabWindows y se instalaran en el directorio que contiene las Librerías de LabWindows. Es necesario por tanto tener instalado el LabWindows previamente, concretamente la versión 2.2 , de lo contrario no se podrá hacer uso de estas Librerías.

Si se cumplen las condiciones anteriores se podrá proceder a la instalación introduciendo el disco uno de las NI-DSP para LabWindows en la disquetera correspondiente y tecleando en la línea de comandos; A:\ SETUP , si el disco origen se encuentra en la disquetera A.

NI-DSP para LabWindows ofrecen soporte para crear programas bajo el entorno gráfico LabWindows. En la tabla 2 se ven los ficheros necesarios para poder desarrollar una aplicación con LabWindows. Las funciones de análisis se suministran en forma de funciones panel, esto permite manipular datos de forma interactiva.

Tabla 2 Lista de ficheros NI-DSP para LabWindows.

Nombre del Fichero	Descripción
DSP.LIB	Librería de Interface DSP. Contiene todas las llamadas a las funciones de usuario.
	Contiene las Librerías de Interface DSP
DSP.OVL	Fichero include para aplicaciones BASIC. Contiene los prototipos de todas
DSP.INC	llamadas a las funciones de usuario de DSP.LIB
	Fichero include para aplicaciones C. Contiene los prototipos de todas llamadas a
	las funciones de usuario de DSP.LIB
DSP.H	
	Kernel de la AT-DSP2200 para placas de 64 K
	.
DSP2200s.OUT	Kernel de la AT-DSP2200 para el resto de las placas
DSP2200.OUT	Modulo Objeto para crear un librería rápida de la DSP.LIB
DSP.OBJ	

También pueden ser instalados uno o mas ficheros sin reinstalar todo el paquete tecleando; A:\ SETUP -x y siguiendo las instrucciones que se van indicando.

4.1.3. Configuración del Software NI-DSP.

Antes de ejecutar las aplicaciones NI-DSP hay que configurar algunos parámetros como: El número de Slot que ocupa la placa, el subtipo de placa, la dirección base, el nivel de interrupción el canal de DMA y la ruta al directorio que contiene las Librerías DSP. Estos parámetros se establecen de forma diferente en función del tipo de bus donde se instale la AT-DSP2200; bus ISA o bus EISA.

Con el software NI-DSP se suministra un programa llamado DAQCONF.EXE que permite la establecer todos los parámetros de configuración señalados antes. Después de realizar la configuración, esta se guarda en el fichero ATBRDS.CFG.

A continuación se define cada uno de los apartados del menú de configuración y su significado.

1. **Board ID**. Se refiere al slot donde se conecta la AT-DSP2200. Cuando la placa esta conectada en un bus ISA este número no tiene que indicar el slot donde se conecte la AT-DSP2200, pero debe tenerse en cuenta para cuando se usen las Librerías NI-DSP o NI-DAQ. Este requisito vine derivado de la compatibilidad con el bus EISA pues en esta arquitectura de bus los slots están numerados y el software de cada placa debe saber donde debe ejecutarse.
2. **Board Type**. Tipo de placa, como el programa de configuración es igual para todas de National Instruments aquí hay que indicar el nombre de la tarjeta sobre la que va correr el software, en este caso ponemos; AT-DSP2200.
3. **Board SubType**. Subtipo de Placa, dentro de la AT-DSP2200 hay tres modelos en función de la cantidad de memoria de que dispongan, como la que tenemos es el modelo más alto de la gama con 384 Kword, debemos indicar; 128/256/384 K WORD.

4. **Base IO Address**. Dirección Base de E/S, con esto le indicamos al software la dirección del puerto de E/S de la AT-DSP2200 para que pueda comunicase con la

tarjeta. La dirección que hemos seleccionado es la 140 hex y debemos poner en este apartado (0x140).

5. **Interrup Level**. Nivel de interrupción, aquí seleccionamos el número de la interrupción hardware que será usada por la tarjeta para realizar las transferencias de datos desde esta al PC y viceversa. Hay que tener encuentra a la hora de seleccionar una interrupción que esta no sea usada por el PC o por otras placas que previamente puedan haber sido conectadas. En este caso la interrupción seleccionada es la 10 ya que en el PC donde se ha realizado la instalación esta no era usada

6. **DMA Chanel 1**. Canal 1 de DMA, Aquí se indica el canal de DMA para las transferencias de datos entre el PC y la AT-DSP2200 y viceversa. Como en el caso anterior hay que asegurarse de que el canal de DMA elegido no esté siendo usado por otro dispositivo o por el PC. En este caso se ha elegido el canal 6 ya que en el PC donde se ha realizado la instalación no era usado.

7. **DMA Chanel 2**. Este apartado es para el caso de tarjetas con dos canales de DMA. Como la AT-DSP2200 sólo tiene uno se deja en blanco.

Kernel Path. Ruta del Kernel, aquí se especifica la ruta donde se encuentra el Kernel o núcleo de la AT-DSP2200. El Kernel es un conjunto de Librerías que se cargan en la memoria del DSP32C. Si no está cargado el Kernel las Librerías NI-DSP no funcionan correctamente. Con las NI-DSP se suministran dos ficheros que contienen

dos núcleos distintos el DSP2200s.out y el DSP2200.out, el primero es para usar con el modelo de la AT-DSP2200 que tiene 64 K word y le segundo es el que su usa con los demás modelos. En este apartado podemos especificar la ruta completa hasta el directorio que contiene las Librerías, Kernel, incluyendo el nombre del fichero, o solo la ruta hasta el directorio que contiene el ficheros con las Librerías. En este caso nosotros debemos cargar luego un Kernel con una rutina que se suministra con las NI-DSP llamada DSP-Load. De esta forma podemos elegir entre cargar el Kernel de forma automática al ejecutar un programa o cargarlo dentro del programa. Con las Librerías de Interface y las Development ToolsKit podemos modificar o añadir nuevas rutinas y cargarlas en el DSP32C. En este caso, la opción elegida es la de cargar el Kernel de forma automática, para ello escribimos la ruta completa hasta donde se encuentra; **c:\nidsp\lib\dsp2200.out**.

Cuando hayamos realizado la configuración pulsamos F10 para salvar la configuración en el fichero ATBRDS.CFG. En el caso de que queramos modificar la configuración mas tarde podemos ejecutar otra vez el programa DAQCOF.EXE. Si no se indica ningún modificador el programa busca el fichero de configuración primero en el directorio de trabajo y luego en el directorio de raíz y si no lo encuentra crea uno nuevo, así si ya tenemos un fichero de configuración y no esta en el mismo directorio que el fichero DAQCONF.EXE podemos indicarle la ruta de la forma **DAQCONF \c:c600\compilar**. Cuando ejecutemos una aplicación NI-DSP el fichero de configuración debe ser accesible desde el programa de aplicación, para ello lo mejor es situarlo en el directorio raíz y así será siempre accesible desde cualquier aplicación.

Para el caso de instalar la AT-DSP2200 en un PC con bus EISA hay que hacer dos configuraciones diferentes.

Cuando se compra un PC con bus EISA se suministra un programa de configuración, normalmente llamado CF.EXE. Esta utilidad, junto con el fichero !NIC1100.CFG que es instalado por el programa SETUP en el directorio raíz, se usa para asignar el número de slot a la AT-DSP2200, para elegir el tipo de memoria, la dirección base, el nivel de interrupción y el canal de DMA. La forma de hacer esto es la siguiente; copiamos el programa CF.EXE en el directorio donde está el fichero !NIC1100 y lo ejecutamos realizando a continuación la configuración de los apartados anteriores. Después es necesario ejecutar el programa DAQCONF.EXE para indicar la ruta del Kernel.

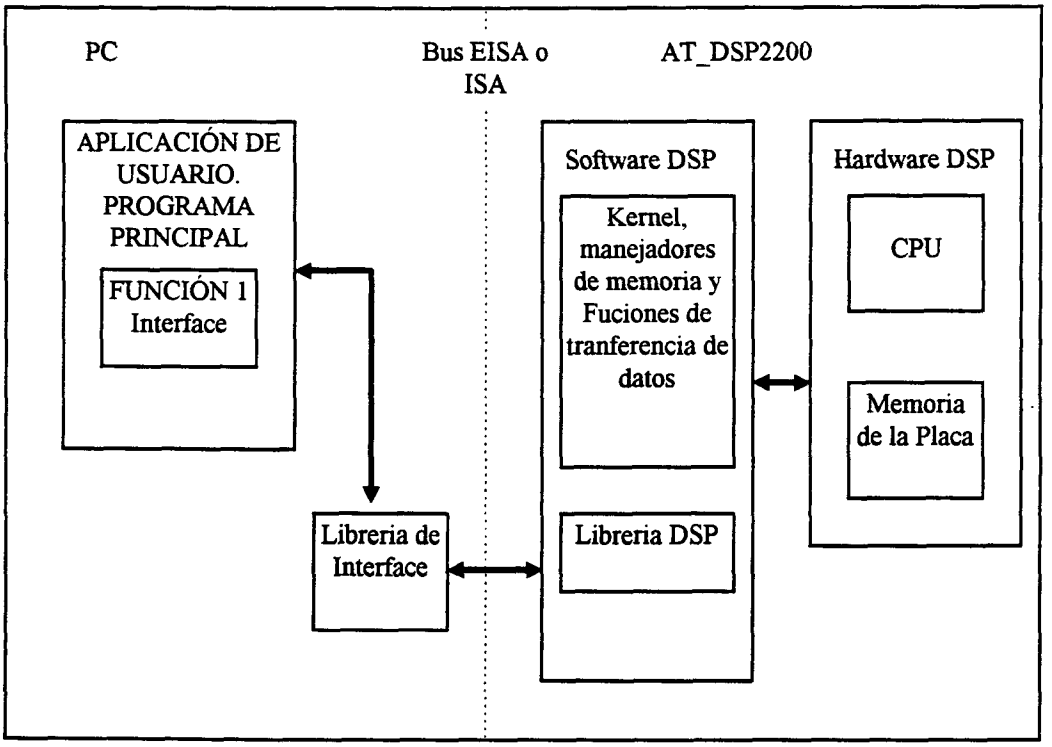
4.1.4 Las Librerías de Análisis NI-DSP

En los capítulos anteriores hemos visto como se instalaba el software NI-DSP y lo que contenía cada parte. Ahora vamos a ver como podemos usar ese software para programar la AT-DSP2200.

Como vimos en el capítulo dos la AT-DSP2200 trabaja de forma conjunta con el PC, como si fuera un procesador especializado y se relaciona con el PC mediante un bus de datos. Las Librerías DSP y las funciones de manejo y transferencia de datos residen y se ejecutan el la tarjeta, concretamente en el DSP32C. Los programas de aplicación y la Librería de Interface, residen y se ejecutan el PC. La AT-DSP2200 no tiene acceso a la memoria del PC por tanto las aplicaciones para la tarjeta deben ser descargadas en ella antes de poder ejecutarse. Una vez descargadas las aplicaciones

pueden funcionar de forma autónoma en la AT-DSP2200 sin la intervención del PC a menos que se requiera el uso de un recurso del PC.

Figura 2. Diagrama de como se comunica el PC con la AT-DSP2200 usando las NI-DSP.



La Librería de Interface, que reside en el PC, sirve como puente entre los programas de aplicación y las rutinas DSP ejecutándose en la placa, figura 2. La Librería de Interface interpreta las llamadas a las funciones desde el programa que está en el PC y decide que tipo de parámetros se pasaran a la tarjeta y cómo se situarán los datos en la memoria de las AT-DSP2200. Como ya dijimos antes todos los datos que vayan a ser procesados por la tarjeta deben residir en esta, por esto se proporcionan dos formas de manipular arrays de datos cuando llamamos a las funciones DSP. Cuando llamamos a una función DSP para representar un array los datos pueden ser pasados de dos formas mediante una variable array (una dirección de

memoria del PC) o mediante un "DSP Handle" (un código a una dirección de memoria de la placa). Para poder saca el máximo partido de la AT-DSP2200 es necesario comprender muy bien la diferencia entre lo que es una dirección PC y un DSP Hadle.

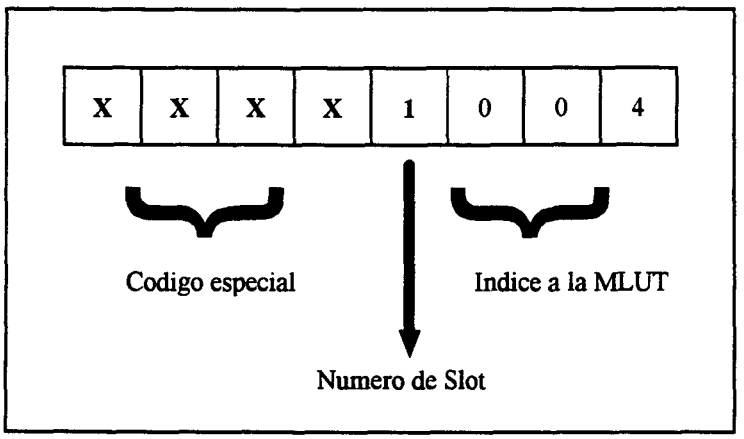
Pasar una función de usuario mediante una dirección PC da como resultado lo siguiente; los datos son copiados de la memoria de PC a la memoria de la placa, procesados y copiados de nuevo a la memoria de PC. Pasar una función mediante un DSP handle requiere que previamente se hayan copiado los datos en la memoria de la tarjeta. La Librería de Interface interpreta los argumentos de las llamadas de la función de usuario para determinar si los datos son pasados como direcciones PC o como DSP Handle. Si es una dirección PC, la Librería de Interface accede al manejador de memoria y a las funciones de transferencia de datos el la placa DSP para asignar un espacio de memoria y copiar los datos a la tarjeta antes de que la Librería DSP que ha sido llamada pueda operar con ellos. Si es un DSP Handle, la librería de Interface puede acceder a la Librería DSP directamente, porque los datos ya han sido copiados en la placa. Usar un u otra forma de pasar los datos depende de las necesidades concretas de cada momento. La primera forma requiere menos llamadas a funciones pero puede relentizar el conjunto de la aplicación debido a que es necesario transferir los datos entre el PC y la AT-DSP2200 en cada llamada a una función.

Las Librerías de Análisis NI-DSP contienen un conjunto de rutinas para manejar la memoria de la tarjeta e incrementar la velocidad y mejorar las

características de una aplicación. Hay rutinas para asignar buffers de memoria, para indexar en buffers y para desasignar buffers. Estas rutinas deben ser llamadas directamente cuando se pasan DSP Hadles como argumento de la función. Como ya dijimos antes cuando se pasa una dirección PC como argumento de la función, esas rutinas son llamadas automáticamente por la Librería de Interface.

La función DSP_Alloc, que asigna memoria en la tarjeta, devuelve un Handle DSP (un puntero) formado por un número entero de 32 bits que contiene información para indicar en que tarjeta se encuentra, para el caso de haber conectada varias, y un índice la Tabla de Consulta de Memoria (MLUT) que es donde se guarda la dirección DSP actual. En la figura 3 vemos como se codifica un DSP Hadle. El ejemplo es el resultado de la llamada, **hand1 =DSP_ Alloc(1,2048L),** que crea un buffer en la memoria de la tarjeta situada en el slot 1 de 2048 bytes.

Figura 3. Codificación de un DSP Handle.



Los primeros cuatro números que se indican como "codigo especial" son los que indican a la Librería de Interface, una vez decodificados, si el argumento es un DSP Handle o una dirección PC. Cuando la Librería de Interface no puede distinguir

con absoluta certeza si se trata de una dirección PC o de un DSP Handle, el Código se interpreta como una dirección PC. A continuación se explica el proceso exacto a seguir con cada uno de los códigos.

Dirección PC.

Este ejemplo de llamada de la función DSP_Sine para producir una señal senosoidal de 1024 puntos, se realiza mediante una dirección PC. Los pasos que realiza la Librería de Interface antes de que se ejecute la función son los siguientes.

DSP_Sine (1, 1024, 2.5, 75.0, 2.0, x)

- Inspecciona el argumento de llamada de la función y determina que es una dirección PC.
- Asigna un buffer del tamaño adecuado en la memoria de la tarjeta usando la función DSP_Alloc, obteniendo un DSP Handle de ese buffer.
- Ejecuta la función DSP_Sine que genera 1024 puntos en el buffer asignado con DSP_Alloc.
- Copia los datos generados al PC usando la función DSP_CopyMen
- Llama a la función DSP-Free para liberar el buffer asignado con DSP-Alloc.

Si se produce algún fallo en alguno de los pasos anteriores, la función que se llama finaliza y se devuelve un código de error .

DSP Handles.

Haremos lo mismo que en el caso anterior pero usando DSP Handles como argumentos de la función.

Handle = DSP_Alloc (3, 4096L)

DSP_Sine(1, 1024, 2.5, 75.0, 2.0, Handle)

Los pasos son los siguientes:

- La Librería de Interface reconoce *handle* como un DSP Handle devuelto por DSP_Alloc. Se asignan 4096 bytes porque los datos son números en coma flotante 1024 x 4.
- Retorna la dirección DSP actual del buffer de la MLUT y genera una señal sinosuidal en el buffer. El valor que está en el buffer *hand* puede ser usado por otras librerías DSP sin tener que copiar otra vez los datos desde el PC. Esta es la principal diferencia entre los dos modos de ejecutar funciones de la Librería DSP. En este segundo caso hay que tener en cuenta que el buffer sigue estando en la placa por lo que si no se va a usar más hay que liberar esa memoria de forma manual ejecutando DSP_Free.

4.1.5. Las Utilidades De Interface NI-DSP

En este apartado se describirá el funcionamiento de las utilidades de interface y su función pero sin adentrarnos demasiado en el, pues esta utilidad está pensada para trabajar con el kit de desarrollo, y ese programa no está disponible en el momento de hacer este trabajo.

Las Utilidades de Interface se usan fundamentalmente para dos cosas; para crear librerías a medida que residan el DSP, modificando las ya existentes o creando

nuevas rutinas y añadiéndolas a las ya existentes. La otra utilidad es crear un interface entre una función creada previamente y un programa residente en el PC.

A continuación se describirá como crear una función y añadirla a la librería y después, se explicará como crear un interface entre la función y un programa que se ejecute en el PC.

Para crear un librería a medida hay que seguir los siguientes pasos:

1. Crear el código fuente en C o ensamblador.
2. Añadir el nombre de fichero que hemos creado al fichero de linkado NIDSPLNK del directorio LIB.
3. Añadir el nuevo nombre de la función al fichero que contiene el listado de todas las funciones. Fichero NIDSP.FNC del directorio DISPATCH.
4. Ejecutar el programa **Dispatch.exe** del directorio DISPATCH para generar los ficheros **dspfncs.h** y **dispatch.s**.
5. Compilar , ensamblar y linkar la librería con el programa **makelib.bat** del directorio LIB. Para pode hacer esto hace falta los ficheros **libc32.a** ,**libap32.a** y **libm32.a** de las **developer ToolsKit.**

Si lo que se pretende es crear un interface para una determinada librería que hayamos creado los pasos son los siguientes.

1. Crear un fichero de descripción de la función.

2. Ejecutar el programa **bldinter.exe** del directorio BLDINTER que se encarga de un fichero de interface en C.

Nota: Cunado usamos las funciones NI-DSP que viene compiladas por el fabricante no es necesario crear un interface, solo tenemos que cargar el Kernel en el DSP tal como se indico en el apartado 4.4

4.1.6. Las Librerías NI-DSP

Las librerías NI-DSP son un grupo de funciones pensadas para realizar cálculos de grandes bloques numéricos usando las ventajas del DSP. Estas funciones pueden ser llamadas desde un aplicación que corra en el PC o en las AT-DSP2200 como posteriormente se explicará. Las llamadas para esas funciones vienen en forma de librerías en el fichero NIDSP.LIB. Además para que las funciones puedan correr en la tarjeta es necesario cargar el fichero DSP2200.out que contiene el Kernel con las librerías y las rutinas de comunicaciones como se indica en el apartado 4.4.

Las Librerías de Análisis pueden ser llamadas desde tres entornos de programación diferentes; Microsoft C, LabWindows y QuickBASIC.

Podemos agrupar las funciones por la función que realizan y así tenemos trece grupos que son los siguiente:

- Tratamiento de señales
- Funciones en el dominio de la frecuencia
- Funciones en el dominio de tiempo

- Funciones para filtrado
- Funciones "windows"
- Funciones Para el Tratamiento de Arrays
- Funciones lineales
- Funciones complejas
- Funciones estadísticas
- funciones para filtrado mediante "fitting"
- Funciones para el manejo y transferencia de datos

Funciones de utilidades.

4.2 EL SOFTWARE NI-DAQ.

Las NI-DAQ son un conjunto de funciones de alto nivel orientadas a la adquisición de datos. Las funciones NI-DAQ traen interfaces para un gran número de lenguajes de programación tanto para DOS como para Windows y también para LabWindows.

En la tabla 3 vemos una relación de dichos lenguajes.

Tabla 3. Lista de lenguajes soportados por las NI-DAQ

NOMBRE DEL LENGUAJE	ENTORNO
Microsoft C 5.0 5.1 6.0 y 7.0	DOS
Turbo C++ 1.0 2.0 y 3.0	DOS
Borland C++ 2.0 y 3.0	DOS
Borland Turbo Pascal 6.0	DOS
QuickBASIC 4.0 y 4.5	DOS

Professional BASIC 7.0 y 7.1	DOS
Visual BASIC para DOS	DOS
VISUAL BASIC 1.0	Windows
Turbo Pascal 1.0 y 1.5	Windows
Borland C++ 2.0 y 3.0	Windows
Microsoft SDK para Windows 3.0 y 3.1	Windows

Las NI-DAQ pueden usarse como funciones independientes o combinadas con las NI-DSP, formando una potente y fácil herramienta de trabajo para realizar la adquisición y procesado de datos.

4.2.1 Instalación de las NI-DAQ Para DOS.

Las NI-DAQ para DOS se instalan igual que las NI-DSP para DOS, por lo tanto no voy a repetir el proceso. En cuanto a la configuración con el fichero DAQCONF.EXE es indiferente que se haga con el que viene con las NI-DSP que con el de las NI-DAQ. Consultar el apartado 4.1.2 de este capítulo para más información.

4.2.2. Instalación de las NI-DAQ Para LabWindows.

Las NI-DAQ para LabWindows se instalan de la misma forma que las NI-DSP para LabWindows. Para poder realizar aplicaciones en el entorno LabWindows es necesario tener previamente instalada una versión 2.2 o superior de este programa. Cuando se instalan las NI-DAQ para LabWindows se copian los ficheros listados a

continuación en los directorios de LabWindows para proporcionar el soporte adecuado a la hora de realizar aplicaciones en este entorno.

- **DATAACQ.OVL.**

  Librería cargable de LabWindows la cual contiene las funciones librería NI-DAQ.

- **DAQ_AT.LIB.**

  Funciones librería NI-DAQ para usar en aplicaciones que usen únicamente tarjetas para PC, AT-DSP2200, y EISA.

- **DAQ_MC.LIB.**

  Funciones librería NI-DAQ para usar en aplicaciones que usen únicamente tarjetas para arquitectura Micro Chanel.

- **DATACQ.INC**

  Fichero para aplicaciones en BASIC, contiene los prototipos para todas las llamadas de usuario.

- **DATACQ.OBJ.**

  Módulo objeto para hacer una librería rápida de la DAQ_AT.LIB o DAQ_MC.LIB

- **DAQCONF.EXE.**

  Utilidad de configuración ya mencionada anteriormente.

4.2.3 Instalación de las NI-DAQ Para Windows.

Para instalar las NI-DAQ para Windows debemos tener previamente instalado el Windows en el PC. Introducimos el disco que contiene la parte de NI-DAQ para Windows en la disquetera correspondiente y tecleamos:

win a :\install

si no estamos dentro del programa Windows. Si estamos dentro, abrimos el administrador de archivos y seleccionando la unidad correspondiente ejecutamos el programa **INSTALL.EXE.** Durante la ejecución de este programa se nos va indicando lo que debemos hacer en todo momento.

Durante la instalación el programa advierte de la realización de algunos cambios en el fichero SYSTEM.INI perteneciente a Windows. Uno de los Cambios consiste en sustituir el Driver de Windows para el manejo de la DMA con memoria virtual por el que incorpora el programa. En el caso de que por alguna causa el programa de instalación no pueda realizar el cambio señalado antes en el fichero SYSTEM.INI.

Para realizar este cambio hay que editar el fichero SYSTEM.INI buscar la línea [386Enh] y a continuación escribir **device = *vdmad**

Cuando el programa haya sido instalado debemos ejecutar el fichero WDAQCONF.EXE, que es el equivalente al DAQCONF.EXE para DOS. cuando se ejecuta este programa aparece una ventana done elegimos la configuración correcta. el método para elegir la configuración adecuada es el mismo que para la configuración bajo DOS.

4.2.4 Las Funciones NI-DAQ

Las funciones NI-DAQ están agrupadas en bloques funcionales en relación a lo que hacen. En los siguientes apartados se describirán todos los grupos de funciones que soporta la AT-DSP2200.

4.2.4.1 Funciones de Inicialización y configuración general

Estas funciones se usan para configurar e inicializar el hardware y software.

Get_DA_Brds_Info.

Cuando se llama esta función da como resultado el código de la AT-DSP2200, la dirección base, el número de interrupción que usa, el canal de DMA que usa y el modo de adquisición que ha sido establecido, además da una indicación de los potenciales conflictos de configuración con otras placas que estén conectadas.

Get_NI_DAQ_Version.

Al llamar a esta función se obtiene la el número de la versión de las librerías NI-DAQ.

Init_DA_Brds.

Inicializa el hardware y software de La AT-DSP2200 a su configuración por defecto.

Master_Slave_config.

Configura a la AT-DSP2200 como maestra y una o mas AT-DSP2200 como esclavas.

Set_DAQ_Mode.

Permite Especificar una configuración distinta a la que viene por defecto en el uso de interrupciones y DMA en la transferencia de datos entre la placa y la memoria.

Trigger_Window_Config.

Configura el nivel de histerisis del trigger analógico.

4.2.4.2 Software de calibración.

Estas funciones realizan la configuración y calibración de la AT-DSP2200.

DSP2200-Calibrate.

Lleva a cabo un calibración de offset de los canales de entrada y salida de la AT-DSP2200.

DSP_Config.

Especifica las operaciones de traslado y demultiplexación que la AT-DSP2200 puede realizar con los datos de entrada y salida.

4.2.4.3. Grupo de funciones de entrada de simples.

Este grupo de funciones permiten leer un dato por uno o los dos canales de entrada. pero no permiten realizar un adquisición continuada de señales. Si no se desea cambiar la configuración por defecto a la que se inicializa la AT-DSP2200 sólo es necesario usar las funciones MAI_Read y MAI_Scale. La configuración por

defecto es; acoplo DC en los dos canales, los dos canales seleccionados, reloj interno seleccionado.

MAI_Clear.

Borra las FIFO de los A/D y todo lo relacionado con el circuito analógico de entrada.

MAI_Coupling.

Selecciona acoplo DC o AC para los dos canales analógicos de entrada

MAI_Read.

Realiza una lectura de los canales seleccionados.

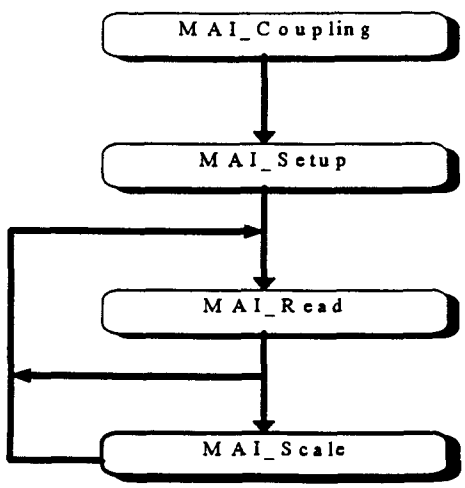
MAI-Scale.

Convierte los valores leídos por MAI-Read a valores de voltaje. La conversión la realiza mediante la fórmula.

MAI-Setup.

Selecciona el canal para ser leído. de los dos canales de entrada.

A continuación podemos ver un diagrama de flujo de un sobre el uso típico de estas funciones.



4.2.4.4. Funciones de entrada de bajo nivel.

La AT-DSP2200 sólo posee una función de entrada de bajo nivel , la función SCAN_Demux. Esta función permite demultiplexar los datos adquiridos por una operación de SCAN, como las anteriores, en filas, cada fila contiene a un canal. Esto permite que sea más fácil acceder a estos datos desde C. Para la aplicaciones en BASIC no es necesario reasignar los datos en filas pues el BASIC los accede mejor en columnas.

4.2.4.5 Funciones de adquisición de datos.

Las funciones de adquisición de datos a diferencia de las funciones anteriores permiten adquirir datos de forma continua a un buffer de memoria o al disco duro. Estas funciones permiten además el uso de triggers para realizar dicha adquisición. Al definir la Funciones de adquisición, a partir de ahora llamadas las funciones MADQ, se emplean algunos términos que conviene aclarar.

Frame: Un frame o trama es un conjunto de muestras adquiridas de todos los canales de entrada analógicos en cada disparo o señal de adquisición. El número de muestras por trama se define como:

Trama = (Pretrig_Scans + Pottrig_Scans) * Número de canales seleccionados.

- Scan: Un Scan es una muestra de cada uno de los canales seleccionados. Un Scan puede contener una muestra si solo se selecciona un canal o dos muestras si se seleccionan los dos canales.
- Scan_Interval: El intervalo entre muestras es el tiempo entre la iniciación de dos Scans consecutivas. Equivalente al tiempo entre muestras en un canal dado.
- Pretrig_Scans: Es el número de Scans a adquirir antes de una señal de disparo.
- Posttig_Scans: Es el número a adquirir después de un señal de disparo.

La adquisición puede ser orientada hacia tramas o hacia scans. Cuando se elige el modo de adquisición orientado a tramas, cada vez que se recibe una orden de adquisición se realiza la captura de una trama, una trama puede contener datos pretrigger y postrigger. Todas las tramas tienen el mismo tamaño , usan los mismos modos de disparo, el mismo número de canales y frecuencia de actualización. La adquisición de tramas termina cuando se recibe una orden de stop, a menos que se haya especificado un número de tramas fijo.

La adquisición orientada a Scan se realiza después de producirse un disparo. Entonces se realiza la adquisición de un número de canales determinado y hasta que se pare la adquisición. También puede especificarse un número de scans determinado.

MDAQ_Check.

Devuelve el estado de la adquisición de datos, y también el número de tramas o scans adquiridas.

MDAQ_Clear.

Para la adquisición, prepara el sistema para terminar la adquisición.

MDAQ_Ext_Setup.

Selecciona la cantidad de datos a ser almacenados en memoria extendida, la cantidad de datos adquiridos con cada disparo, y si la adquisición es orientada a scans o trama. Esta función sólo está disponible para el DOS:

MDAQ_Get.

Transfiere datos desde el buffers de adquisición a un buffer especificado por el programador, esta tarea la hace mientras la adquisición esta en progreso o después de que la adquisición ha terminado.

MDAQ_ScanRate.

Selecciona la frecuencia de adquisición **MDAQ_Setup.** Selecciona la cantidad de dato al almacenar en el buffer de adquisición de memoria baja, la cantidad de datos a adquirir en cada disparo y si la adquisición está orientada a scans o tramas.

MDAQ_Start.

Comienza la operación de adquisición.

MDAQ_Stop.

Para la Adquisición de datos manteniendo la configuración.

MDAQ_Trig_Delay.

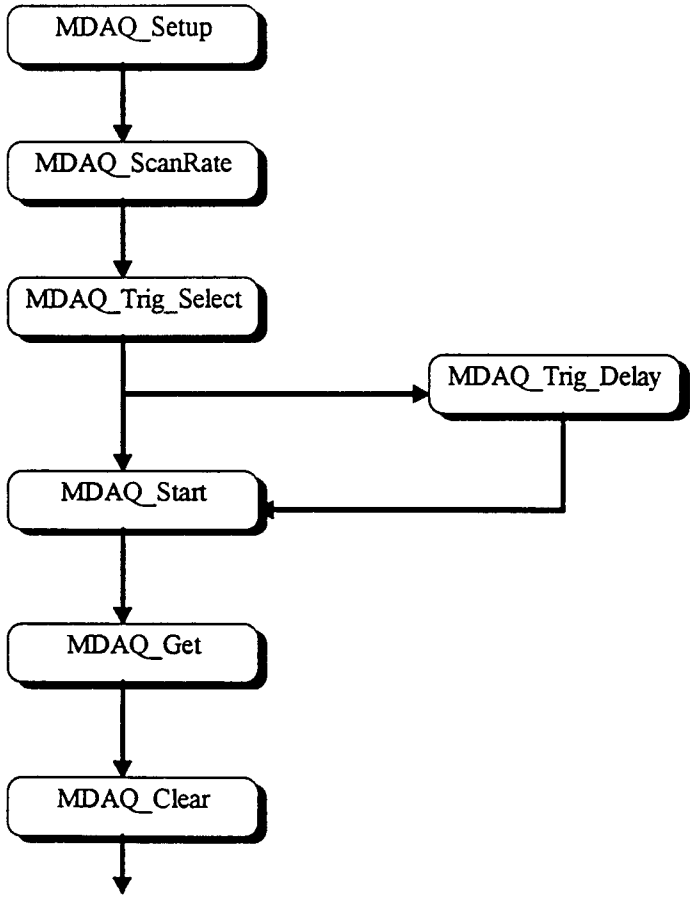
Selecciona el tiempo de retardo antes de iniciar la adquisición después de que se reciba una orden por algunos de los triggers.

MDAQ_Trig_Select.

Selecciona las fuentes de disparo y configura las condiciones de disparo analógicas o digitales.

En la figura 5 se ve un diagrama de flujo que muestra la forma de operar de las funciones comentadas anteriormente.

Figura 5. Diagrama de flujo da la operación de las funciones de adquisición



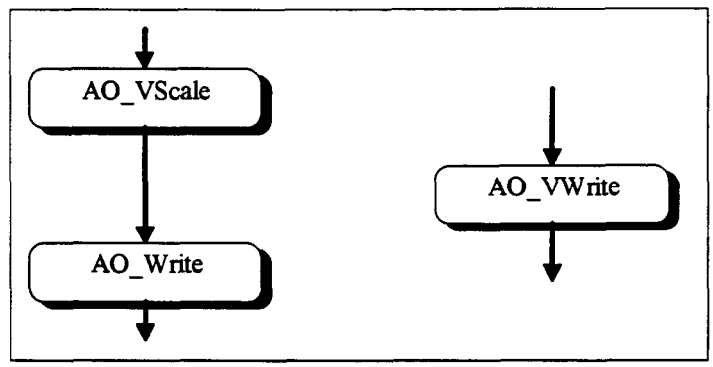
4.2.4.6. Funciones de analógicas de salida.

Estas funciones se usan para ejecutar conversiones D/A simples, como escribir un dato en un puerto. Las funciones son las siguientes.

AO_VScale.

Convierte un voltaje en coma flotante a binario, para escribirlo en los DACs de salida.

Figura 7. Diagrama de flujo de las funciones de salida analógicas



AO_Write. Escribe un valor binario a uno de los canales analógicos de salida.

AO_VWrite. Acepta un valor de voltaje en coma flotante y lo convierte en un valor binario adecuado para ser escrito en los DACs. Esta función además escribe el valor en los DACs. La función AO_VWrite es equivalente a las dos anteriores.

En la figura 7 podemos ver un diagrama de flujo del funcionamiento de las funciones mencionadas antes.

4.2.4.7. Funciones de generación de forma de onda.

Las funciones de generación de forma de onda son un conjunto de rutinas que permiten realizar una generación de ondas de forma continua desde el disco o desde

un buffer de memoria. Este conjunto de funciones realiza llamadas a funciones de bajo nivel que se describirán en el apartado siguiente.

WFM_from_Disk.

Asigna un fichero de disco a uno o mas canales de salida, además permite seleccionar la frecuencia y el números de veces que le fichero será generado. Esta función se ejecuta hasta que se haya terminado la generación.

WFM_Op.

Asigna un buffer de memoria a uno o más canales analógicos de salida, seleccionando la frecuencia y el número de veces que será generado.

4.2.4.8 Funciones de generación de formas de onda de bajo de nivel

Estas funciones permiten la el control de la generación de formas de ondas.

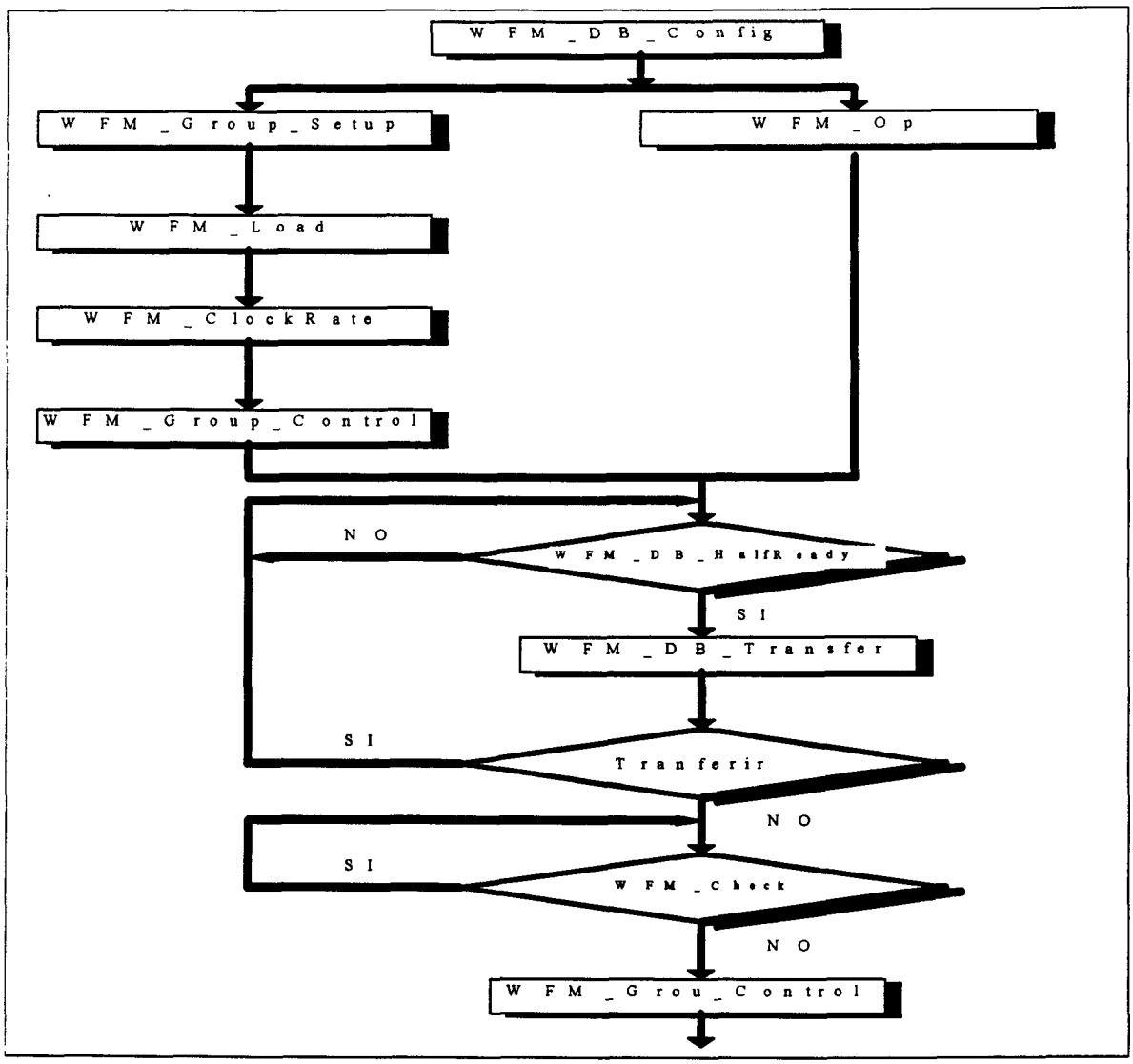
WFM_Chan_Control.

Para o reinicializa la generación de formas de onda para el caso de usar un canal.

WFM_Check.

Da información sobre la operación de generación de forma de onda.

Figura 8. Diagrama de flujo del uso de las funciones WFM.



WFM_ClockRate.

Indica la frecuencia de actualización de los DACs de salida.

WFM_DB_Config.

Habilita o desabilita la generación con doble buffer.

WFM_DB_HalfReady.

Comprueba si la mitad del buffer está disponible para un nuevo dato durante la operación de generación con doble buffer.

WFM_DB_StrTranfer.

Transfiere un nuevo dato de un buffer de datos a uno o más buffers de generación (seleccionados con WFM_Load) cuando la generación se está realizando. Esta función espera hasta que los datos del buffer de datos puedan ser transferidos a el buffer de generación.

WFM_DB_Transfer.

Transfiere un nuevo dato a uno o más buffers de generación (seleccionados con WFM_Load) cuando la generación está realizándose. Esta función espera hasta que el dato del buffer inicial pueda ser transferido al buffer de generación.

WFM_Group_Control.

Controla un generación de forma de onda para un grupo de canales analógicos de salida.

WFM_Group_Setup.

Asigna uno o más canales a un grupo de generación de formas de onda. El grupo por defecto para la AT-DSP2200 es el uno.

WFM_Load.

Asigna un buffer a uno o más canales de salida analógicos e indica numero de ciclos de formas de onda a generar.

WFM_Scale.

Convierte un array de datos en punto flotante a un array binario.

Tal y como hemos podido ver hasta ahora existen dos forma de generar ondas o reproducir ficheros con la AT-DSP2200. Los dos modos producen los mismos resultados sólo que uno es más simple de implementar que el otro. En el caso de las

rutinas de bajo nivel además existe un control mucho mayor por parte del programador sobre la generación, aunque la dificultad de realizar el programa también aumenta. En la figura la figura 8 podemos ver un diagrama de flujo de las funciones WFM de bajo nivel.

4.2.4.9 Funciones por disparo le Bus RTSI.

La AT-DSP2200 tiene tres señales que pueden ser conectadas a las líneas de disparo del bus RTSI. Estas señales del bus RTSI con sus códigos se muestran el tabla 4.

Tabla 4. Señales de interrupción del bus RTSI.

Nombre de la Señal	Dirección de la Señal	Código de la Señal
HWTrig*	Bidireccional	0
WCAD	Fuente	1
RTSIrig*	Fuente	2

RTSI_Clear.

Desconecta todas las líneas de disparo del bus RTSI en la placa especificada.

RTSI_Clock.

Conecta o desconecta el sistema de reloj del bus RTSI.

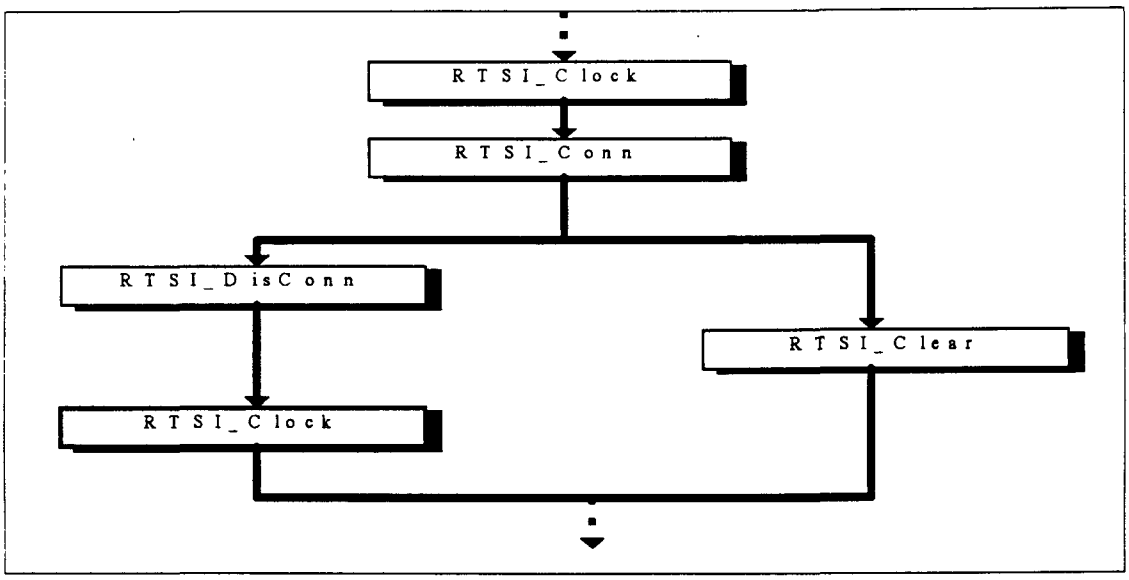
RTSI_Conn.

Conecta una señal de la placa a la línea de disparo especificada del bus RTSI.

RTSI_DisConn.

Desconecta una señal de la placa de una línea de disparo especificada del bus RTSI.

Figura 9 Diagrama de flujo del uso del bus RTSI



Una aplicación que use el bus RTSI se realiza en tres pasos básicos.

1. Conectar las señales de la AT-DSP2200 al bus RTSI.
2. Ejecutar la aplicación
3. Desconectar las señales del bus RTSI de las AT-DSP2200 .

En la figura 9 vemos un diagrama de flujo de como se hacen estos pasos.

4.3 LA TÉCNICA DEL DOBLE BUFFER

Las técnicas de adquisición de datos basadas en un sólo buffer suelen funcionar bastante bien en la mayoría de los casos. No obstante en los casos en los que se necesita adquirir o generar datos a altas velocidades y en tiempo real el sistema de buffer simple no suele funcionar de forma correcta. National Instruments ha desarrollado una técnica llamada doble buffer que consiste en un buffer circular dividido en dos mitades. La técnica del doble buffer se aplica tanto para las operaciones de entrada como las de salida.

Cuando usamos el doble buffers para operaciones de entrada los datos de entrada comienzan escribiéndose en la primera mitad del buffer circular y luego en la segunda mitad del buffer, mientras se escribe en la segunda mitad, las rutinas NI-DAQ copian los datos desde la primera mitad al buffer de transferencia y de aquí al disco. Después de que se haya llenado la segunda mitad del buffer circular, se empieza otra vez con la primera mitad del buffer y sobrescribe los datos con nuevos datos. Las rutinas NI-DAQ deben copiar ahora los datos de la segunda mitad antes de que se termine de escribir la primera mitad. El proceso se repite de forma indefinida..

Cuando usamos doble buffers para operaciones de salida de datos el funcionamiento es similar al proceso anterior. La operación comienza escribiéndose datos en la primera mitad del buffer circular. Después se generan los datos de la segunda mitad del buffer circular, entonces NI-DAQ copia los datos del buffer de transferencia a la primera mitad del buffer circular. Cuando se haya generado la

segunda mitad del buffer circular se comienza generando la primera mitad y otra vez las rutinas NI-DAQ cargan los datos del buffer de transferencia a la segunda mitad del buffer circular.

Los problemas que pueden ocurrir con el doble buffer son de dos tipos; el primero es que los datos se sobrescriban en el buffer circular antes de que puedan ser generados o pasados al buffer de transferencia, el otro problema es que los datos sean sobrescritos en el buffer de transferencia antes de que se puedan guardar o generar.

4.4 PROGRAMACIÓN CON LAS NI-DSP.

Como ya hemos dicho en el apartado 4.1 las funciones NI-DSP se pueden usar con tres entornos de programación; Microsoft C, QuickBASIC y LabWindows. Las funciones NI-DSP se llaman de cada uno de estos entornos como cualquier función. Los pasos a seguir para crear programas con cada uno de estos entornos de programación se describen a continuación.

4.4.1 Método para Realizar una Aplicación usando las Funciones NI-DSP En LabWindows.

Existen varias formas de crear un programa desde el entorno LabWindows, en los siguientes apartados veremos cuales son estas formas.

- Escribir la aplicación en el editor de LabWindows. Crear un fichero ejecutable usando el comando **Create.EXE** del menú del editor. Ejecutar **LWMAKE** mediante el comando **Create.EXE**
- Escribir parte de la aplicación en un editor normal. Usar luego el comando **Create.EXE** para crear una aplicación tipo para usar luego con LWMAKE. Completar luego la aplicación en el mismo editor. Ejecutar el comando LWMAKE desde la línea de comandos del DOS para compilar y linkar la aplicación.
- Escribir la aplicación en un editor normal. Ejecutar LWMAKE desde la línea de comandos del DOS, introducir toda la información necesaria y guardar los resultados en una aplicación tipo. Luego usa LWMAKE para compilar y linkar la aplicación.
- Escribir toda o parte de la aplicación fuera del entorno de LabWindows. Usar LWMAKE solo para crear un fichero de ejecución por lotes (.bat) y otro con directivas para el linkador (.LNK).

4.4.2 Método para Realizar una Aplicación usando las Funciones NI-DSP Con Microsoft C o QuickBASIC.

Si la configuración del software NI-DSP se ha producido de forma correcta, siguiendo los pasos que se describen a continuación podremos realizar programas en Micrsoft C y QuickBASIC que contengan funciones NI-DSP. En el caso de Microsft C debemos tener la librería LLIBCE.LIB y compilar los programas en el modelo largo de memoria. En el caso de QuickBASIC no debemos borrar ningún módulo objeto de la Librería de Interface DSP.LIB.

Para realizar un programa con Microsoft C hacemos lo siguiente:

- Escribir el programa en un editor que genere código en ASCII.
- Incluir el fichero cabecera **nidaqdsp.h**, qué contiene la definición de todas las funciones NI-DSP.
- Escribir la llamada a la función NI-DSP que se desea. Si la función se ejecuta como una función residente en el PC hay que indicar en la llamada el número de slot donde está la tarjeta. En el caso de que la aplicación sea una aplicación residente en la AT-DSP2200 no es necesario indicar el slot.

  Llamada a una función DSP residente en el PC:

  DSP_Add(board, x, y, samples, z);

  Llamada a una función DSP residente en la AT-DSP2200 :

  DSP_Add(x, y, samples,z);

- Cuando hayamos terminado de escribir el programa lo compilamos en el modo de memoria largo. Para esto desde la línea de comandos escribimos **cl /c /AL nombrePrograma.c.**
- Después de linkar el programa lo compilamos incluyendo la librería nidsp.lib. Para hacerlo desde la línea de comandos escribimos **link nombreProgrma ,,, nidsp.**

Después de hacer esto tenemos un programa ejecutable listo para funcionar.

4.4 PROGRAMACIÓN CON LAS NI-DAQ.

Las funciones NI-DAQ permiten ser implementadas en más lenguajes y entornos que las NI-DSP. La restricción en cuanto a entornos de programación de las

anteriores funciones en lo que motivó de forma determinante la elección del lenguaje Microsoft C para realizar la aplicación objeto de este trabajo.

En la líneas siguientes trataré de explicar los pasos a seguir para desarrollar aplicaciones en cada uno de los lenguajes listados en la tabla 3 de la sección 4.2.

4.5.1 Método para Realizar una Aplicación usando las Funciones NI-DAQ con Microsoft C en Entorno DOS.

Para poder desarrollar aplicaciones con Microsoft C que usen las NI-DAQ tenemos que tener la versión 5.0 o posterior del compilador y la versión 3.61 o superior del linkador.

- Escribir el código fuente de la aplicación en un editor que genere ficheros ASCII. Incluir los ficheros cabecera **nidaq.h**, que contiene la definición de los prototipos de las funciones NI-DAQ, y **nidaqerr.h,** que contiene los códigos de error de las funciones NI-DAQ.
- Compila el código fuente usando el modelo largo de memoria de la forma

  cl /c /AL programa.c
- Linkar el código objeto resultante del paso anterior con la librería. ATDAQ_C.LIB.
-

  link programa ,,, ATDAQ_C.LIB

Como resultado tendremos un programa ejecutable que hace llamadas a las funciones NI-DAQ.

4.5.2 Método para Realizar una Aplicación usando las Funciones NI-DAQ con QuickBASIC o Profesional BASIC en Entorno DOS.

Para poder desarrollar aplicaciones en este leguaje debemos disponer de una versión del compilador de QuickBASIC 4.0 o superior o las versiones 7.x de Profesional BASIC. Los pasos a seguir son los siguientes:

1. Debemos cargar la Librería Quick de las funciones NI-DAQ
2. Añadir la siguiente línea al principio de fichero fuente. Esta línea contiene las declaraciones de las funciones de la librería NI-DAQ.

   REM $INCLUDE:'NIDAQ.INC'
3. Escribir el código fuente haciendo las llamadas a las funciones NI-DAQ como cualquier función. Tenemos que tener en cuenta que cuando llamamos una función NI-DAQ en QuickBASIC hay que sustituir el subrayado por un punto, por ejemplo cuando llamamos a la función AI_Config, debemos hacer AI.Config.
4. Si se ha cargado la librería Quick de las funciones NI-DAQ del apartado 1, el programa puede ser ejecutado inmediatamente. En caso contrario se puede crear una aplicación que se ejecute fuera del entorno de QuickBASIC o BASIC 7.x , para ello hay que hacer lo que se indica en los dos pasos siguientes.

- Compilar el código fuente con uno de los dos compiladores mencionados antes. Desde la línea de comandos escribimos:

  Para QuickBasic:

bc rograma.bas (cuando linkamos con brun40.lib, brun45.lib o brt70enr.lib)

bc /o programa.bas (cuando linkamos con bcom40.lib,bcom45, o

c170enr.lib).

Para BASIC 7.x:

bc /o programa.bas

Linkar el fichero objeto con una versión 3.61 o superior de Microsoft Overlay Linker. El linkado se puede hacer de dos forma en función de las necesidades de cada aplicación. Si se crea un fichero ejecutable que necesite acceso al módulo run-time BRUN40.EXE o BRT70ENR.EXE debemos escribir desde la línea de comandos lo siguiente;

Versión de BASIC	LÍNEA DE COMANDOS
QuickBASIC	link /NOE/NOD programa ,,, brun40 ATDAQ_C SUPPORT
BASIC 7.0	link /NOE/NOD programa ,,, brt70enr ATDAQ_C SUPPORT
BASIC 7.1	link /NOE/NOD programa ,,, brt71enr ATDAQ_C SUPPORT

Si sólo se se quiere realizar una aplicación ejecutable única hay que escribir en la línea de comandos lo siguiente:

Versión de BASIC	LÍNEA DE COMANDOS
QuickBASIC	link /NOE/NOD programa ,,, bcom40 ATDAQ_C SUPPORT
BASIC 7.0	link /NOE/NOD programa ,,, bcl70enr ATDAQ_C SUPPORT
BASIC 7.1	link /NOE/NOD programa ,,, bcl71enr ATDAQ_C SUPPORT

La ventaja del primer método es que el tamaño del de fichero ejecutable que es más pequeño. La desventaja es que la ruta hasta el módulo run-time BRUN40.EXE o

BRT70ENR.EXE debe ser suministrada mediante el teclado en el momento de la ejecución

Se puede reducir el tamaño Librerías Quick NI-DAQ para DOS, dejando más memoria par programas y datos, compilando los programas de forma que contengan solo las llamada a las funciones que se van a usar en ese programa y "linkando" el módulo objeto con la librería NI-DAQ apropiada. Supongamos que vamos a crear un programa que llame a las funciones NI-DAQ AI.Read y AI.Vscale. Construimos una librería Quick ATDAQ4x.QLB como se describe a continuación:

- Crear un fichero fuente copiando las declaraciones de funciones para las funciones mencionadas antes. Al fichero fuente lo llamamos programa.bas.

```
DECLARE FUNCIÓN AI.READ% CDECL ALIAS "_AI_READ"
(BYVAL a%, BYVAL c%, SEG d%)
DECLARE FUNCTION AI.SCALE % CDECL ALIAS "_AI_VSCALE"
(BYVAL a%, BYVAL c%, SEG d#)

err.num% = AI.Read (brd%, canales%, ganancia%, valores%)
err.num% = AI.Scale (brd%, ganancia%, valores%, voltaje#)
```

- Compilar el fichero fuente.

```
bc programa.bas
```

- Linkar el fichero objeto producido en el apartado anterior con la libreria NI-DAQ adecuada. Llamaremos nuestra librería Quick AIAO.QLB.

link /Q/NOE/NOD programa AIAOSTUB.OBJ, AIAO.QLB,, ATDAQ_C+BQLB40+SUPPORT.

- Cargar la librería personalizada en el entorno BASIC. La forma es:

  qb /l aiao.qlb

Nota: Como en QuickBASIC no se puede linkar más de una librería cuando hacemos una librería ejecutable hay que combinar la librería de soporte SUPPORT.LIB con la librería ATDAQ_C.LIB usando el fichero LIB.EXE.

4.5.3 Método para Realizar una Aplicación usando las Funciones NI-DAQ con Borland Turbo C, Borland Turbo C++ O Borland C++ en Entorno DOS.

El proceso para realizar una aplicación con los lenguajes mencionados antes es:

1. Escribir el código fuente de la aplicación en un editor que genere ficheros ASCII. Incluir los ficheros cabecera **nidaq.h**, que contiene la definición de los prototipos de las funciones NI-DAQ, y **nidaqerr.h,** que contiene los códigos de error de las funciones NI-DAQ.
2. Compilar el código fuente con Turbo C 2.0, Turbo C++ o Borland C++ usando el modelo de memoria largo seleccionando -ml en la línea de comandos.
3. Linkar el fichero objeto con la librería NI-DAQ adecuada como se indica en la tabla.

Compilador	Librerías
Turbo C 2.0	ATDQTC2.LIB
Turbo C++	ATDQTCPP.LIB
Borland C++	ATDQBCPP.LIB

4.5.4 Método para Realizar una Aplicación usando las Funciones NI-DAQ con Borland Turbo Pascal en Entorno DOS.

Para crear una aplicación en Turbo Pascal que haga uso de las funciones NI-DAQ, se siguen los siguientes pasos.

1. Escribir el código fuente de la aplicación en un editor que genere ficheros ASCII.
2. Añadir la unidad NIDAQ a la clausula USES en el código fuente.
3. Cuando se compile el programa debemos tener en cuenta que Turbo Pascal pueda encontrar la unidad NI-DAQ.

4.5.5 Método para Realizar una Aplicación usando las Funciones NI-DAQ con LabWindows en Entorno DOS.

Al instalar las NI-DAQ para LabWindows se instalan en los directorios de este programa y por tanto el desarrollo de una aplicación se hace igual que cualquier otra aplicación LabWindows El método para crear aplicaciones usando LabWindows es idéntico al descrito en el apartado 4.3.2

4.5.6 Método para Realizar una Aplicación Usando las Funciones NI-DAQ con Microsoft C y SDK en Entorno Windows.

Para crear aplicaciones bajo entorno Windows con el compilador Microsoft C hay que tener en cuenta algunas consideraciones especiales. Por ejemplo para asignar

memoria hay que usar las funciones de Windows **GlobalAlloc()** y **GlobalFree()** en lugar de las funciones **malloc()** y **free()**. Después de asignar la memoria hay que bloquearla con **GlobalLock()**. o desbloquearla una vez usada con **GlobalUnlock().**

A continuación se indican los pasos a seguir para realizar una aplicación.

1. Con un editor que genere código ASCII crear tres ficheros; un fichero con el código fuente tipo *.C un fichero de recursos tipo *.rc y un fichero de definiciones para el linkador tipo *.def. Si se incluye una llamada a funciones NI-DAQ hay que incluir el fichero cabecera **WDAQ_C.H** que contiene la definición de los prototipos de la función.
2. Compilar el fichero fuente con el compilador de Microsoft usando dos modificadores -Gw y -Zp de la forma:

   cl -c -As -Gsw -Oas -Zpe programa.c
3. Linkar los módulos con el linkador de Microsoft tal como se indica:

   link /NOD programa,,, ATWDAQ libw slibcew, programa.def
4. Compilar el fichero de recursos dentro del fichero .res y combinarlo con el fichero con el fichero ejecutable usando el compilador que está contenido en la utilidad SDK.

   rc progrma.res

4.5.7 Método para Realizar una Aplicación usando las Funciones NI-DAQ con Borland C++ en Entorno Windows.

Los pasos son los siguientes:

1. Abrir un módulo de proyectos.
2. Crear un fichero del tipo .cpp con el código fuente. Tenemos que incluir el fichero cabecera **WDAQ_BC.H** que contiene las definiciones de los prototipos de las funciones.
3. Poner opciones/aplicaciones a *Windows App.*
4. Crear un fichero de recursos .res usando el Borland Whitewater Resource Toolkit. Después añade este fichero a la lista de ficheros en fichero proyecto.

4.5.8 Método Para Realizar Una Aplicación Usando Las Funciones NI-DAQ Con Turbo Pascal En Entorno Windows.

Para realizar una aplicación con Turbo Pascal seguimos los siguientes pasos.

1. Crear un fichero tipo .pas (fichero fuente de pascal), incluir la unidad de objetos Windows Wobjects, WinTypes y WinProcs. Añadir además la cabecera {Y$ WDAQ_TP.INC} que es un módulo de definiciones de las funciones NI-DAQ.
2. Crear un fichero de recursos usando el Borland Whitewater Resource Toolkit y salvarlo en un fichero .res. Hay que añadir este fichero al fichero ejecutable usando la directiva del compilador {$R...}
3. Poner la directiva del compilador {N$+} para activar los puntos de coma flotante extendidos

A parte de los paso anteriores tenemos que tener en cuenta alguna aspectos del funcionamiento del Pascal en el manejo de punteros y cadenas.

El Pascal no admite punteros de tipo *huge*. Por lo tanto debemos crear nuestra propia aritmética de punteros cuando accedemos a buffers de memoria mayores de 64 KBytes.

Otro aspecto a tener en cuenta en el uso de las cadenas que difiere en Pascal de Windows. Una cadena en Pascal estándar consiste de un array de hasta 255 caracteres, con el primer byte reservado para la longitud de la cadena. Sin embargo, Windows y las funciones NI-DAQ esperan una cadena terminada en un nulo, tal como lo hace el lenguaje C. Para salvar este problema, la extensiones de Turbo Pascal para Windows soportan la terminación de un nulo. Para habilitar esa opción debemos incluir la directiva {$X+} en el programa.

4.5.9 Método Para Realizar Una Aplicación Usando Las Funciones NI-DAQ Con Microsoft Visual Basic En Entorno Windows.

Para crear un programa en Visual BASIC, crear las formas y el código como un programa normal, llamado las funciones NI-DAQ como funciones normales. La definición de las funciones prototipo se incluye añadiendo el módulo cabecera WDAQ_VB.BAS como un nuevo modulo el fichero proyecto (File/Add File...).

Como nota final a los modos de programación usando las NI-DAQ debemos decir que estas se pueden combinar con las NI-DSP de forma que unas las primeras se encargarían de el proceso de adquisición y las segundas de la parte de procesado de los datos.

4.5 PROGRAMACIÓN CON TOOLSKIT.

Aparte de los métodos de programación señalados antes hay otro que consiste en programar directamente la AT-DSP2200 accediendo a todos sus registros y al WEDSP32C. Esta forma es sin duda la mejor de todas las formas de programación, aunque también la más compleja, en un programa podemos combinar código en C, en ensamblador y funciones panel de LabWindows. Además podemos construirnos nuestras propias rutinas al estilo de las NI-DSP para realizar acciones concretas.

En definitiva y a modo de resumen podemos decir que programar usando las rutinas NI-DSP o NI-DAQ es una forma más cómoda pero limita el rendimiento de la AT-DSP2200. En cambio combinado las rutinas anteriores con la programación en ensamblador es la única forma de obtener un rendimiento óptimo de las ventajas de la AT-DSP2200 y en especial del procesador digital de señal.

5. EL SOFTWARE DESARROLLADO PARA LA AT-DSP2200

5.1 JUSTIFICACIÓN.

La AT-DSP2200 se suministra sin ningún tipo de programa para usuarios finales, como ya hemos dicho en capítulos anteriores se acompaña con dos tipos de librerías, las NI-DSP y las NI-DAQ, estas librerías sirven de base para construir aplicaciones y las he usado para realizar mi programa. Como también se ha comentado las aplicaciones de la AT-DSP2200 están limitadas debido a que no disponemos de la principal herramienta de programación que nos permitiría acceder a la AT-DSP2200 de una forma completa. Por tanto lo que he hecho ha sido prescindir del la capacidad de trabajar en tiempo real y usando las librerías he tratado de construir una aplicación que permita las acciones básicas en el procesado de audio digital, adquisición, generación , edición y filtrado.

La elección del lenguaje para realizar el programa, Micrososft C, ha estado motivada por las restricciones impuestas por las NI-DSP. Así como las NI-DAQ vienen preparadas para ser compiladas con una gran cantidad de lenguajes, las librerías NI-DSP sólo vienen para QuickBASIC, LabWindows y Microsoft C. En el momento de comenzar el trabajo yo no conocía ninguno de los lenguajes anteriores, y a la hora de aprender uno de ellos me decidí por el C porque me pareció el más potente de los tres. El LabWindows que es una herramienta de programación para

realizar entornos gráficos y cuyo código se escribe en C, también era una buena elección pero la versión de la que se disponía no era compatible con las librerías. Como resultado, sólo quedaba optar por el Micrososft C.

5.2 ESTRUCTURA DEL PROGRAMA

El programa, que ha sido realizado de forma totalmente modular. Partiendo de una estructura común como el menú principal que aparece al cargar el programa se han agregado unas aplicaciones. El programa principal apenas ocupa algunas líneas de código para controlar la elección de una acción mediante el menú, luego se le pasa el control al procedimiento que se haya llamado, se ejecuta este, y al finalizar devuelve el control al programa principal. La idea ha sido crear un programa que se pueda adaptar a otras aplicaciones futuras, si por ejemplo en el futuro se crea una utilidad para el cálculo de la FFT para incorporarla al menú principal basta con añadírsela al fichero "menu.h" y asignarle un código. un ejemplo de esto se muestra en las siguientes líneas.

```
#define TRUE 1
#define FALSE 0
/*--------------------definicion de tipos--------------------------*/
typedef char cadena[15];
typedef unsigned char BYTE;
typedef unsigned int  WORD;
typedef unsigned char BOOLEAN;
void menu(struct REGSMENU);
```

```
struct OPCIONES
{
  cadena        Opcion;
  BYTE          CodOpcion;
  BOOLEAN   OpcionActiva;
};
struct REGSMENU
{
  WORD   XInin,XFinal;
  cadena     Titulo;
  WORD    TamanoMenu;
  char        *ptr;
  BOOLEAN     Activo;
  BYTE        Elecs;
  struct OPCIONES ListaOpciones[5];
 };
struct REGSMENU DesMenu[4]={18,
                    80,
                    "Fichero",
                    11,
                   NULL,
                    TRUE,
                    3,
                    {"Cargar",21,TRUE,
```

```
"Borrar",23,TRUE,
"Salir",24,TRUE,
"Ir al DOS",22,TRUE},
114,  170,
"Editar",
11,
NULL,
FALSE,
4,
{"Reproducir",31,TRUE,
"Borrar",32,TRUE,
"Copiar",33,TRUE,
"Zoom",34,TRUE},
206,
295,
"E/S",
11,
NULL,
TRUE,
3,
{"Adquirir",42,TRUE,
"Genera Rapido",41,TRUE,
"Generar",43,TRUE},
308,
```

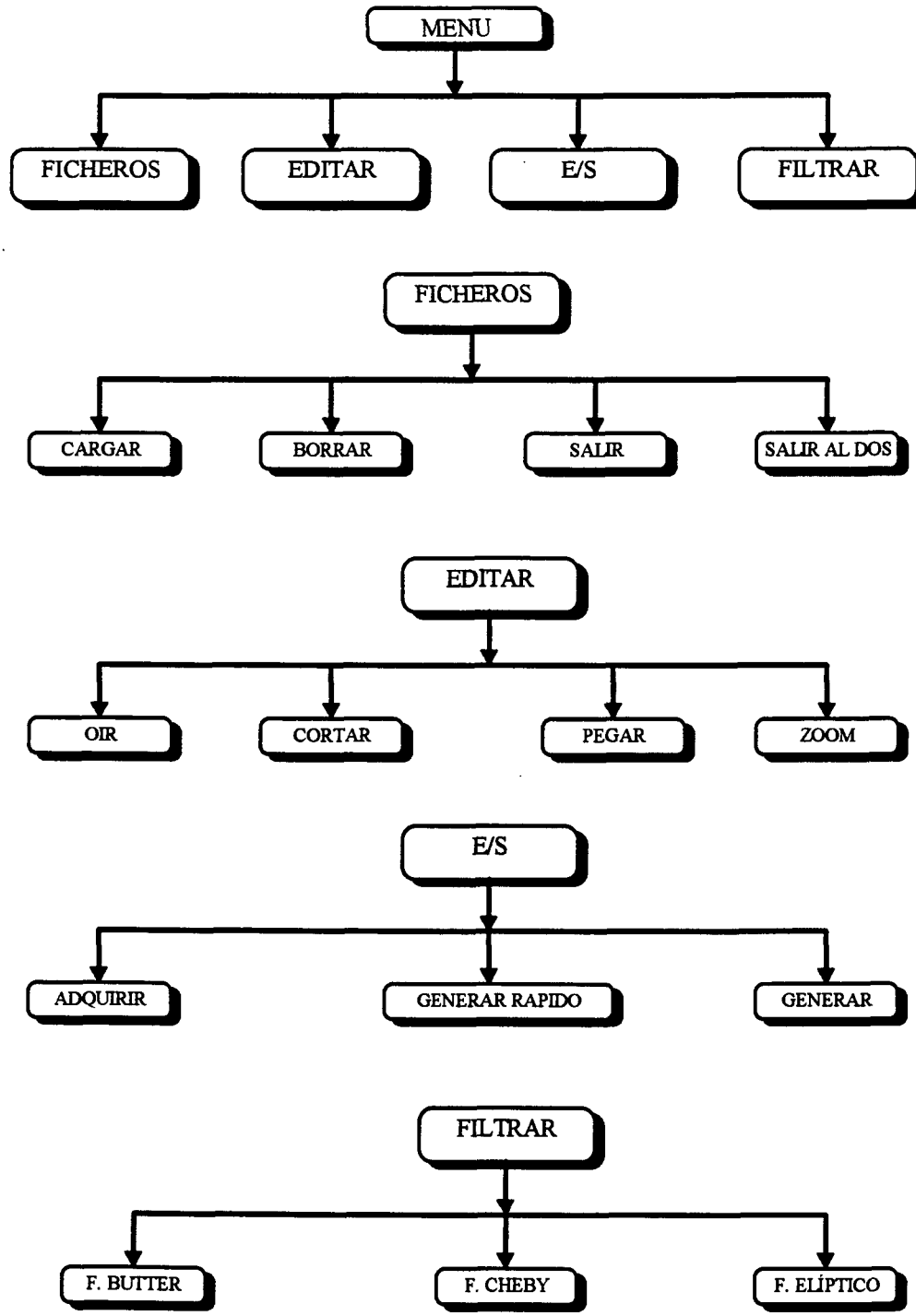
```
381,
"Procesar",
11,
NULL, TRUE,3,
 {"Butterworth",51,TRUE,
"Chebyshev",52,TRUE,
"Eliptico",53,TRUE
 "FFT",54,TRUE}};  ⟶ Hemos añadido la nueva
                     función al menú.
```

Como podemos ver resulta muy fácil, la nueva aplicación ya tendrá un punto de entrada desde el menú principal que permitirá entrar en ella.

EL motivo de haber hecho el programa en modo gráfico ha venido impuesto por la necesidad de representar las formas de onda. El entorno gráfico, aunque más complejo de programar, resulta más atractivo para el usuario.

5.3 DESCRIPCIÓN DE LOS PROCEDIMIENTOS Y DIAGRAMAS DE FLUJO.

Como ya hemos comentado el programa está estructurado en módulos que son llamados desde el menú principal, en el siguiente diagrama se puede ver esta estructuración.



En los apartados siguientes describiremos cada uno de estos procedimientos con más detalle.

5.3.1 Menú Principal

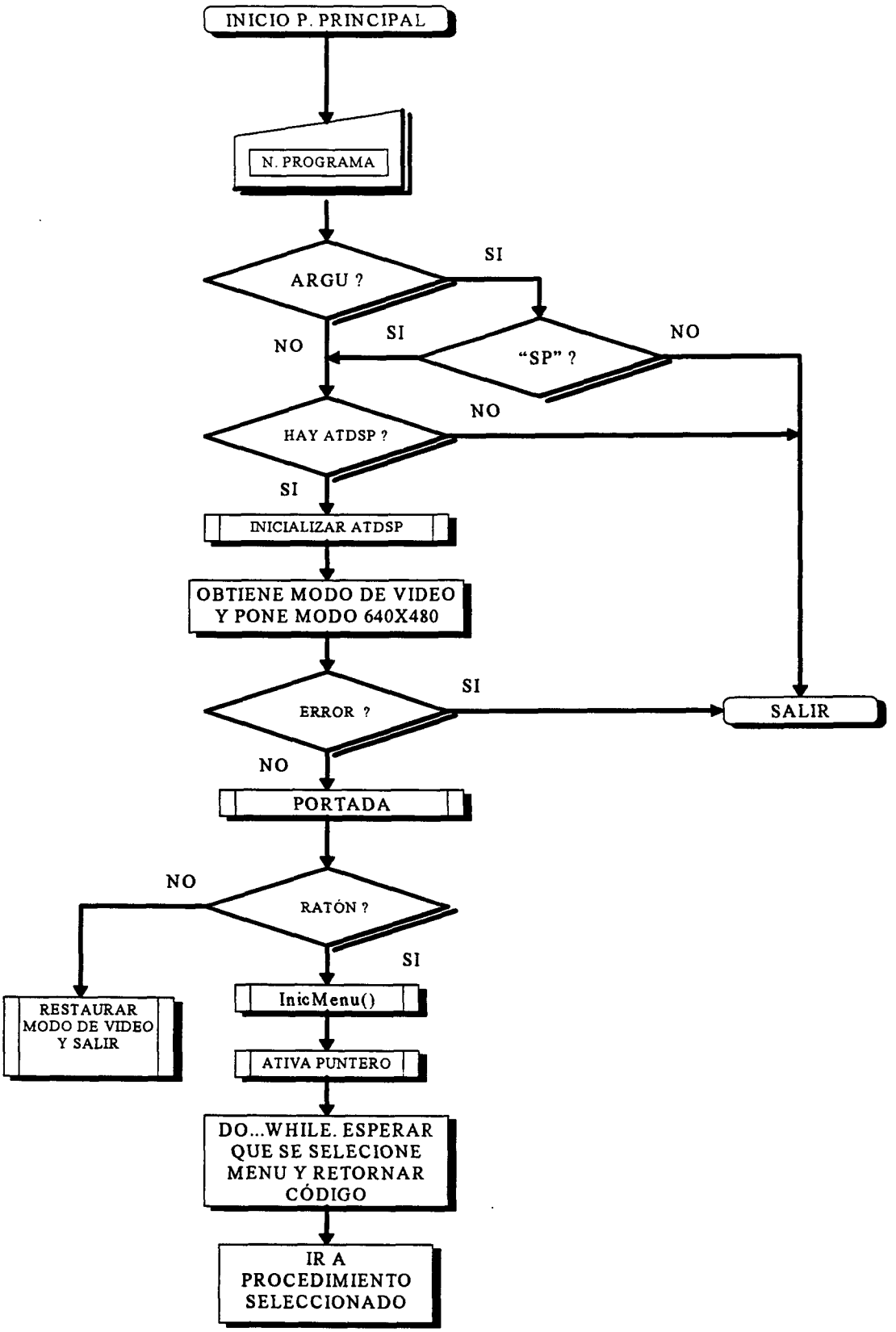
El procedimiento principal consta de un bucle "do...while" que se encarga de gestionar todo el sistema de menús. Cuando tecleamos el nombre del programa se

llama siempre al procedimiento principal. Este procedimiento analiza los argumentos de la línea de comandos del nombre del programa. Si la llamada se hace sin argumentos primero comprueba si hay si la AT-DSP2200 está conectada, si no está se sale del programa y se indica un mensaje de error, luego se inicializa la AT-DSP2200, se obtiene el modo de vídeo activo se salva y luego se cambia al modo de vídeo más alto, normalmente 640x480, llama al procedimiento que dibuja la pantalla de presentación, pone la pantalla de color azul, comprueba si hay ratón instalado, si no hay ratón se restaura el modo de vídeo y se abandona el programa, llama al procedimiento InicMenu() que se encarga de poner el menú en la parte alta de la pantalla en función de lo que haya en el fichero "menup.h" y por último activa el puntero de ratón. Luego entra en el bucle y espera a que el puntero de ratón se sitúe sobre uno de los apartados del menú y se pulse la tecla izquierda del ratón. Para saber cuando se produce este evento el bucle "do...while" está leyendo constantemente la posición del ratón en la pantalla y el estado de los botones. La función "leeraton()" se encarga de esto devolviendo el estado del ratón cada vez que se llama. Cuando se ha detectado que se ha seleccionado una opción del menú principal llamamos al procedimiento Menu(), que es el encargado de desplegar el menú y gestionar el desplazamiento de la barra, cuando seleccionamos una de las opciones pulsando la tecla izquierda del ratón se sale de este procedimiento y este retorna un código correspondiente a la opción seleccionada. Este código es analizado por la estructura "case" que está dentro del bucle "do... while" y se encarga de llamar a la rutina que se haya seleccionado. En el caso que no queramos ver la pantalla de presentación cada vez que entremos en el programa cuando lo ejecutemos debemos poner el nombre del programa seguido del argumento "sp", sin presentación, en este caso cuando se

analice la línea de comandos y se compruebe que el argumento es "sp" el programa se ejecuta de la misma forma que en el paso anterior sólo que no aparece la pantalla de presentación. En caso de que el argumento sea distinto de "sp", el programa se termina y se indica que el argumento correcto es "sp".

En este procedimiento como hemos visto interviene varios procedimientos, los procedimientos "leeratón()", "punteroon()", "punteroff()", son comunes al todas las rutinas del programa, en cuanto al la rutina de selección de menú su funcionamiento es como sigue.

Cuando selecciones una opción antes de desplegar calcula el tamaño de la ventana a desplegar, para ello lee la lista de opciones y calcula la longitud de la cadena más larga que va ser desplegada, la altura la calcula sabiendo el número de opciones que hay en cada menú desplegable. Luego salva lo que hay debajo de la pantalla en una variable de tipo puntero, crea la barra en memoria, dibuja la caja del menú, la rellena con las variables y dibuja la barra en el primer nombre. Luego entra en un bucle que comprueba la posición del ratón y si este se desplaza hacia abajo dentro de la caja del menú dibuja la barra en otra posición. Para salir basta con pulsar la tecla izquierda del ratón o la tecla ESC para salir sin seleccionar nada. Se borra la caja del menú restaurando la pantalla. El diagrama de flujo que se muestra a continuación refleja este proceso.



5.3.2 Función de carga de ficheros

Este procediemiento es el que nos permite seleccionar un fichero de los existente y mostralo en pantalla. Está dividido en dos partes la primera es la que se encarga de mostrarnos la caratula del carga, aqui podemos elegir un fichero o cambiar de directorio. El segundo procedimiento asociado nos permite abrir ese fichero, comprobar si es compatible y dibujarlo en pantalla mostrando sus características principales.

Al realizar el primer procedimiento se me planteaba un problema, no era posible saber a priori el número de ficheros que podía haber en en directorio activo, de forma que no era posible asignar a priori la memoria para guardar los nombres de los ficheros. El problema se resolvió haciendo uso de la memoria dinámica con las funciones sumistradas por el C para tratarla, estas funciones son entre otras **malloc()** y **realloc()**.

El funcionamiento del procedimiento es como sige. Se crea una tabla de punteros a punteros que se va reasignando dinamicamente a medida que vamos encontrando ficheros.

Al entrar en este procedimiento se dibuja la caratula , se obtine el directorectorio activo y se crea una tabla de punteros a punteros para guardar 10 ficheros. Se comienza la busqueda de los primeros 10 ficheros del directorio, si se encuentran más ficheros mediante la función realloc se reasigna el tamaño de la tabla

para que sea capaz de guardar otros 10 ficheros más, si todavia quedan más ficheros se vuelve a reasignar y así de forma sucesiva hasta que se hayan leido todas los ficheros.

Con los ficheros leidos y guardados en un buufer de memoria lo que hacemos ahora es presentarlos en pantalla mediante un "scrooll" de 10 ficheros. Tomamos los diez primeros ficheros del buffer comprobamos sus atributos y procedemos de la siguiente forma. Si en un directorio lo escrbimos en rojo si es un fichero lo escribimos en negro. Escribimos sólo los 10 primeros ficheros del buffer y el procedimiento se queda a la espera de ordenes. Si se detecta la pulsación de las teclas, Página arriba o Página Abajo y es posible lee, otra página de ficheros desde el buffer de memoria (una págian son 10 ficheros). Si detecta el puntero de raton en el recuadro llamado fichero espera hasta que se introduzca un nombre de fichero o un especificador de búsqueda (por ejemplo *.c) si se introduce el nombre de un fichero que no existe se borra el recuadro y se espera a que se introduzca un nombre correcto. Si introducimos un nombre de fichero correcto y este existe nos lo muestra en el recuador inferior y se queda a la espera de recibir más ordenes. Si la siguiente orden es pulsar ENTER el fichero se carga. En realida la carga no se produce de forma inmediata, antes se comprueba que el fichero no este vació. Antes de llamar a la rutina que se encarga de dibujar el fichero se restablece el directorio activo, si se ha cambiado, se libera la tabla de punteros a punteros. Tambien podemos cambiar en directorio de trabajo y buscar por otros directorios, para ello hay que situar el punntero de raton sobre el cuadro que mueste el directorio de trabajo y pulsar una tecla del ratón. Escibimos el nombre del directorio al que queramos cambiarnos. Si vamos a retroceder en el arbol hay que

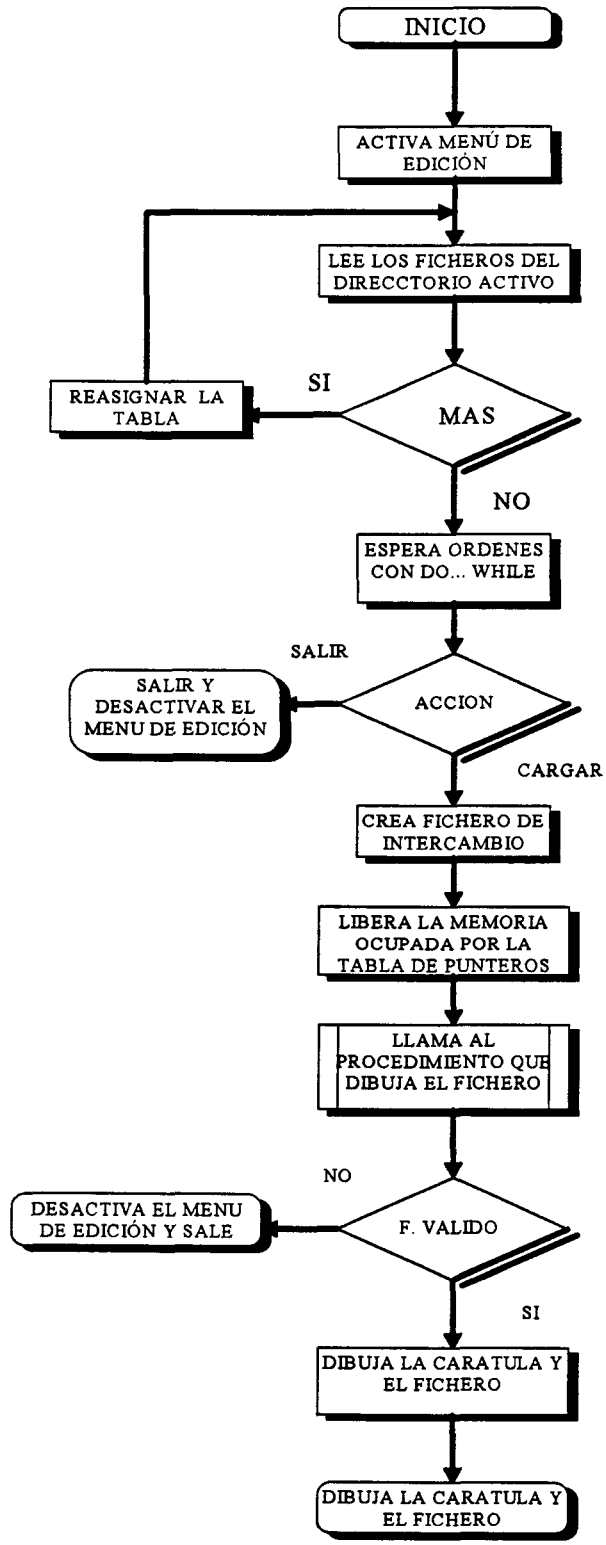
escribir ".." como en DOS. Para cambiar de directorio se debe escribir siempre sólo el nombre del directorio.

Por ultimo antes de salir cuando hemos seleccionado un fichero se crea el llamado fichero de intercambio, este fichero es el que guada el nombre del fichero que vamos a cargar y que permite que sea accesible por otros procedimientos que antes de ejecutarse comprueaban la existencia de dicho fichero.

Despues de esto llamamos a la función **dibuja()** esta función abre el fichero de intercambio y lee el nombre del fichero, comprueba que este es un fichero válido leyendo su cabecera. En el caso de que no sea un fichero compatible lo indica mediante un mensaje y se retorna al menu principal. Si el fichero es valido extrae la información de la cabecera y lo muestra en pantalla. La presentación del fichero se hace leyendo los datos del fichero y escribiendo pixels en pantalla, este procedimeinto es más lento que la cargar el fichero directamente a memoria pero por otro lado nos permite trabajar con ficheros más grandes.

Antes de llamar a la rutina que se encarga de cargar un fichero la parte del menú denominada "editar" está inactiva en el momento de cargar un fichero antes de llamar a la primera rutina lo que se hace es activar este menu para que cuando el fichero se haya cargado ya se encuentre activado. Cuando borramos el fichero de la pantalla volvemos a desactivar este menu, con esto evitamos que se pueda producir un error por hacer uso de este menu cuando no hay un fichero seleccionado.

A continuación vemos el diagrama de flujo de las rutinas de carga.

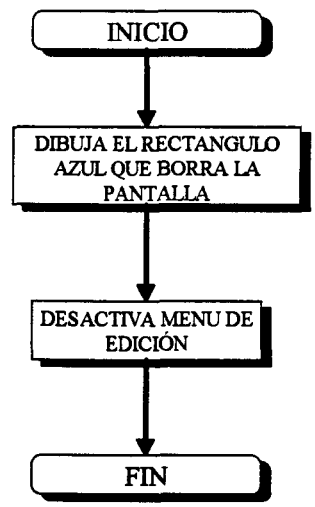


5.3.3 Función para Borrar.

Este procedimiento es el que se encarga de borrar la pantalla. Lo que hace la función es dibujar un rectángulo azul con lo que se borra lo que esta debajo. Esta

función además borra el fichero de intercambio si existe y desactiva el menú del edición.

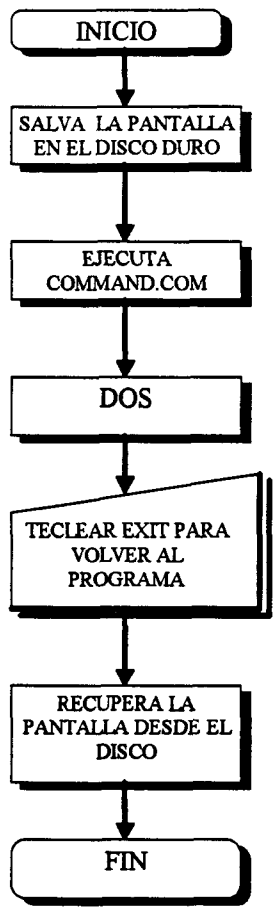
En el siguiente diagrama de flujo se muestra el diagrama de flujo.



5.3.4 Función que permite salir al dos momentaneamente.

Esta función nos permite salir al sistema operativo MS-DOS de forma momentanea. Cuando llamamos esta función lo primero que se hace es salvar toda la pantalla en el disco para luego poder restaurarla al volver, despues se cambia el modo de video, pasando al modo texto. Para salir al dos mediante el comando **system** de C ejeutamos otro COMMAND.COM. Esto hace que en este momento haya dos procesadores de comandos abiertos. Para volver al programa lo que hacemos es teclear EXIT que es una instrucción del DOS para salir del COMMAND.COM. Cuando salimos de este procesador se recupera la pantalla que habiamos salvado en el disco duro y se retorna al menu.

En el siguiente diagrama de flujo podemos ver la forma de hacer lo que acabamos de comentar.



5.3.5 Función que permite oír un trozo de fichero que haya sido marcado.

Este procedimiento pertenece al grupo de procedimientos de edición. Su prototipo es el siguiente.

void reproducir(unsigned long PunteroInicial, unsigned long PunteroFinal, int origen);

Este procedimiento puede ser llamado directamente abriendo el menú de edición o también puede ser llamado desde la función "ZOOM" para reproducir un trozo que hayamos ampliado. Para poder distinguir el origen de la llamada en la declaración del procedimiento hay una variable que indica el origen, si la variable **origen** es cero, se llama del procedimiento principal y por tanto los valores de los punteros que indican donde comenzar la reproducción y donde parar son ignorados, se llama a la función marcar para obtener el valor de estos punteros. Si la variable **origen** uno, indica que la llamada se hace desde la función "ZOOM", y en este caso se consideran los valores de los punteros que indican el comienzo y el final a reproducir y no se llama a la función marcar. Luego abre el fichero de intercambio para buscar el nombre del fichero activo. Abre este fichero y lee su cabecera para saber la frecuencia a la que debe ser generado y si son uno o dos canales. Se calcula la posición donde debemos situar los punteros dentro del fichero en función de la zona de pantalla donde hemos marcado. Pregunta por el número de veces que queremos oír el trozo seleccionado. Con toda la información llama a la función WFM_From_Disk() de la librería NI-DAQ que se encarga de generar el fichero, los parámetros de esta función se explican más adelante. Si no ocurre ningún error después de generar el fichero se llama a la función Init_DA-Brds() para inicializar la AT-DSP2200 a los valores por defecto. La función WFM_From_Disk() crea un doble buffer en la memoria RAM del PC y programa la DMA para realizar las trasferencias del fichero hasta los canales de salida de la AT-DSP2200. Cuando ha finalizado la generación de un fichero borra los buffers de memoria y retorna el control al procedimiento que la llamó. Por último se borra la barra que nos permite ver lo que se ha marcado, esto se hace dibujando la misma barra pero con los colores de fondo.

La función WFM_from_Disk se declara de la siguiente forma:

Estado = **WFM_from_Disk** (placa, nu_canales, VectCanal, NomFichero, PunteroInicial, PunteroFinal, Iteraciones, frecuencia);

placa: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

nu-canales: dato de tipo **integer** e indica el número de canales que van a ser generados. Este dato se extrae de la cabecera.

VectCanal: array de tipo **integer** que indica por el canal de la tarjeta que se generará el fichero, si es un fichero mono VectCanal[0] = 0 y si es esterero VectCanal[0] = 0 VectCanal[1] = 1.

NomFichero: dato de tipo **string** que indica el nombre del fichero.

PunteroInicio: dato de tipo **unsigned long** que indica el punto de comienzo del trozo de fichero que hemos seleccionado.

PunteroFinal: dato de tipo **unsigned long** que indica el punto donde termina el trozo de fichero que hemos seleccionado.

Iteraciones: dato de tipo **unsigned long** que indica el número de veces que vamos a repetir el trozo seleccionado. Un cero indica que el fichero se generará hasta el final

frecuencia: dato de tipo **double** que indica la frecuencia a la que se generará las muestras. Este dato se extrae de la cabecera.

Si el fichero se ha generado correctamente **Estado** es igual a cero en caso contrario retorna el código de error devuelto por la función **ErrPrint**.

La función **Init_DA_Brds** inicializa la AT-DSP2200 , esta es una función indocumentada, tiene como parámetro de entrada el número de slot donde se conecta la tarjeta y devuelve el código del tipo de placa que tiene conectada, que en es este caso debe ser 18. La función se declara de la forma:

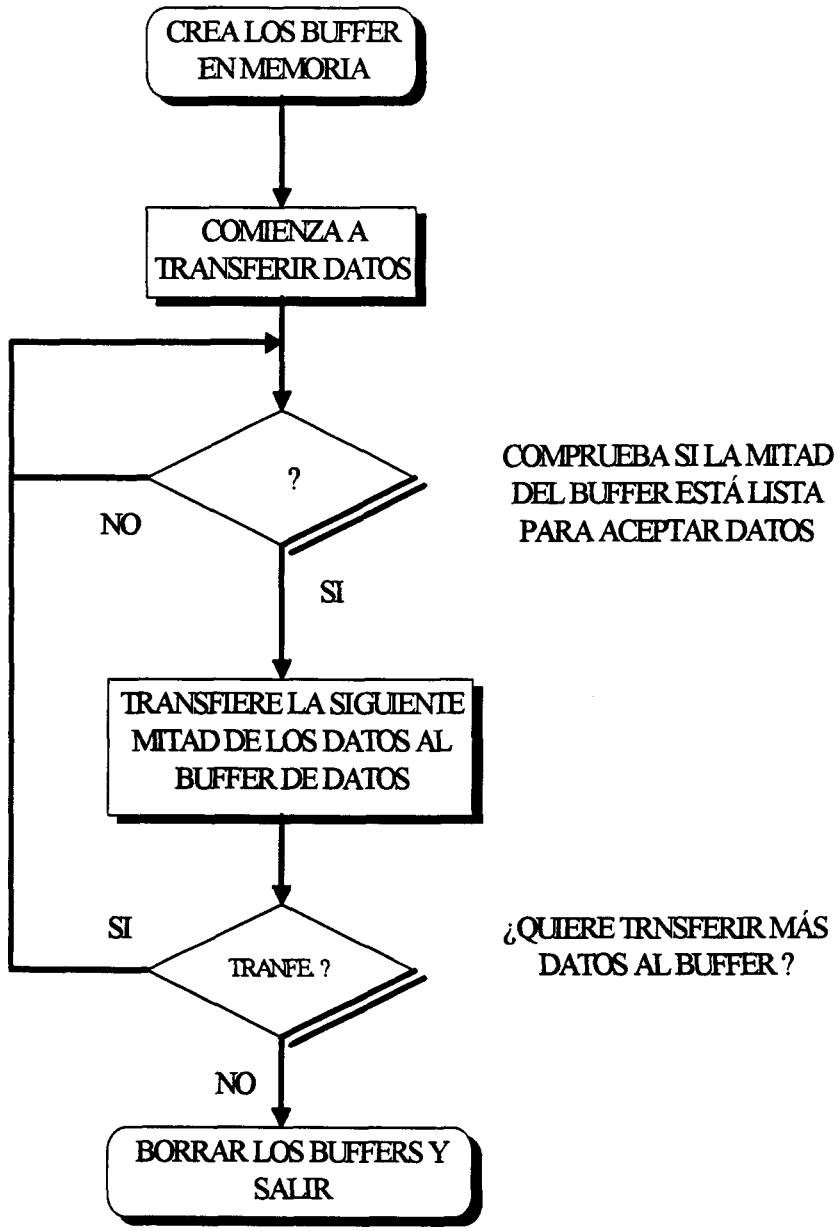
estado = **Init_DA_Brds**(slot, &CodigoTarjeta);

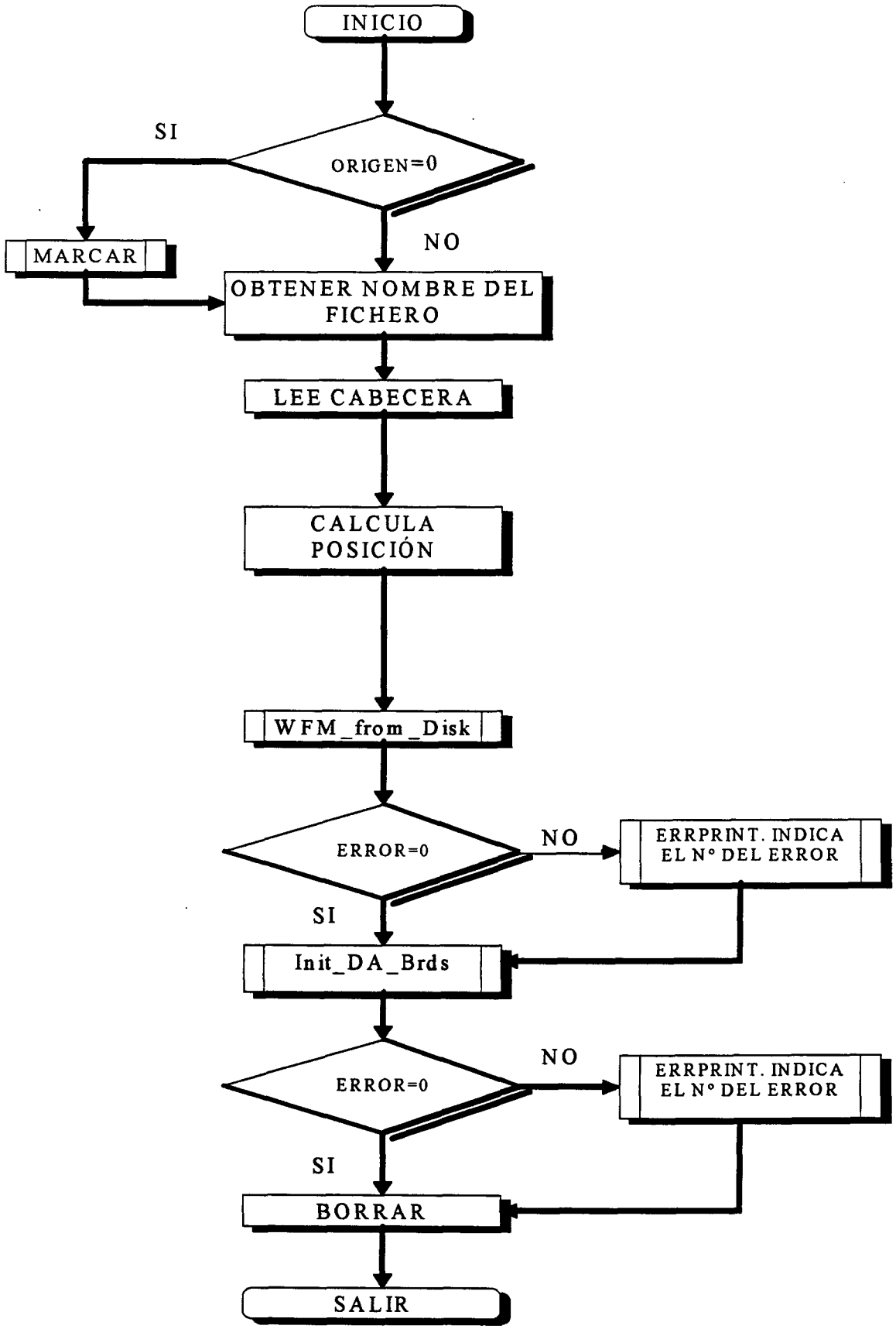
slot: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

CodigoTarjeta: dato de tipo integer que indica el tipo de tarjeta que esta conectada, el código 18 corresponde a la AT-DSP2200.

estado: si la función se ha ejecutado correctamente estado es igual cero de lo contrario es distinto de cero y la función **ErrPrint** nos indica el código del error.

A continuación podemos ver el diagrama de flujo del procedimiento WFM_from_Disk y luego un diagrama del procedimiento completo.

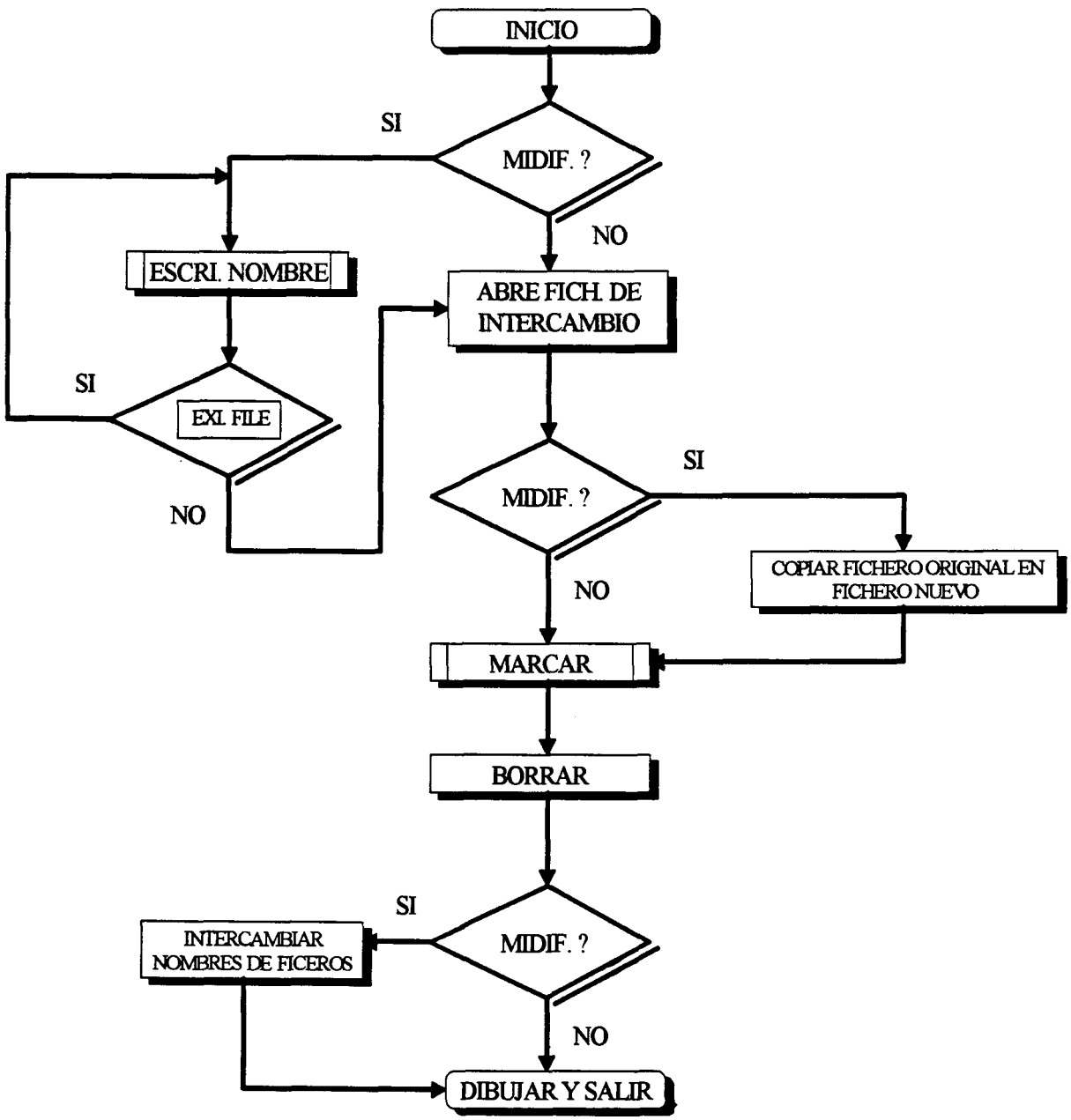




5.3.6 Función para borrar

Este procedimiento pertenece al grupo de edición. Cuando se llama a este procedimiento lo primero que se hace es preguntar al usuario si el fichero original se va a modificar o si por el contrario los cambios hechos afectaran al fichero original. De esto se encarga la rutina **ModiFile()**. Esta rutina se encarga de presentar una pequeña pantalla con dos botones y la pregunta "¿Quiere guardar los cambios en otro fichero?". Si contestamos positivamente se devuelve el valor uno y si lo hacemos de forma negativa el valor cero. Luego abrimos el fichero de intercambio para ver el nombre del fichero activo. Si hemos contestado afirmativamente a la pregunta de modificar el fichero llamamos la rutina **NuevoFile()** que se encarga de abrir un menú que nos permite escribir el nombre del nuevo fichero, esto se hace llamando a la función **escribe()** que nos permite escribir en modo gráfico. Después de escribir el nombre del fichero comprobamos si ya existe con la función **NuevoFile()**, en el caso de que ya exista nos lo indica y nos pide que introduzcamos otro nombre. Si hemos contestado que queremos guardar los cambios en otro fichero, el fichero original se copia en el nuevo fichero y en la pantalla aparece un mensaje diciendo que esperemos a que se realicen los cambios. En este punto se llama al procedimiento **marcar()** que nos permite seleccionar el trozo de fichero a cortar. Una vez seleccionado un trozo de fichero se crea un fichero temporal y se copia en el fichero a cortar desde el comienzo hasta el primer punto que hemos marcado, luego movemos el puntero hasta el segundo punto y copiamos desde aquí hasta el final del fichero ahora borramos el fichero principal y copiamos el temporal sobre el primer fichero. En este punto si hemos elegido salvar los cambios tenemos dos ficheros el original modificado y el que

hemos indicado para guardar los cambios, basta pues con intercambiar los nombres. Por último llamamos a la función **dibuja()** que se encarga de mostrar los cambios en la pantalla dibujando el fichero. A continuación se muestra un diagrama de flujo del procedimiento **borrar()**.

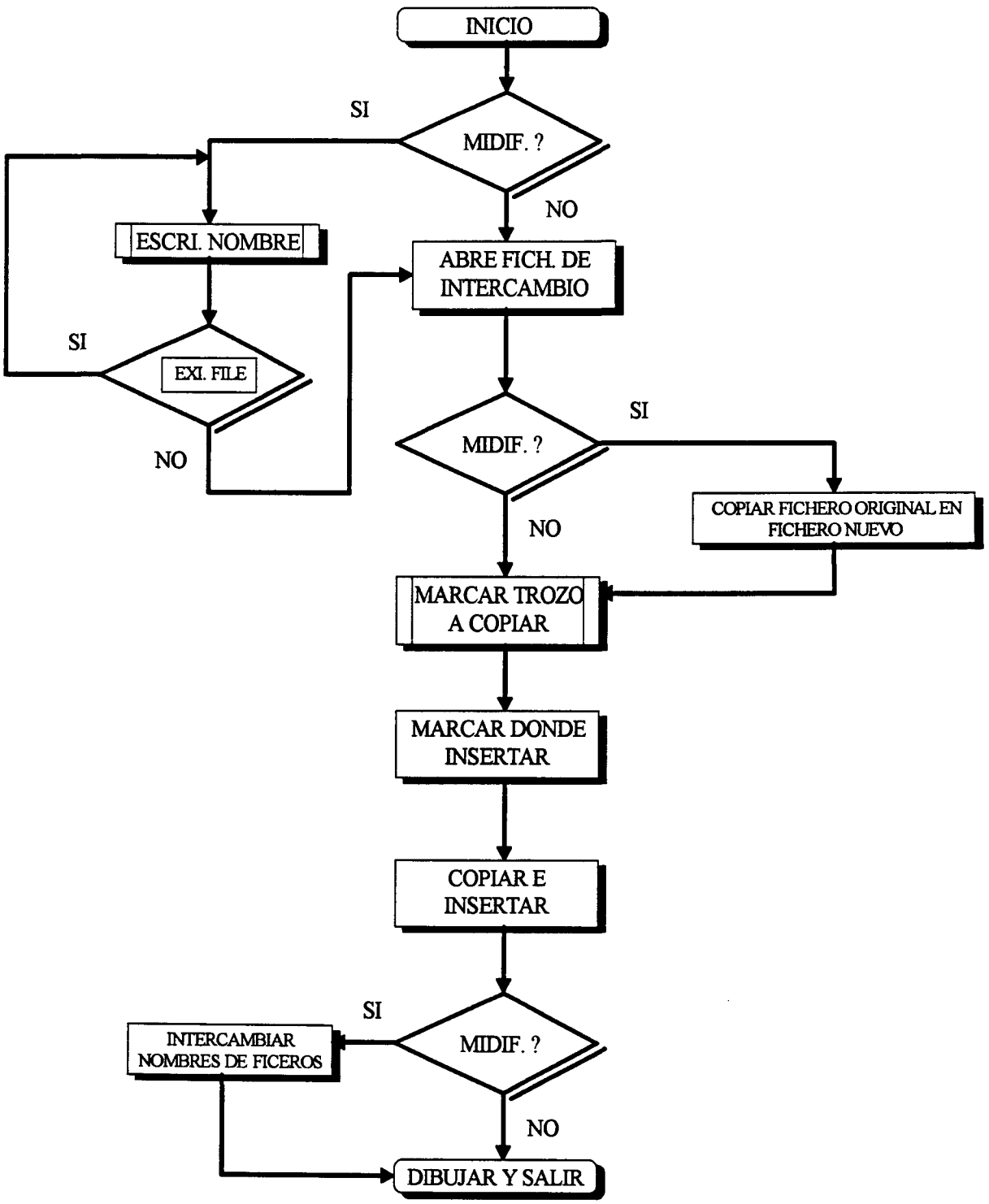


5.3.7 Función para copiar.

Este procedimiento pertenece al grupo de edición. Su funcionamiento es similar en cuanto a la primera parte, pregunta si se va a modificar el fichero original comprueba que el fichero no exista, llama a marcar. Ambas funciones comparten los procedimientos **ModiFile()**, **dibuja()**, **NomFile()**, **NuevoFile()** y **marcar()**. Se no indica que marquemos el trozo a copiar y luego el lugar donde queremos insertar ese trozo. El procedimiento de copiado e inserción se realiza con ayuda dos ficheros temporales y es de la forma siguiente; copiamos el trozo seleccionado en un fichero temporal, luego copiamos el fichero original hasta el punto de inserción en otro fichero temporal, copiamos el trozo a insertar en ese fichero a partir de ese punto y por último copiamos la última parte del fichero original en el fichero temporal. Llegados a este punto tenemos un fichero original con la información exacta y en el fichero temporal los datos originales con el trozo que habíamos seleccionado copiado en el lugar que queríamos.

Luego en función de lo que hubiéramos indicado al principio; modificar el fichero original o no modificarlo realizamos lo siguiente: si habíamos indicado guardar los datos si modificar renombrados el fichero temporal con el nombre que le habíamos dado al principio al fichero. Si por el contrario habíamos decidido modificar el fichero original borramos el primer fichero y renombramos el temporal con el nombre del primer fichero. Luego llamamos a la función dibujar y salimos del procedimiento.

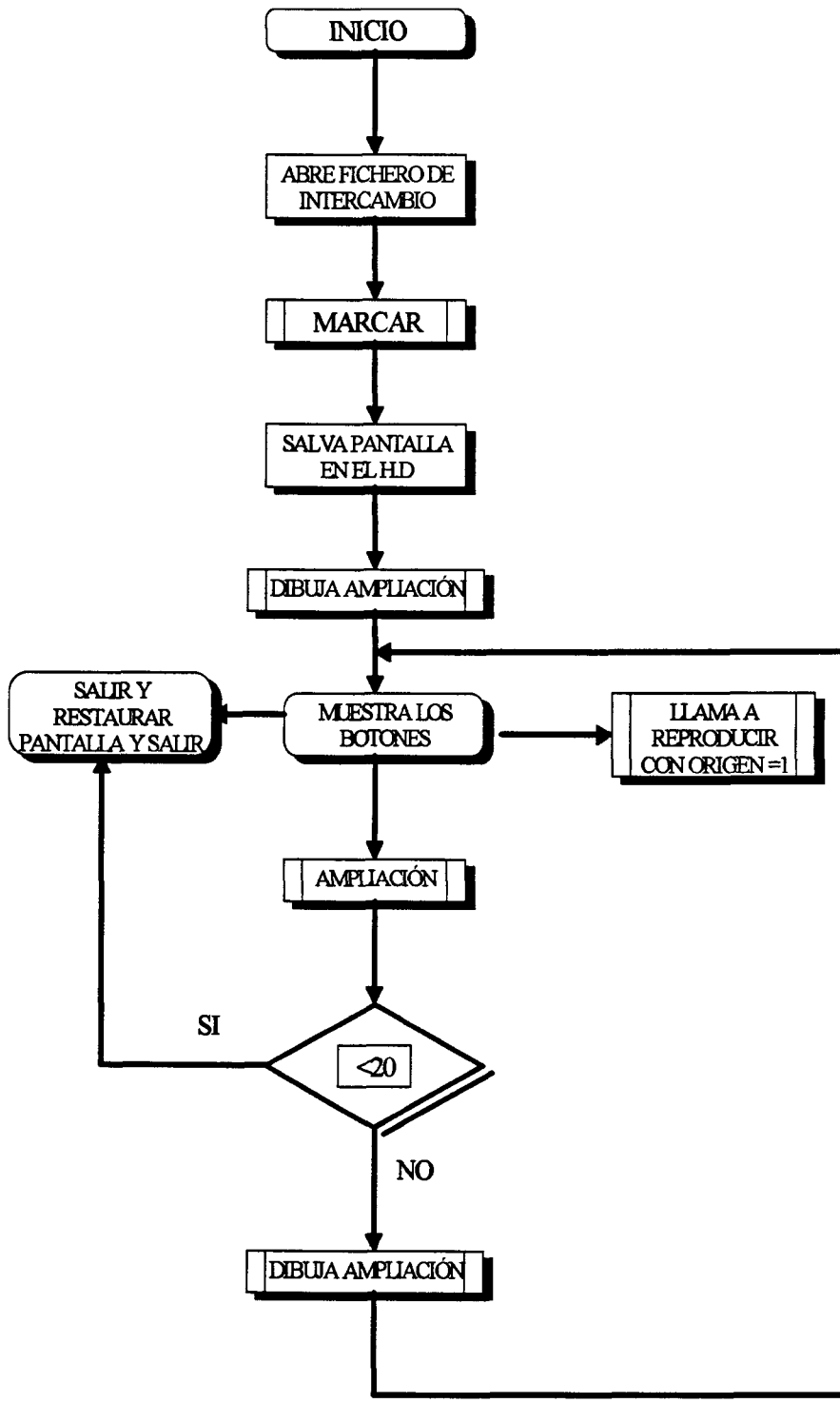
El diagrama de flujo de flujo de este procedimiento es el que se muestra a continuación.



5.3.8 Función de zoom.

Este procedimiento pertenece al grupo de edición. Permite seleccionar un trozo de fichero y verlo en detalle. Cuando llamamos a este procedimiento lo primero que hacemos es abrir el fichero de intercambio para conocer el nombre del fichero activo, luego se presenta un mensaje indicándonos que marquemos la zona a ampliar, cuando hayamos hecho esto y antes de presentar la imagen ampliada salvamos el dibujo que representa la forma de onda del fichero en un archivo temporal en disco. La razón de salvar la pantalla a disco y no a un buffer de memoria es que el tamaño que ocupa es imagen en memoria es muy grande y considerando el tamaño del programa podaríamos tener problemas de memoria. Con el la imagen de pantalla salvada en el disco borramos esta de la pantalla y dibujamos la ampliación. Cuando hemos dibujado la ampliación ponemos tres botones en la pantalla para que seleccionemos una de las tres opciones disponibles, "SALIR", "OÍR", "ZOOM". La primera opción nos permite salir y se restaura la imagen original que teníamos en el disco. La segunda opción llama al procedimiento **reproducir()** con la variable origen a uno para indicar que la llamada se hace desde este procedimiento y no es necesario marcar otra vez el trozo a oír. Cuando el fichero ha sido reproducido se retorna a este procedimiento y podemos volver a seleccionar una de las opciones. Por último la tercera opción nos permite continuar ampliando más. Cada vez que hacemos una ampliación se comprueba que entre los dos punteros hay mas de 20 muestras para ampliar de lo contrario se nos indica mediante un mensaje que ya no se puede ampliar más el fichero. En este caso también retorna a la pantalla principal restaurando esta

desde el disco duro. El siguiente diagrama de flujo indica el de forma gráfica como se hecho este procedimiento.



5.3.9 Función para adquirir datos.

Este procedimiento está dividido en dos partes claramente diferenciadas, la primera se encarga de dibujar el menú que nos permite seleccionar los parámetros de la adquisición en este menú podemos configurar varios parámetros como número de canales, frecuencia de muestreo tipo de acople, nombre del fichero donde se guardaran los datos, si se usará trigger para la adquisición y si se usa el tipo, analógico digital. Cuando hayamos seleccionados todos estos parámetros llamamos al procedimiento **leer()** que tiene como entrada estos parámetros. El procedimiento es uno de los más importante y se encarga, primero de configurar la tarjeta para la adquisición con los datos que le pasa el procedimiento **adquisición()** y segundo de realizar la transferencia de los datos desde la AT-DSP2200 hasta el disco duro en el tiempo de adquisición para. Para hacer todo este proceso hace uso de varias rutinas pertenecientes a las NI-DAQ y de la memoria extendida. Cuando pulsamos la tecla "GO !" del menú de adquisición primero se configura la tarjeta y luego si hay memoria extendida comienza la adquirir datos y a traspasarlos al disco duro. Para poder hacer esto en tiempo real primero creamos un gran buffer en memoria extendida, un buffer circular de unos 4 Mbytes, la tarjeta mediante el canal de DMA va colocando los datos en este buffer , mientras esto ocurre otra rutina se encarga de irlos extrayendo en porciones de 14336 bytes hasta un buffer intermedio de esta capacidad para luego ser escritos en el disco duro, el truco consiste en ser capaces de ir desalojando el buffer de adquisición con la suficiente rapidez para que cuando este se llene y comience a escribir por el principio no sobreescriba datos que aún no hayamos salvados. Es como una carrera sin fin de forma que tenemos que ser capaces

de aprovechar los momentos en que la DMA no esta ocupada transfiriendo datos para por robo de ciclo escribir en el disco ya que la escritura en este también se hace mediante DMA. Como podemos comprobar el PC queda sobrecargado de trabajo por lo que es fundamental elegir un ordenador con un disco duro rápido y un microprocesador 386 de por lo menos 40 Mhz. Esta es la configuración que yo he utilizado y funciona correctamente, no obstante si se usan procesadores superiores y más rápidos en alguna ocasiones sería necesario variar la cantidad de datos a transferir aumentando de 14336 bytes a 16384 bytes.

Ahora vamos a describir las rutinas NI-DAQ que interviene en el procedimiento **leer().**

errNum=MAI_Coupling (placa, numCanales, acoplo);

Esta función permite seleccionar el acoplamiento de los canales de entrada entre AC o DC.

placa: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

numCanales: dato de tipo **integer**, aquí indicamos el en número de canales de tarjeta. Como estas rutinas están hechas para se compatibles con otras tarjetas de National Instruments es necesario en este caso poner dos, pues la AT-DSP2200 tiene dos canales de E/S.

acoplo: array de tipo **integer**, este array tendrá sólo dos elementos , uno para cada canal, y nos permite seleccionar el acoplo de la forma.

acoplo[0] = 0 Para acoplo AC del canal cero, ACH0.

acoplo[1] = 0 Para acoplo AC del canal uno, ACH1.

acoplo[0] = 1 Para acoplo DC del canal cero, ACH0

acoplo[1] = 1 Para acoplo DC del canal uno, ACH1

Es importante saber que ambos canales deben ser configurados con el mismo acoplo. De lo contrario ocurrirá un error.

errNum: Si la función se ejecuta correctamente devuelve un cero caso contrario un número distinto de cero que es el código de error correspondiente.

errNum = MAI_Setup(placa, numCanales, ListCanales, ListGana, IntervaloMux, baseTiempo, modoMux);

Esta función selecciona los canales analógicos a ser leídos, en otras tarjetas además puede seleccionar la ganancia por canal y otros parámetros.

placa: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

numCanales: dato de tipo **integer**, aquí indicamos el en número de canales de tarjeta. Como estas rutinas están hechas para se compatibles con otras tarjetas de National.

ListCanales: array de tipo **integer**, este array contiene una lista de los canales que serán leídos. El tamaño máximo del array es de dos elementos y podemos indicar que

queremos usar los dos canales o sólo el canal cero (ACH0) o sólo el canal uno (ACH1), la forma de hacerlo es la siguiente.

ListCanales[0] = 0 Usamos el ACH0.

ListCanales[0]= 1 Usamos el ACH1.

ListCanales[0]= 0
Usamos ambos canles ACH0 y ACH1
ListCanales[1] = 1

ListGana: array de tipo **interger**, se indica mediante este array las ganancias de cada canal. En la AT-DSP2200 como no se puede ajustar la ganancia eso lo ponemos cero.

IntervaloMux: Dato de tipo **unsigned int**, en la AT-DSP2200 este valor es ignorado y lo ponemos a cero.

baseTiempo: dato de tipo **integer**, en la AT-DSP2200 est valor no es ignorado y lo ponemos a cero.

modoMux: dato de tipo **integer**, en la AT-DSP2200 est valor no es ignorado y lo ponemos a cero.

errNum: Si la función se ejecuta correctamente devuelve un cero caso contrario un número distinto de cero que es el código de error correspondiente.

ErrNum = MDAQ_ScanRate(placa, intervalo, baseTiempo);

Esta función nos permite seleccionar la frecuencia de adquisición para los canales seleccionados anteriormente. La selección se hace indicando la base de tiempos y un factor de división de esa base de tiempos.

placa: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

intervalo: dato de tipo **unsigned int**, con esto seleccionamos los intervalos de división de las frecuencias bases para obtener distintas frecuencias de muestro. los intervalos válidos de división son 1, 2, 4 y 8.

baseTiempo: dato de tipo **integer**, seleccionamos una de las frecuencias base para la adquisición. Las frecuencia base son la que se muestran en la tabla 1.

Tabla 1. Lista de las frecuencias base

Código	Frecuencia
0	51.2 Khz.
1	48 Khz
2	44.1 Khz
3	32 Khz

Cuando seleccionamos una frecuencia no ponemos la frecuencia en si sino el código.

errNum: Si la función se ejecuta correctamente devuelve un cero caso contrario un número distinto de cero que es el código de error correspondiente.

errNum=MDAQ_Ext_Setup(placa, TamaBuffer, scansOFrames, preTrigScans, postTrigScans, extMemAddr);

Esta función asigna un buffer en memoria extendida para almacenar los datos temporalmente antes de guardarlo en el disco duro, además selecciona el modo de adquisición.

placa: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

TamaBuffer: dato de tipo **long**, indica el tamaño del buffer de adquisición usado durante el proceso de digitalización. El tamaño de este buffer se define en función del tipo de adquisición, si es de tipo tramas (FRAMES), o de tipo muestras (SCANS). En mi caso la adquisición es orientada a tramas y el buffer se declara para guardar 60 tramas. Cada trama contiene 14336 muestras.

scansOFrames: dato de tipo **integer**, con esta variable se indica a la función si la adquisición se orienta a la adquisición de tramas (FRAMES) o muestras (SCANS). En este caso hemos decidido el modo de tramas.

En la adquisición orientada a tramas se adquiere un número determinado de tramas son adquiridas. Cada trama puede contener datos pretrigger y postrigger. Todas las tramas tienen el mismo tamaño y usan el mismo modo de disparo, número de canales y frecuencia de adquisición. En la función MDAQ_Start se especifica el número de tramas a adquirir o también se puede seleccionar adquirir un número de tramas indefinido, en este programa se ha seleccionado este último modo.

La adquisición orientada a muestras (SCANS) es un modo postrigger. Después de que se recibe un disparo se digitaliza un número de muestras determinado o un número indefinido si ha sido especificado así.

preTrigScans: dato de tipo **long**. Es el número de muestras que serán reunidas en una trama antes de un disparo. Los valores de 1 y 2 para preTrigScans no son válidos.

postTrigScans: dato de tipo **long**. Es el número de muestras que serán reunidas en una trama después de un disparo. Los valores de 1 y 2 para postTrigScans no son válidos. Cuando la adquisición está orientada a tramas como en este caso el tamaño de buffer viene dado por preTrigScans + postTrigScans.

extMemAddr: Es la dirección física del puntero a la memoria extendida. Esta memoria comienza el la dirección física 0x00100000 y es la dirección que toma la función por defecto.

El programa antes de ejecutarse verifica que existe memoria extendida, lo que no puede hace es verificar si esta se está usando por otros programas o por un disco RAM, el usuario debe tener esto en cuenta antes de ejecutar el programa.

errNum: Si la función se ejecuta correctamente devuelve un cero caso contrario un número distinto de cero que es el código de error correspondiente.

errNum = MDAQ_Trig_Select (placa, digitalTrig, flanco, analogTrig, rampa, NivelTrig, analogFuente);

Esta función selecciona las fuentes de disparo y configura el trigger analógico y el digital.

placa: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

digitalTrig: dato de tipo **integer**. Esta variable habilita o desabilita el trigger digital tal como se indica a continuación.

digitalTrig = 0 Trigger digital desabilitado.

digitalTrig = 1 Trigger analógico desabilitado.

flanco: dato de tipo **integer**. Esta variable sirve para indicar el flanco de actuación de la señal de disparo digital.

flanco = 0 la señal de disparo es activa por flanco de bajada.

flanco = 1 la señal de disparo es activa por flanco de subida.

analogTrig: dato de tipo de tipo **integer**. Esta variable permite habilitar el circuito de disparo analógico.

analogTrig = 0 Trigger analógico desabilitado.

analogTrig = 1 Trigger analógico habilitado.

rampa: dato de tipo **integer**. Esta variable permite seleccionar la pendiente de la rampa de disparo, positiva o negativa.

rampa = 0 Disparo por rampa negativa.

rampa = 1 Disparo por rampa positiva.

NivelTrig: dato de tipo **integer**. Esta variable especifica el valor de tensión a partir del cual se producirá el disparo. El rango de tensiones es ± 2.828.

analogFuente: dato de tipo **integer**. Selecciona la fuente de disparo analógico de la siguiente forma.

0 = Canal 0

1 = Canal 1

errNum: Si la función se ejecuta correctamente devuelve un cero caso contrario un número distinto de cero que es el código de error correspondiente.

errNum = MDAQ_Start (placa, numtrigg);

Está función es la que se encarga de iniciar la adquisición mediante un disparo cuando pulsamos la tecla "GO !". La variable numtrigg indica el número de disparos y se especifica el valor cero la adquisición no para hasta que no reciba una señal que lo indique.

placa: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

numTrigg: dato de **long.** Esta variable indica la si la adquisición se realiza de forma indefinida o si se adquieren un número de tramas determinadas. En el programa que he realizado para la AT-DSP2200 esta variable es cero.

errNum: Si la función se ejecuta correctamente devuelve un cero caso contrario un número distinto de cero que es el código de error correspondiente.

errNum=MDAQ_Check (placa, fullCheck, acqDone, TramasDone, MuestrasDone);

Esta función indica el estado de la adquisición y si esta se ha terminado.

placa: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

fullCheck: dato de tipo **integer**. Esta variable indica como será la información que nos entregará esta función.

fullCheck = 0 Indica que se retorna información parcial.

fullCheck = 1 Indica que se retorna información completa.

acqDone: dato de tipo **integer**. Esta variable se pone a uno cuando la adquisición ha finalizado. Si se elige adquisición continua, como es el caso, esta variable se mantiene a cero todo el tiempo y hasta que no se para la adquisición, cuando esto ocurre la variable pasa a uno.

TramasDone: dato de tipo **long**. Esta variable indica el número de la última trama adquirida.

MuestrasDone: dato de tipo **long**. Esta variable indica el número de la última muestra adquirida. En mi programa como la adquisición es orientada a TRAMAS esta variable no se usa.

errNum: Si la función se ejecuta correctamente devuelve un cero caso contrario un número distinto de cero que es el código de error correspondiente.

errNum = MDAQ_Get (placa, scansOFrames, getOTap, numToGet, startFrame, starScan, timeout, getbuffe, numGotten, lastFrame, lastScan,acqDone);

Esta función se encarga de transferir los datos desde el buffer de adquisición hasta el buffer intermedio que está en memoria baja. Esta transferencia se realiza en el tiempo de la digitalización.

placa: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

scanOFrames: Dato de tipo **integer**. Variable de entrada, indica si la función MDAQ_Get tiene que transferir tramas o muestras. En nuestro caso devuelve tramas. por tanto esta variable debe ser igual a uno. En el caso de devolver muestras sería cero.

getOTap: dato de tipo **integer**. Esta variable indica el método de transferencia usado.

getOTap = 0 La transferencia se hace por el método llamado GET. Esto indica que se transfieren los datos tomando los mas antiguos. En mi programa se ha elegido este método.

getOTap = 1 La transferencia se hace por el método llamado. TAP. Esto indica que se transfieren los datos tomando los más recientes.

numToGet: dato de tipo **long**. Esta variable indica el numero de muestras o tramas devuelto por el buffer de adquisición en mi programa se devuelven las tramas de una en una.

startFrame: dato de tipo **long**. Esta variable indica el número de la de la trama que será copiado. En el programa este valor es cero lo que quiere decir que se copiará la última trama adquirida.

startScan: dato de tipo **long**. Esta variable indica que muestra será copiada al buffer intermedio. Cuando la adquisición se orienta a tramas como en este caso el valor de esta variable es ignorado.

tiemeout: dato de tipo **long**. Esta variable indica el número de ticks de reloj a esperar para datos que no hayan sido adquiridos.

timeout = -1 Espera indefinidamente.

tiemeout = 0 Retorna inmediatamente si el dato no ha sido adquirido. Este es el valor que toma la variable en mi programa.

getBuffer: esta variable es un puntero de tipo huge **integer**. Es un espacio de memoria definido en la memoria baja que almacena una trama de forma provisional antes de ser guardada en el disco duro. En el caso de digitalizar dos canales los datos se almacenan en el mismo de forma alternada como se indica a continuación:

(muestra de canal 1, muestra de canal 0), (muestra de canal 1, muestra de canal 0), (muestra de canal 1, muestra de canal 0), etec.

numGotten: dato de tipo **long**. Esta variable indica el número de tramas que fueron copiadas desde el buffer de adquisición hasta el buffer intermedio.

lastFrame: dato de tipo **long**. Esta variable indica el número de la última trama que fue copiada desde el buffer de adquisición hasta el buffer intermedio.

lastScan: dato de tipo **long.** Esta variable indica lo mismo que la anterior pero para el caso de que la adquisición sea orientada a scans.

acqDone: dato de tipo **integer.** Esta variable retorna un cero si la adquisición ha sido completada y un uno si no se ha completado. Si como en este caso se ha elegido adquisición continua esta variable siempre valdrá cero.

errNum: Si la función se ejecuta correctamente devuelve un cero caso contrario un número distinto de cero que es el código de error correspondiente.

errNum=MDAQ_Clear (placa);

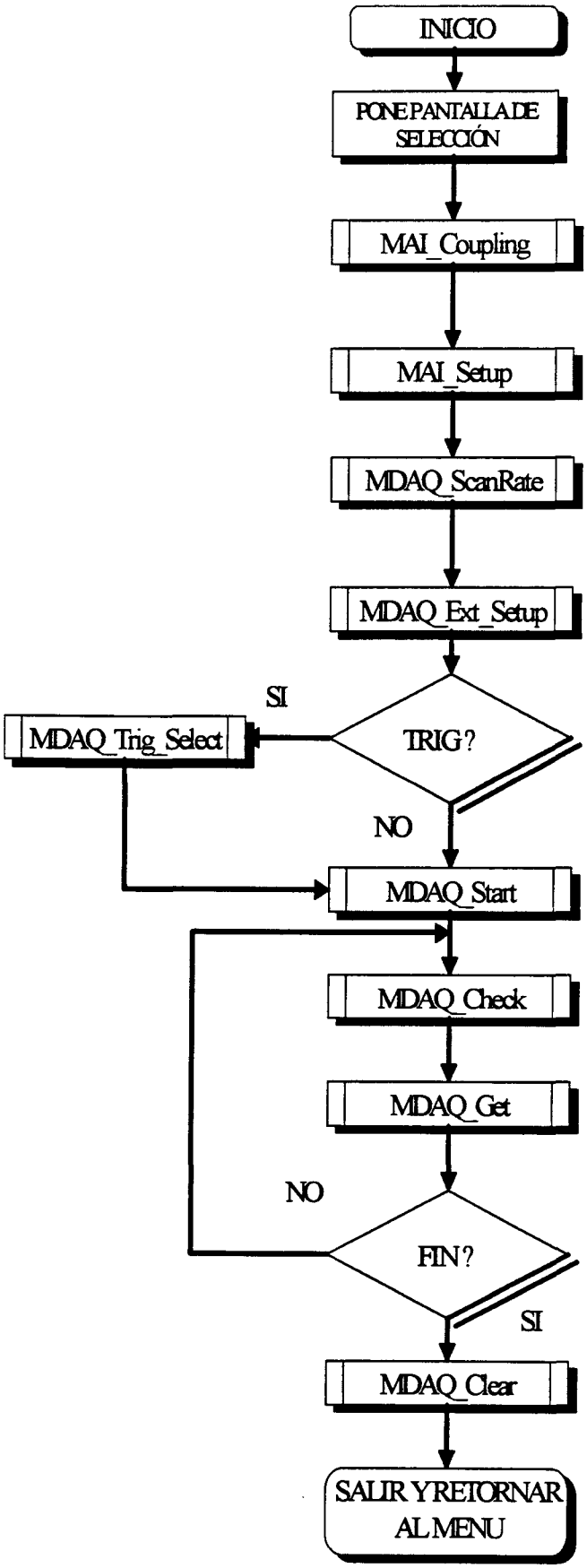
Esta función para la adquisición permite que la aplicación pueda terminar correctamente.

placa: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

errNum: Si la función se ejecuta correctamente devuelve un cero caso contrario un número distinto de cero que es el código de error correspondiente.

Tenemos que hacer notar que toda las funciones que hemos comentado antes devuelven un valor cuando son llamadas. Así después de llamar a cada función en el renglón seguido llamamos a la función **Errprint();** esta función muestra en pantalla el mensaje error correspondiente y la ejecución del programase interrumpe.

En la siguiente página podemos ver un diagrama de flujo del proceso de adquisición.



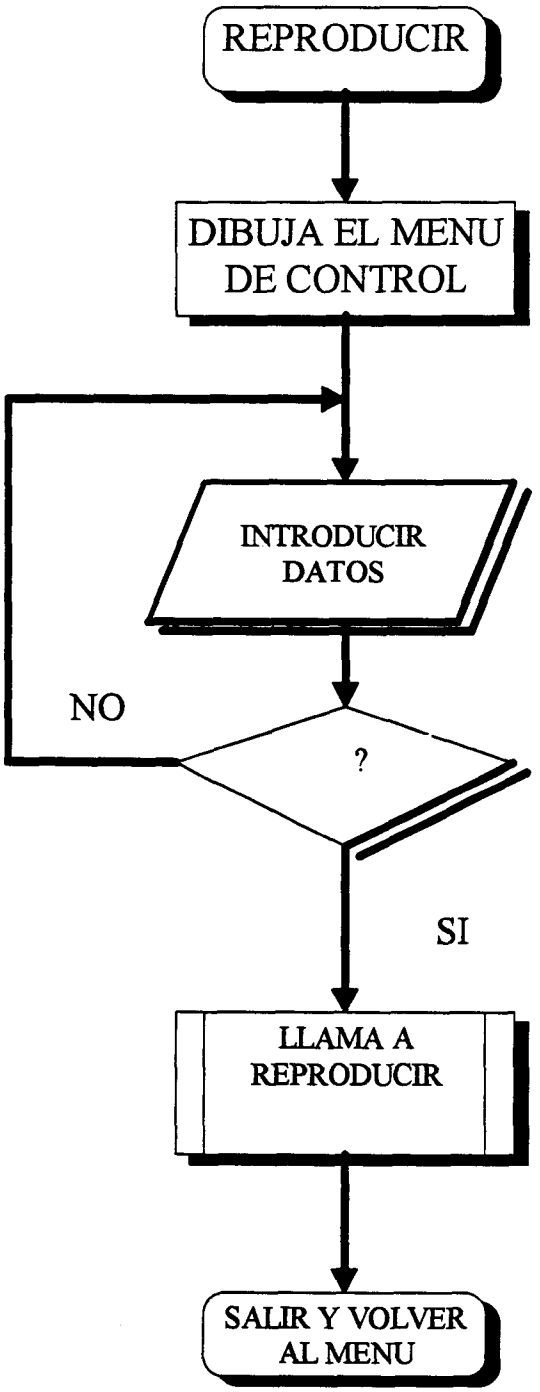
5.3.10 Función que permite reproducir un fichero de forma controlada.

Este procedimiento perteneciente al grupo de E/S también está dividido en dos partes, la primera es la encargada de dibujar la pantalla de diálogo que permite al usuario introducir los datos para generar un fichero controlando de forma manual la frecuencia a la que será generado este. Cuando usamos esta función se ignara la cabecera del fichero y se usan como parámetros de la generación los introducidos por el usuario. Esto tiene la ventaja de poder experimentar con diferentes frecuencias de muestreo observando como influyen en la generación de datos.

En el menú de la aplicación y antes de pulsar el botón GO! debemos especificar el nombre del fichero, la frecuencia de este y si queremos que sea generado por uno o dos canales. Cuando indicamos una frecuencia de generación distinta de la que se muestrearon los datos pueden ocurrir dos cosas: si la frecuencia que indicamos es mayos que la que se usó para muestrear el fichero este se reproducirá de forma acelerada, como cuado oimos un disco a de 45 revoluciones a 75. Si por el contrario indicamos una frecuencia menor el efecto producido es el contrario y oímos el fichero relentizdo. Tambien podemos reproducir un fichero que hayamos digitalizado en un solo canal por ambos canales, para ello debemos doblar la frecuencia de muestreo.

Como dijimos al principio esta rutina consta de dos partes la primera que dibuja la pantalla y al segunda que se encarga de generar el fichero. La rutina de generación es la que ya hemos explicado en el apartado 5.3.3 y usa la técnica de doble buffer.

A continuación se muestra el diagrama de flujo de dicho procedimiento.

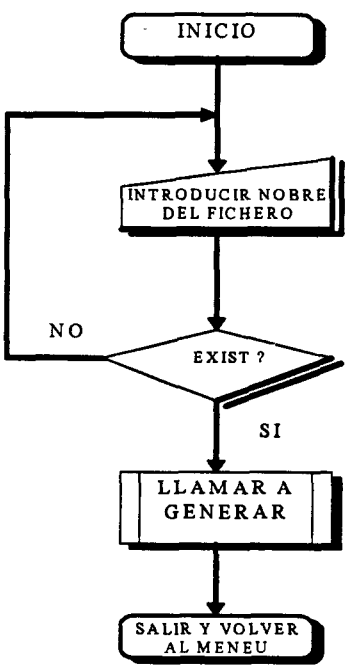


SI LOS DATOS NO HAN SIDO COMPLETADOS NO SE LLAMA A GENERAR

5.3.11 Función que permite reproducir un fichero con solo introducir su nombre.

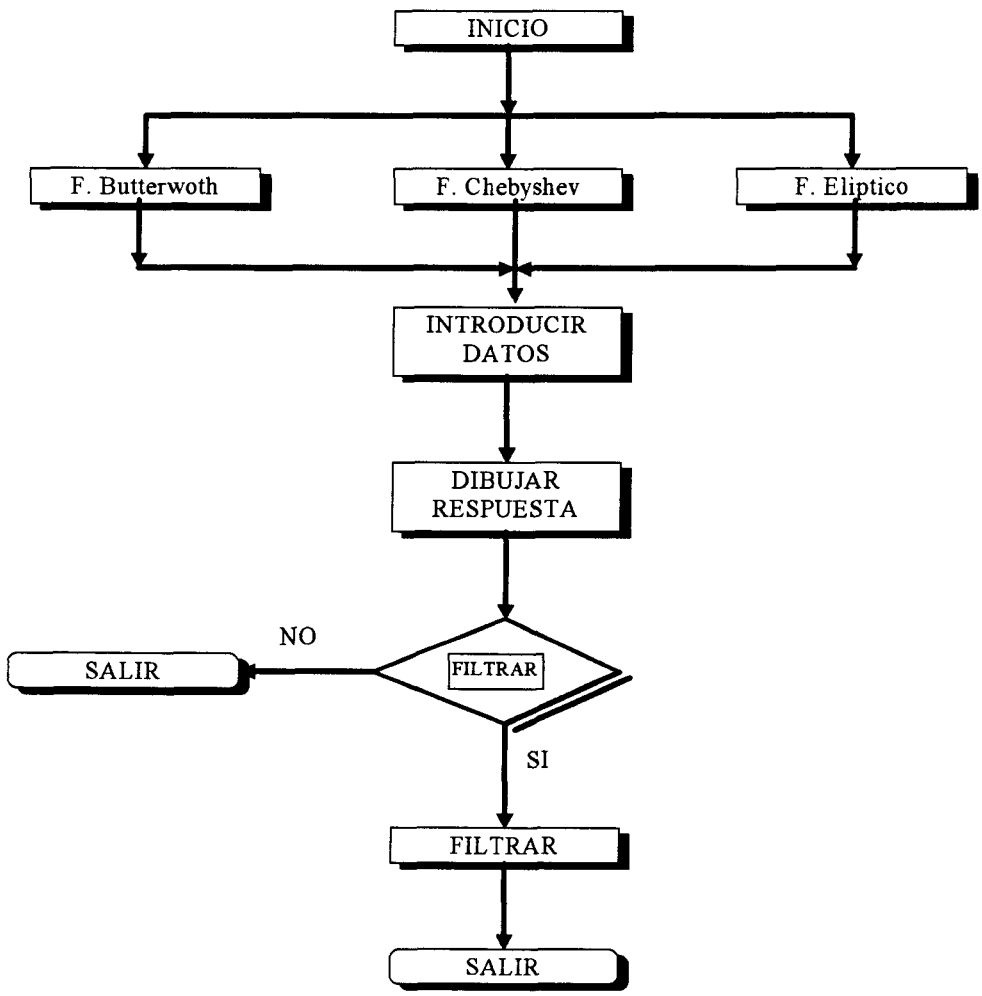
Este procedimiento es básicamente el anterior pero abrebiado. Se diseño con el fin de que se pudiran generar los ficheros de una forma rápida sin tener que estar especificando los datos por teclado o raton.

El funcionamiento es por tanto muy parecido sólo que aquí no tenemos pantalla de presentación sino que se abre una ventana para que indiquemos el nombre del fichero luego de comprobar que el fichero existe se abre y se lee la cabecera para obtener el número de canales y la frecuancia de muestreo y posteriormente se llama a la rutina reproducir. En el siguiente diagrama de flujo podemos ver este procedimiento.



5.3.12 Función que permite calcular un filtro y filtrar un fichero.

Este procedimiento pertenece al grupo de rutinas de filtrado. Estas rutinas estan estructuradas de forma que comparten las mismas funciones, la presentación por pantalla y algoritmo de filtrado. Los tres tipos de filtros que están implementados el Butterworth, Chebyshev y elíptico tienen una rutina encargada de realizar el cálculo de sus coeficientes y luego llaman a las rutinas de presentación y filtrado de forma común. En la figura siguiente se muestra un diagrama de como funciona esta parte del programa.



Como hemos dicho todas las funciones son comunes y la única diferencia consiste en indicar que tipos de coeficientes se calcularan. A continuación se

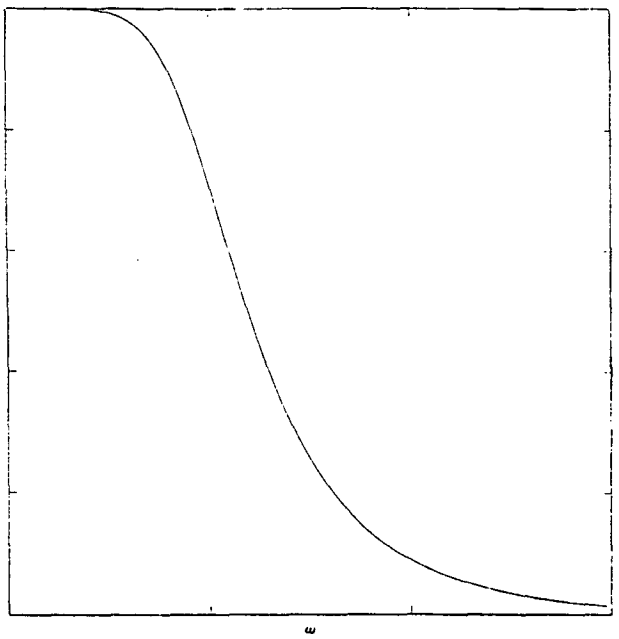
describen las funciones que realizan el calculo de los coeficientes de los filtros y el algoritmo de filtrado.

Lo primero es llamar a la rutina **dsp_coeficientes** que es la encargada de dibujar la pantalla para recojer los datos para realizar el cáculo de los coeficientes del filtro. En función de el tipo de filtro elegido se solicta al usurio que introduzca los datos. la función **dsp_coeficientes** acepta como argumento un entero que nos indica el tipo de filtro seleccionado. luego se dibujan los controles y se queda a la espera de recivir datos dentro de un bucle do.. while, existe la posibilidad de salir pulsando la tecla "exit". No se permite realizar el cálculo de los coeficientes hasta que no se hayan rellenado todos los datos, esto impide que le podamos pasar a la función valores sin significado. Cuando hayamos rellenado todos los datos y pulsemos "GO!" se realiza una llamada al procedimiento **coeficientes.** Este procedimiento se encarga de realizar el caáculo de los coeficientes del filtro para ello hace uso de las tres rutinas NI-DSP.**DSP_Bw_Coef(), DSP_Ch_Coef()** y **DSP_Elp_Coef()**. Ahora explicaremos el funcionamiento de cada una de ellas.

estado=DSP_Bw_Coef(placa, fs, f_low, f_hi, orden, a, b, tipo);

Esta función NI-DSP calcula los coeficientes de un filtro digital tipo Butterworth paso bajo, paso alto, rechaza banda o pasa banda. Los filtros Butterworth tienen una caida poco pronunciada pero una buena caractristicas dentro de la banda de paso. En la figura 1 podemos ver la curva de respuesta de este tipo de filtros

Figura 1. Respuesta de un filtro Butterworth



placa: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

fs: dato de tipo **float** que indica la frecuencia de muestreo para el cáculo de el fitro.

f_low: dato de tipo **float** que indica la frecuencia de corte inferior del filtro. En el caso de un filtro paso alto este valor debe ser cero. El valor de f_low debe se menor o igual que fs/2($f\_low \leq fs/2$).

f_hi: dato de tipo **float** que indica la frecuencia de corte superior del filtro. En el caso de un filtro paso bajo debe ser cero. El valor de f_hi debe se menor o igual que fs/2 ($f\_hi \leq fs/2$) .

orden: dato de tipo **integer** que indica el orden del filtro. Este dato debe se siempre mayor de cero.

tipo: dato de tipo **integer** que indica el tipo de filtro que queremos calcular. El valor se especifica segun la tabla 2.

Tabla 2. Equivalencia de códigos.

VALOR DE TIPO	TIPO DE FILTRO
0	Filtro Paso Bajo
1	Filtro Paso Alto
2	Filtro Paso Banda
3	Filtro Rechaza Banda.

a: dato de tipo **float** o de tipo **DSPHandle**. Esta varible es un array de numeros enteros donde se guarda el resultado de los coeficientes directos del filtro.

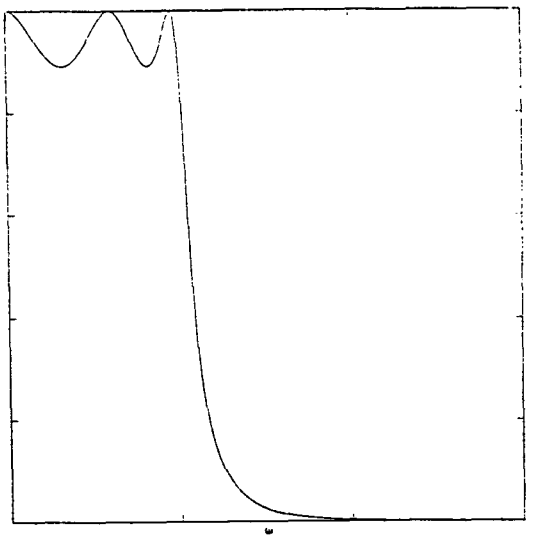
b: dato de tipo **float** o de tipo **DSPHandle.** Esta varible es un array de numeros enteros donde se guarda el resultado de los coeficientes inversos del filtro.

estado: Si la función se ejecuta correctamente devuelve un cero caso contrario un número distinto de cero que es el código de error correspondiente.

estado = DSP_Ch_Coef (placa, fs, f_low, f_hi, orden, rizado, a, b, tipo);

Esta función NI-DSP calcula los coeficientes de un filtro digital tipo Chebyshev paso bajo, paso alto, rechaza banda o pasa banda. Los filtros Chebyshev tienen buenas caracteristicas de caida pero por contra tienen un rizado en la banda de paso. En la figura 2 podemos ver la curva de respuesta de este tipo de filtros

Figura 2 . Respuesta de un filtro Chebyshev



placa: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

fs: dato de tipo **float** que indica la frecuencia de muestreo para el cáculo de el fitro.

f_low: dato de tipo **float** que indica la frecuencia de corte inferior del filtro. En el caso de un filtro paso alto este valor debe ser cero. El valor de f_low debe se menor o igual que fs/2(f_low $\leq$ fs/2)

f_hi: dato de tipo **float** que indica la frecuencia de corte superior del filtro. En el caso de un filtro paso bajo debe ser cero. El valor de f_hi debe se menor o igual que fs/2 (f_hi $\leq$ fs/2) .

orden: dato de tipo **integer** que indica el orden del filtro. Este dato debe se siempre mayor de cero.

rizado: dato de tipo **float** que indica el rizado en decibelios de la banda de paso. El valor debe ser mayor de cero. Como sabemos los filtros de tipo Chebyshev tienen una

caida mayor fuera de la banda de paso que los Butterworth pero por el contrario el rizado en la banda de paso es mayo y debe ser especificado en esta variable

tipo: dato de tipo **integer** que indica el tipo de filtro que queremos calcular. El valor se especifica segun la tabla 2.

a: dato de tipo **float** o de tipo **DSPHandle**. Esta varible es un array de numeros enteros donde se guarda el resultado de los coeficientes directos del filtro.

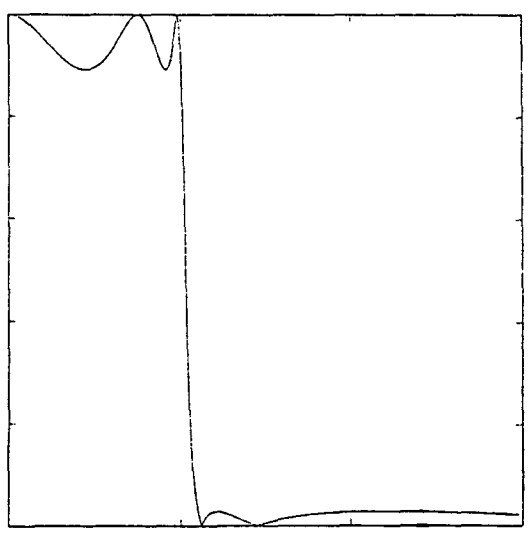
b: dato de tipo **float** o de tipo **DSPHandle.** Esta varible es un array de numeros enteros donde se guarda el resultado de los coeficientes inversos del filtro.

estado: Si la función se ejecuta correctamente devuelve un cero caso contrario un número distinto de cero que es el código de error correspondiente.

estado = DSP_Elp_Coef (placa, fs, f_low, f_hi, orden, rizado, atenuacion, a, b, tipo);

Esta función NI-DSP calcula los coeficientes de un filtro digital elíptico paso bajo, paso alto, rechaza banda o pasa banda. Los filtros elípticos son los que mejores caracteristicas de caida tienen fuera de la banda de paso pero a costa de más rizado

Figura 3. Respuesta de un filtro Eliptico



placa: dato de tipo **integer** que indica el slot donde esta conectada la tarjeta. El los ordenadores con Bus ISA este número no es significativo, pero debe ser el mismo que el que indicamos cuando configuramos la tarjeta con el fichero DAQCONF.EXE.

fs: dato de tipo **float** que indica la frecuencia de muestreo para el cáculo de el fitro.

f_low: dato de tipo **float** que indica la frecuencia de corte inferior del filtro. En el caso de un filtro paso alto este valor debe ser cero. El valor de f_low debe se menor o igual que fs/2($f\_low \leq fs/2$)

f_hi: dato de tipo **float** que indica la frecuencia de corte superior del filtro. En el caso de un filtro paso bajo debe ser cero. El valor de f_hi debe se menor o igual que fs/2 ($f\_hi \leq fs/2$) .

orden: dato de tipo **integer** que indica el orden del filtro. Este dato debe se siempre mayor de cero.

rizado: dato de tipo **float** que indica el rizado en decibelios de la banda de paso. El valor debe ser mayor de cero y menor que el valor de la **atenuacion**.

atenuacion: dato de tipo **float** que indica la atenuación en decibelios fuera de la banda de paso. El valor debe ser mayor de cero.

tipo: dato de tipo **integer** que indica el tipo de filtro que queremos calcular. El valor se especifica segun la tabla 2.

a: dato de tipo **float** o de tipo **DSPHandle**. Esta varible es un array de numeros enteros donde se guarda el resultado de los coeficientes directos del filtro.

b: dato de tipo **float** o de tipo **DSPHandle.** Esta varible es un array de numeros enteros donde se guarda el resultado de los coeficientes inversos del filtro.

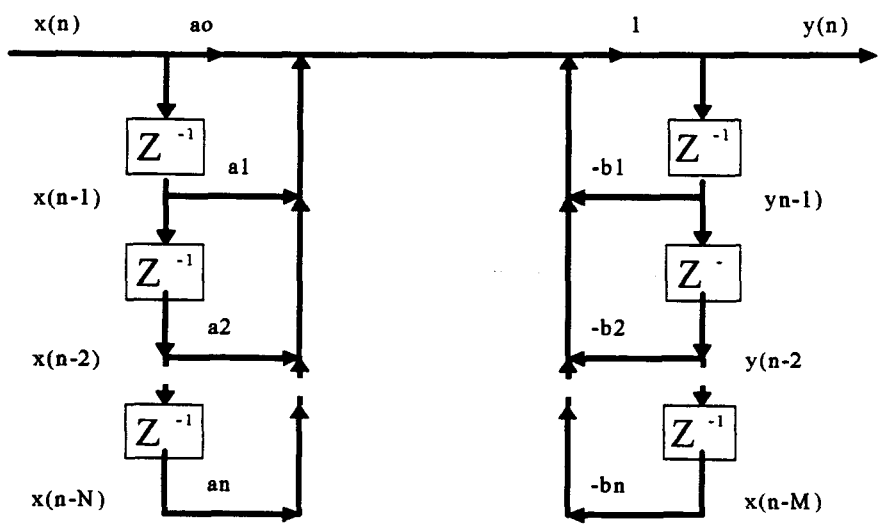
estado: Si la función se ejecuta correctamente devuelve un cero caso contrario un número distinto de cero que es el código de error correspondiente.

Estas funciones que acabamos de describir nos dan los valores para calcular las curvas de respuesta del filtro y realizar el algoritmo del filtrado de un filtro IIR (Filtro recursivo).

Después de cálcular los valores se los pasamos a la función **PoneCoeficientes** que es la encargada de mostrar los coeficientes del filtro en la pantalla. Cuando los coeficientes han sido mostrados lo que hacemos es llamar a la función **freqz** que se encarga de calcular y mostrar las las curvas de respuesta de magnitud y fase del filtro. En este punto podemos salir o filtrar un fichero de los tengamos en el disco duro aplicandole las características del filtro que acabamos de diseñar.

Como el objetivo de este trabajo no es el desarrollo y estudio de los filtros digitales, sino que es parte del mismo no voy a extenderme mucho en este tema, que por otra parte podría ser materia de un trabajo por si mismo. La rutina de filtrado que he programado realiza un filtrado de tipo IIR segun la forma directa I, esta forma se muestra en la figura 4.

Figura 4. Forma directa I



De esta estructura se deduce la ecuación:

$$y(n) = a_0 * x(n) + a_1 * x(n-1) + a_2 * x(n-2) + \ldots + a_n * x(n-N) - b_1 * y(n-1) - {}_2 * y(n-2) - \ldots - b_n * y(n-M)$$

Calculando la transformada Z de esta ecuación tenemos la función h(z) que nos relacina la entrada con la salida.

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_0 + a_1 * Z^{-1} + a_2 * Z^{-2} + \ldots + a_n * Z^{-N}}{1 + b_1 * z^{-1} + b_2 * Z^{-2} + \ldots b_n * Z^{-M}}$$

Esta es pues la función de transferencia de un filtro digital IIR (Infinite Inpulse Response), tambien llamado filtro recursivo o ARMA (Autoregresive moving-average filter). Para calcular este algoritmo el ordenador debe realizar N+M+1 multiplicaciones y N+M sumas, existen otras estructuras que requieren más cálculo y también menos pero nosotros nos vamos a centrar en la que hemos comentado.

Las rutinas DSP de las que hablamos antes realizan el cálculo de los coeficientes y la rutina de filtrado con esos coeficientes realiza la implementación del filtro. Para realizar esta rutina se parte de la ecuación de diferencias que relaciona la señal de salida con la entrada de la forma mostrda en lafigura 4. En un proceso de filtrado hay que tener en cuenta que las condicones iniciales y las finales, normalmente la primera vez que realizamos el filtrado las condiciones iniciales son cero a menos que tengamos otras procedentes de otro proceso los datos fueran de una longitud sufiente para se cargados en memoria y filtrados en una sola pasada no sería muy

dificil realizar el algoritmo de filtrado. El problema surge cuando tenemos un fichero extenso que no puede ser cargado en memoria. Los datos deben ser cogidos en trozos, filtrados y guardados en otro fichero de forma sucesiva hasta que se haya completado el filtrado. Es en este caso donde las condiciones finales juegan un papel muy importante, si no se tienen en cuenta las condiciones finales del primer trozo como condiciones iniciales a la hora de filtrar el segundo trozo nos encontraremos con discontinuidades entre las porciones filtradas. Es por esto muy importante prestar mucha atención a las condiciones finales. A continuación se describe el método seguido para hacer el algoritmo.

Supongamos que la ecuación de diferencias es de orden dos y que queremos realizar el filtrado de un array de datos que contine 20 muestras que tomaremos de cinco en cinco. La ecuación para el orden dos se muestra a continuación.

(5.1)

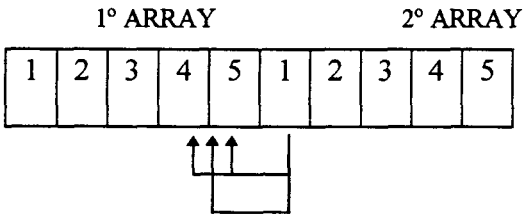
$$y(n) = a_0 * x(n) + a_1 * x(n-1) + a_2 * x(n-2) - b_1 * y(n-1) - b_2 * y(n-2)$$

La ecuación va tomando valores en función de lo que valga n

(5.2)

$$n = 0 \Rightarrow y[0] = a_0 * x[0] + a_1 * x[-1] + a_2 * x[-2] - b_1 * y[-1] - b_2 * y[-2]$$
$$n = 1 \Rightarrow y[2] = a_0 * x[1] + a_1 * x[0] + a_2 * x[-1] - b_1 * y[0] - b_2 * y[-1]$$
$$n = 2 \Rightarrow y[2] = a_0 * x[2] + a_1 * x[1] + a_2 * x[0] - b_1 * y[1] - b_2 * y[0]$$
$$n = 3 \Rightarrow y[3] = a_0 * x[3] + a_1 * x[2] + a_2 * x[1] - b_1 * y[2] - b_2 * y[1]$$
$$n = 4 \Rightarrow y[4] = a_0 * x[4] + a_1 * x[3] + a_2 * x[2] - b_1 * y[3] - b_2 * y[2]$$
$$n = 5 \Rightarrow y[5] = a_0 * x[5] + a_1 * x[4] + a_2 * x[3] - b_1 * y[4] - b_2 * y[3]$$

Supongamos que el array de datos tiene la forma:



Para entender el proceso de filtrado hay que pensar que cojemos los datos de cinco en cinco de esta forma para los primeros cinco datos las condicones iniciales las podemos cosiderar cero pero cuando tomemos los otros cinco debemos impedir que haya una ruptura con los datos anteriores, para ello lo que hacemos es que guardamos las condiciones finales del primer bloque y las aplicamos como condiciones iniciales en el segundo bloque sumando estas al resultado. Podemos ver en las ecuaciones anteriores como en el primer caso cuando evaluamos el dato y[0] tenemos que considerar los datos del bloque anterior para a1 y a2 y b1 y b2 en el segundo caso para y[1] solo tenemos que tener en cuenta las condiciones finales para b2 y en el tercer paso y[2] ya no es necesario evaluar las condicones finales del bloque anterior. Podemos decir que el bloque es autónomo. Este procedimiento hay que hacerlo con cada bloque. En general podemos deducir que el número de pasadas en las que hay que tener en cuenta el orden las C.F. es igual al orden del filtro. En las ecuaciones siguientes podemos ver como se usan las condiciones finales de un proceso como condicones iniciles del siguiente.

(5.3)

$$C.F.[0] \Rightarrow y[0] = a_1 * x[4] + a_2 * x[3] - b_1 * y[4] - b_2 * y[3]$$
$$C.F.[1] \Rightarrow y[1] = a_2 * x[4] - b_2 * y[4]$$

Estas son las condiciones finales despues de tratar el primer bloque y deben ser sumadas como condiciones iniciales al bloque siguiente como se indica en las ecuaciones siguientes.

(5.4)

$$y[0] = a_0 * x[0] + C.F.[0]$$
$$y[1] = a_0 * x[1] + a_1 * x[0] - b_1 * y[0] + C.F[1]$$

Otra parte interesante del conjunto de rutinas de filtrado es la encargada de realizar el cáculo de la respuesta en magnitud y fase del filtro.

La relación entre la entrada y la salida para un filtro IIR se define tambien segun la ecuación 5.5

(5.5)

$$y(n) = \sum_{k=0}^{M} a(k) * x(n-k) - \sum_{k=1}^{N} b(k) * y(n-k)$$

La función de transferencia del filtro se define como la relación Y(Z)/ X(Z) , done Y(z) es la trasnformada Z de la salida y(n) y X(z) es la transformada Z de la entrada x(n). Usando la definición de la trasnformada z podemos obtener la función de transferencia de de un filtro IIR defindo en 5.5.

$$H(z) = \sum_{n=0}^{\infty} h(z) * Z^{-n} \quad (5.6)$$

Esta función de tranferencia nos da la relación entre los términos a(n) y b(n) tal como se muestra en la ecuación 5.7.

$$H(z) = \frac{\sum_{n=0}^{M} a(n) Z^{n}}{\sum_{n=0}^{N} b(n) Z^{n}} = \frac{A(z)}{B(z)} \quad (5.7)$$

La respuesta en frecuencia del filtro se calcula poniendo $z = e^{jw}$ en la ecuación 5.6. El resultado de esto se ve en la ecuación 5.8.

$$H(\omega) = \sum_{n=0}^{\infty} h(n) e^{-jn\omega} \quad (5.8)$$

En realida la expresión corecta sería -jwTn pero para simplificar y asumiendo el periodo de muestreo T=1 podemos dejar la ecuación 5.8. El resultado de la ecuación anterior en un valor complejo formado por la magnitud y la fase. La respuesta en frecuencia es una función de varible y periodo 2π como se demuestra a continuación .

$$\begin{aligned} H(\omega + 2\pi) &= \sum_{n=0}^{N-1} h(n) e^{-j(\omega + 2\pi)n} \\ H(\omega + 2\pi) &= \sum_{n=0}^{N-1} h(n) e^{-j\omega n} e^{-j2\pi n} \quad (5.9) \\ H(\omega + 2\pi) &= H(\omega) \end{aligned}$$

La frecuencia viene dada por en radianes por segundo o en hercios con la realación $= 2\pi f$.

Para calcular la respuesta lo que tenemos que hacer en calcular la FFT del numerador y del denominador de la ecuación 5.8. Evaluamos una serie de puntos

fijos, en mi caso 256, como el orden del filtro está limitado a 128 y normalmente será menor para poder calcular la FFT debemos rellenar de ceros los vectores hasta 256. Es decir si el filtro es de orden 4=M=N, habrá que poner 4-256= 252. Si llamamos L a los puntos a cacular la FFT será L-M y L-N lo que tengamos que rellenar de ceros. La ecuación que tenemos pues que implemetar será la siguiente:

$$H\left(\frac{2\pi * k}{L}\right) = \frac{FFT\{a(n)\}}{FFT\{b(n)\}} \quad (5.10)$$

Esta ecuación se deduce sabiendo que la FFT se define como:

$$c(k) = \sum_{n=0}^{n-1} h(n) e^{-j2*k*\pi/L} \quad (5.11)$$

Para K=0,1,...L-1.

Si comparamos las ecuaciones 5.11 y 5.8 tenemos que:

$$c(k) = H(\omega)|\omega = 2 * k * \pi / L = H\left(\frac{2k\pi}{L}\right) \quad (5.12)$$

Para K=0,1,...,N-1

A continuación en la página siguiente mostrará el diagrama de flujo del procedimiento de filtrado.

6.4 PROGRAMAS ADICIONALES

De forma separada se suministran varios programas más que están pensados para funcionar de forma autónoma y que sirven como herramientas para mejorar la eficacia del trabajo. Estas utilidades son concretamente tres, un conversor del formato

WAV de Windows al formato que yo le he dado mis ficheros, un conversor que permite transformar un fichero generado con la AT-DSP2200 al formato WAV de Windows, por último se presenta un visor de ficheros a pantalla completa que nos permite ver los ficheros en porciones de 2048 Kbytes.

5.4.1 Programa que transforma un fichero del formato DSP al formato WAV.

Como dice el título este programa realiza la conversión de formatos DSP a WAV. Antes de nada hay que explicar el formato WAV que pasaremos a hacer a continuación.

Aunque normalmente solemos denominar como ficheros WAV a todos aquellos que cumplen con las especificaciones multimedia de Windows, la realidad es que el verdadero nombre de estos ficheros es RIFF (Resource Interchange File Format), estos ficheros tiene unas prestaciones muy amplias aunque Windows solo los usa para guardar sonidos digitalizados. Hay que destacar que este formato no afecta a la utilización del mismo si no al formato de los datos, de hecho estos ficheros pueden ser leídos o escritos de forma normal.

En los ficheros RIFF la información se agrupa en bloques denominados "chunks". Cada chunks consta de tres campos: un código de cuatro caracteres que hace las veces de identificador de chunk, una palabra doble que especifica la longitud del mismo en bytes y un campo de datos. Existen dos tipos de chunks que pueden contener en su interior otros chunks (en este caso subchuncks) que son el RIFF y el

LIST. En concreto, el primer chunk en el fichero siempre tiene que ser un RIFF y todos los demás chunks están contenidos en este.

El chunk RIFF incluye un campo adicional en los cuatro primero bytes del campo de datos que es el que identifica el tipo de información y estructura del fichero (recibe el nombre de form type). Por ejemplo, los ficheros de audio de Microsoft tienen el identificativo WAVE.

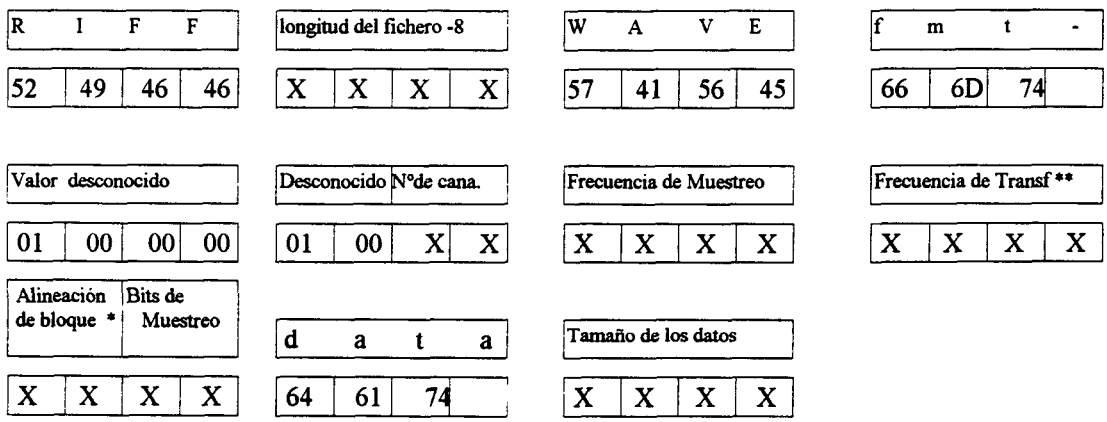
El chunk LIST también incluye un campo adicional, situado igualmente en los cuatro primeros bytes del campo del campo datos. En este caso lo que se especifica es el tipo lista (list type), código de cuatro caracteres que identifica el contenido de la misma. Un ejemplo sería una lista de tipo INFO constituida por los subchunks y ICRD, es decir, información sobre el copyright y la fecha de creación del fichero.

Hasta aquí hemos visto la definición formal de un fichero RIFF. En el caso de Windows este formato se ha modificado ligeramente, por ejemplo solo tiene el chunk RIFF y no el LIST. Las versiones más avanzadas de algunos compiladores de lenguaje C incorporan rutinas para el manejo de estos formatos de forma casi automática, pero no era el caso de la versión del compilador que yo usé. Además en la información sobre estos ficheros se especifica con detalle la forma de acceder su información y como se organizan los datos. En mi caso no me fue posible acceder a esta información en el momento de desarrollar la aplicación y resolví el problema de otra forma, quizá poco elegante pero efectiva. Estudié varios ficheros WAV digitalizados tanto en 8 como en 16 bits y en mono y estéreo y llegué a la conclusión de cual debería se la estructura de los mismos. Luego construí una aplicación que fuera capaz de recrearla

y se la añadí como cabecera a los ficheros generados por mi programa. El resultado ha sido satisfactorio y las aplicaciones Windows reconocen los ficheros como validos para ser reproducidos.

Del análisis de varios ficheros y de lo dicho sobre el formato RIFF se deduce que la cabecera de un fichero WAV esta formada por 44 bytes, en la figura 4 se muestra esta cabecera formando grupos de 2 bytes.

Figura 4. Cabecera de un fichero WAV.



* Numero de bytes en los que se empaqueta la información.

** La frecuencia de transferencia, que será igual la frecuencia de muestreo por el numero de bytes en los que se empaqueta la información.

Lo que hemos denominado como alineación de bloques es un valor que nos indica la unidad mínima de información. Es decir cuando tenemos un fichero de 16 bits mono la unidad mínima de información es 2 puesto que cada muestra se empaqueta en un 2 bytes. En la tabla 3 se muestra un tabla con los valores para los distintos tamaños de muestreo.

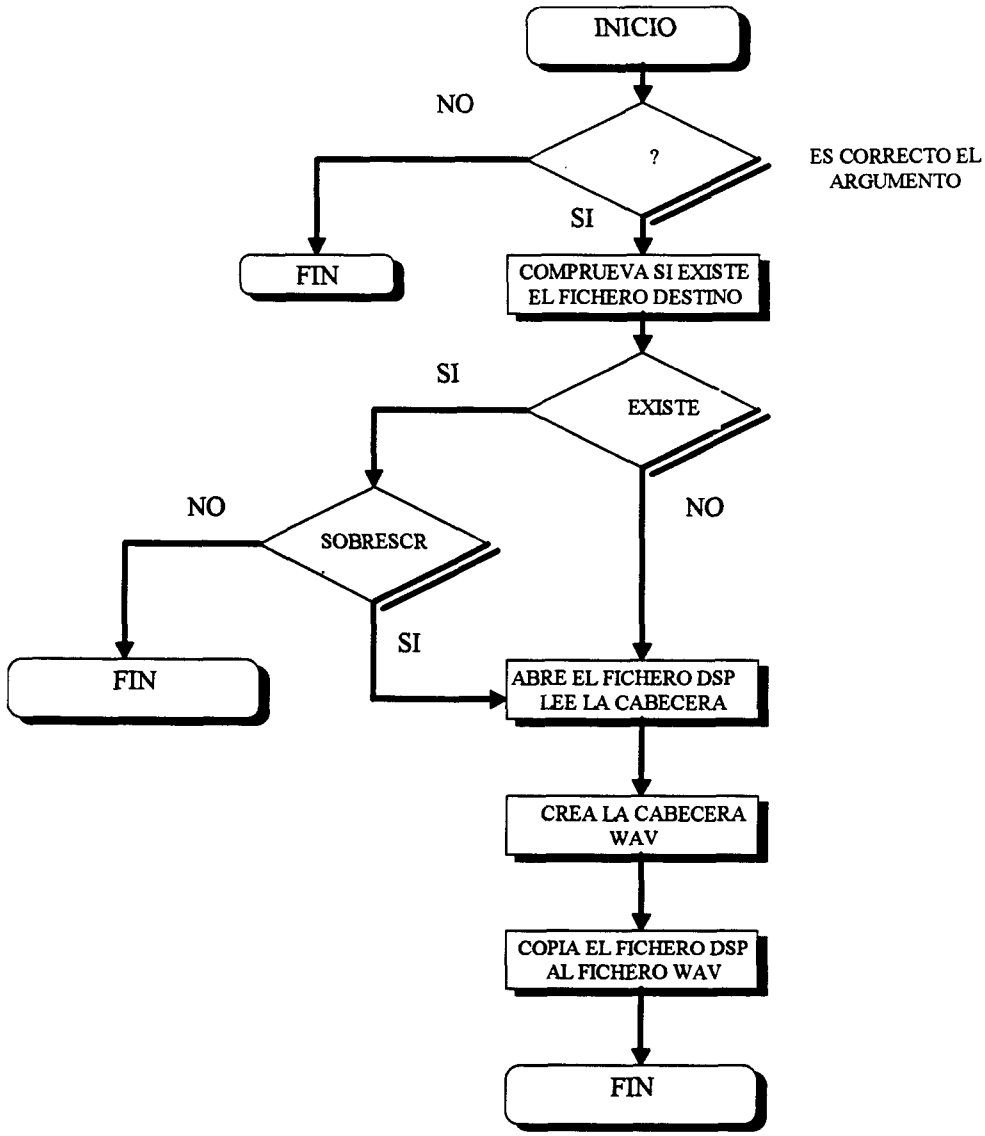
Tabla 3. Tabla de alineación de tramas.

TIPO DE MUESTREO	EMPAQUETADO
8 bits, mono	1 Byte
8 bits, estéreo	2 Bytes
16 bits, mono	2 Bytes
16 bits, estéreo	4 Bytes

Después de ver como es la cabecera de un fichero WAV para crear ficheros de este tipo solo es necesario construirla y añadírsela al fichero que queramos convertir en WAV. Esto es exactamente lo que hace el programa llamado DSP_WAV.EXE.

Al ejecutar este programa hay que darle dos argumentos, el nombre del fichero origen, que debe se un fichero DSP y el nombre del fichero destino, que será el fichero con formato WAV.

El diagrama de flujo de la página siguiente muestra el diagrama de flujo del programa.



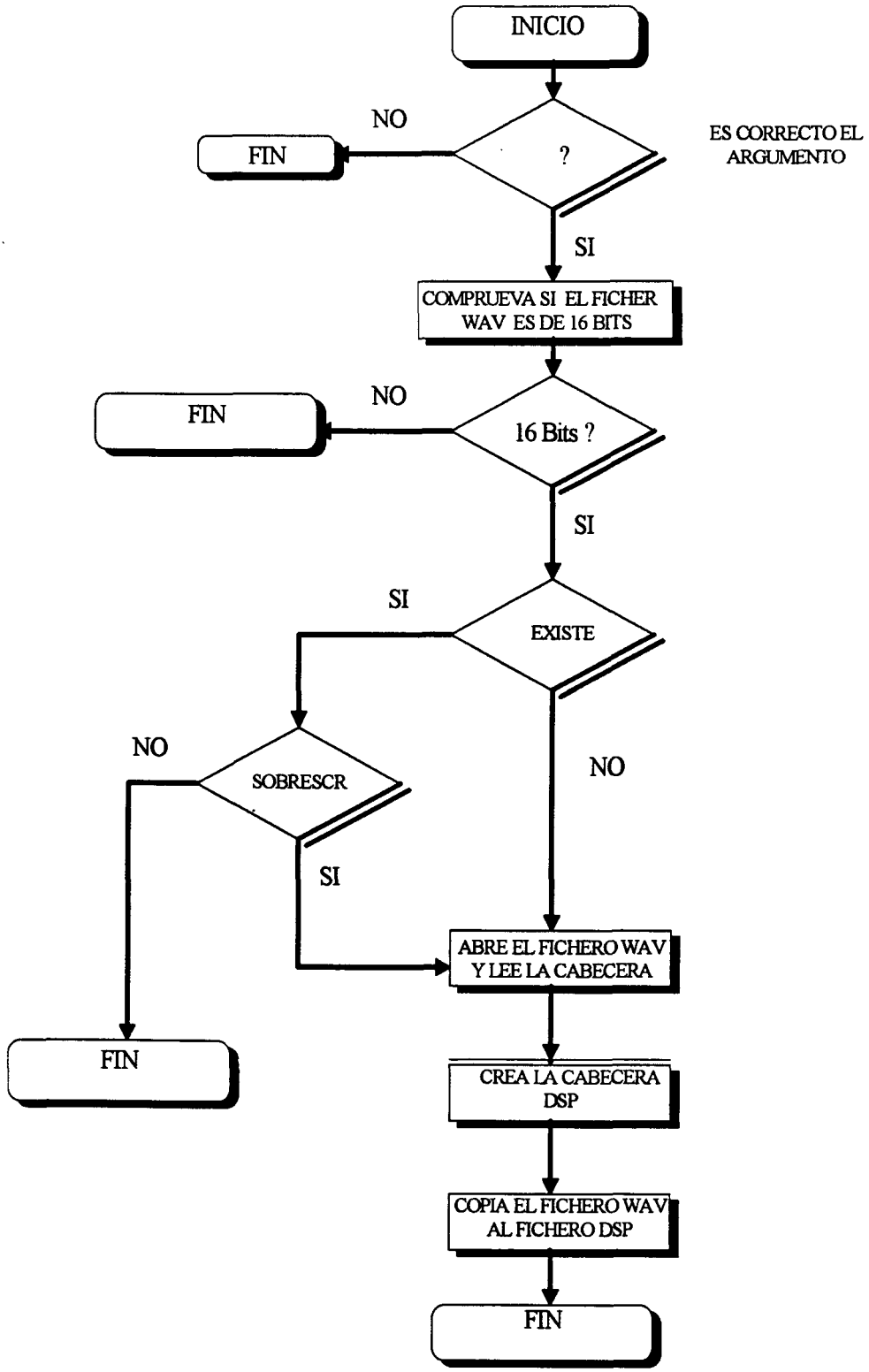
5.4.2 Programa que transforma un fichero del formato WAV al formato DSP.

En este caso realizamos la tarea contraria, lo que nos permite reproducir en nuestra tarjeta ficheros que hayan sido capturados con otras, con tal de que estas generen el formato WAV.

En este caso el proceso es el inverso. Primero comprobamos se el fichero WAV ha sido creado con una tarjeta de 16 bits en cuyo caso se puede convertir. Si no fuera así abandonaríamos la conversión. Luego extraemos la frecuencia de muestreo y el número de canales del fichero y creamos un fichero, que será el fichero DSP de salida. Por último copiamos un fichero en otro. Este programa, igual que el anterior funciona tecleando desde la línea de comandos el nombre del programa y el nombre de los dos ficheros el fuente y el destino. El fichero fuente será el fichero WAV y el destino el fichero DSP.

Estos dos programas se han diseñado para poder dar compatibilidad entre los ficheros creados con la AT-DSP2200 y las tarjetas de sonido más populares del mercado, tipo Sound Blaster, además de permitirnos que podamos aprovechar las herramientas para el procesado de señales que puedan estar disponibles.

En la página siguiente podemos ver el diagrama de flujo que nos muestra como se realiza la transformación del formato WAV al formato DSP.

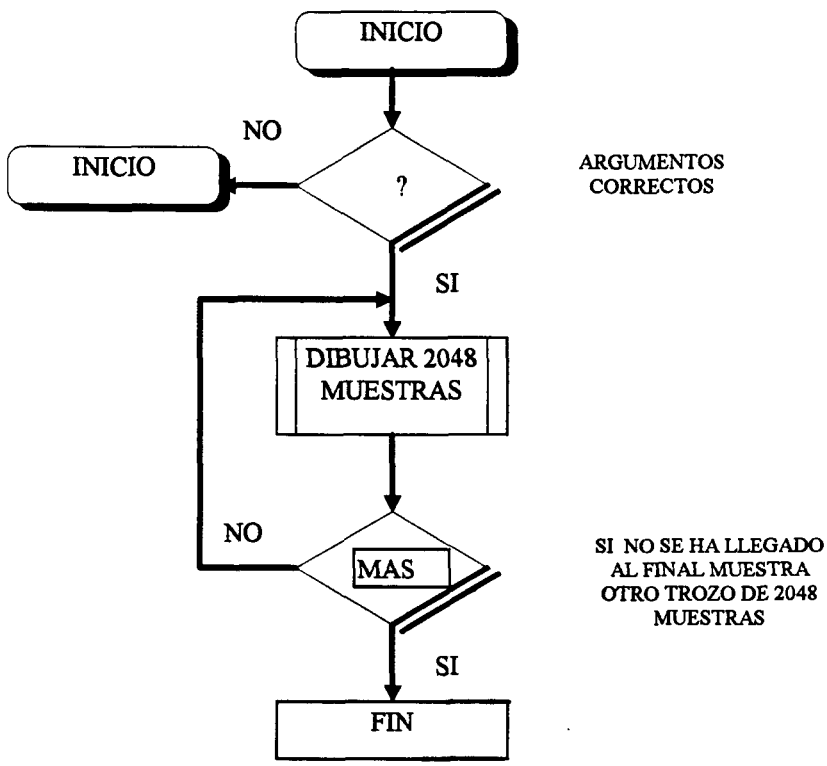


5.4.3 Visor defcheros

Con este programa he querido desarrollar una aplicación para poder comparar dos ficheros. La idea es permitirnos ver porciones de los mismos de forma simultánea en la pantalla para poder comparar uno con otro.

El programa funciona aceptando tres argumentos, el nombre del fichero 1 el nombre del fichero 2 y si los ficheros son mono o estéreo. Es importante tener en cuenta que no se puede comparar un fichero mono con uno estéreo pus ello produciría un error. Los ficheros son mostrados en porciones de 2048 Kbytes.

A continuación se muestra el diagrama de flujo de este programa.



1.1 INTRODUCCIÓN

En esta parte se especifican las condiciones de funcionamiento de la tarjeta y del software bajo las cuales se garantiza el correcto funcionamientos de este trabajo. El pliego de condiciones se refiere no sólo a la aplicación software que he desarrollado sino a la de los equipos donde este debe ejecutarse que deben cumplir unas norma mínimas.

1.2 CONFIGURACIÓN MÍNIMA.

La configuración mínima viene impuesta, primero por la tarjeta AT-DSP2200 y segundo por el software que he desarrollado. Aunque ambos requerimientos son muy similares en este caso, si se usa la AT-DSP2200 como herramienta de desarrollo independiente y sin hacer uso del programa, la configuración hardware mínima puede variar.

Para el caso que nos ocupa debemos poseer el siguiente soporte.

- Un PC AT 386 o superior con al menos 4 MB de RAM configurada como memoria extendida y con una velocidad mínima de 40 MHZ.
- Un Slot de 16 bits disponible en el PC.
- Un disco duro de al menos 40 MB y con un tiempo de acceso no superior a los 15 ms.
- Una tarjeta VGA con una resolución de al menos 640 x 480 y 256 colores.

- Un ratón de dos botones.

Como el programa ha sido hecho para funcionar bajo el sistema operativo MS-DOS es pues imprescindible que la configuración que hemos comentado antes este equipada con una versión de este sistema 5.0 o superior.

3.1 CONDICIONES DEL SOFTWARE.

Para que funcione correctamente el software que he desarrollado para la AT-DSP2200 aparte de contar con la plataforma hardware comentada en el apartado anterior en necesario contar con dos ficheros que acompañan al software que viene con la tarjeta. Estos ficheros son el DSP2200.OUT y el fichero DAQCONF.EXE. Cuando hayamos instalado el software con el programa de instalación que se suministra al efecto lo primero que debemos hacer es ejecutar el fichero DAQCONF.EXE y realizar la configuración de la tarjeta tal como se indica el manual adjunto a este trabajo. Si la AT-DSP2200 no esta conectada o está mal configurada el programa no funcionará.

También es importante asegurarnos de que el PC posee al menos 4MB de memoria extendida de lo contrario la aplicación de adquisición de datos no funcionará correctamente. Aunque si lo hará el resto del programa. Es importante tener en cuenta que si deseamos realizar algún proceso a las señales debemos disponer de un disco duro con una capacidad de al menos el doble del fichero que vamos a tratar y en algunos casos, cuando se trata de sonido estero, del triple.

El programa debe ser ejecutado desde un disco duro para que funcione correctamente.

4.1 CONDICIONES DE LA AT-DSP2200

En este apartado se especifican las condiciones de trabajo de la AT-DSP2200. Estas condiciones son independientes del software que esté corriendo sobre ella y se refieren en especial a su conexión.

La tarjeta tiene un consumo de 2.5 a 5V por lo que no debe conectarse a Ordenadores con fuentes de poca potencia o que estén sobrecargados. La temperatura de trabajo debe estar comprendida entre 0 °C y +70 °C, aunque se recomiendan 25 °C. El Ordenador al que se conecte debe tener sistema de ventilación y este debe permanecer tapado para ayudar a la ventilación.

LA tarjeta debe conectarse en una ranura de expansión del bus del PC de tipo ISA de 16 bits o una EISA, pero nunca en una de 8 bits.

En el momento de la instalación y posteriormente debe evitarse tocar con las manos los componentes y en especial el Procesador Digital, para evitar que la electricidad estática pueda dañarla. La tarjeta viene dentro de una bolsa antiestática y no debe sacarse de ella hasta el momento de la instalación.

Siempre que sea posible debemos conectar la Tarjeta en un Slot de forma que los espacios a ambos lados de la AT-DSP2200 queden despejados, con esto evitaremos introducir ruido a los conversores.

Si usamos el Bus RTSI, accesible sólo desde el interior, debe usarse un conector adecuado. Para elle hay que consultar a National Instruments que son los dueños del Bus.

Las conexiones de señal tanto de entrada como de salida se realizan mediante cinco conectores tipo RCA situados en la parte posterior, dos de ellos son para los canales de salida, otros dos para los de entrada y el quinto se usa como fuente de trigger digital. Deben respetarse las especificaciones impedancia y niveles de tensión que se citan el anexo 1, referido a las especificaciones técnicas.

PRESUPUESTO

1.1 INTRODUCCIÓN

El presupuesto se ha elabora en pensando en dotar a un laboratorio de una herramienta completa para el procesado de señales, por ello incluimos el precio del ordenador. En cuanto al precio del software desarrollado por mi, lo he hecho sin contar el tiempo que he tardado en aprender el lenguaje C y tampoco he contado el tiempo que he tardado en reunir toda la información. Es decir el cálculo esta hecho suponiendo que las horas dedicadas a la programación son horas efectivo.

El precio del software puede parecer algo excesivo pero hay que tener en cuenta que estos son los gastos de desarrollo contando en tiempo invertido. Para el caso de convertirlo en un producto comercial vendiendo el precio del paquete podría descender considerablemente.

2.1 PRESUPUESTO

NOMBRE DEL PRODUCTO	PRECIO
Ordenador tipo 386/40 con una disquetera de 1.44 MB, un disco duro de 420 MB, un ratón, un monitor color, una tarjeta de vídeo VGA y 4 MB de memoria RAM.....	145.000
Sistema Operativo MS-DOS 5.0.....	6.000
Tarjeta AT-DSP2200. Con 384 Kwords. Con el software NI-DAQ, NI-DSP y manuales.....	700.000
Software para adquisición, generación y filtrado.....	880.000
Compilador Microsoft C. Versión 6.0.....	23.000
PESETAS TOTALES.....	1.754.000

ANEXO I

ANEXO 1. ESPECIFICACIONES TÉCNICAS.

En este anexo se indican las especificaciones Técnicas de la AT-DSP2200 dadas por el fabricante.

PROCESADOR DSP.

- Procesador. WE DSP32C.
- Velocidad de reloj. 50 MHz.
- Ciclo de Instrucción. 80 nseg.
- Flujo Máximo. 25 MFLOPS, 12.5 MIPS.

MEMORIA.

- Memoria en el Procesador 1.5 Kwords de cero estados de espera.
- Memoria en la Tarjeta. 384 Kwords de cero estados de espera.

CONTROLADOR DE DMA.

- Controlador de DMA del WE DSP32C.
  - De la entrada paralela a memoria local.
  - De memoria local a la salida Paralela.
  - De la entrada serie a la memoria local.
  - De memoria local a la salida serie.

SOPORTE DE INTERRUPCIONES.

- A la AT-DSP2200 6 (hardware)

- Externa	PIF (INTREQ1)
	Trigger (INTREQ2)
- Interna	Registro de datos de E/S Paralelo lleno.
	Registro de datos de E/S Paralelo vacío.
	Buffer de entrada serie lleno.
	Buffer de salida serie vacío.
• Al PC	1 (hardware)
- Causada por	Interrupción DSP.
	PDF del DSP.
	TC de DMA.

SEÑALES DEL BUS RTSI.

• Líneas de trigger.	6.
• Enlaces serie.	12 líneas serie que soportan 1 enlace full-dúplex con una velocidad de transferencia de 25 Mbits/seg cada una.

ENTRADAS ANALÓGICAS.

• Número de canales	dos.
• Impedancia de entrada.	450 KΩ en paralelo con 65 pF.
• Acoplamiento de entrada.	AC o DC.
• Resolución.	16 bits.
• Rango de la señal.	± 2.828 V (2 Vrms)

- Máxima tensión de entrada. ± 20 V (Con la tarjeta Apagada o encendida)
- Rango de ajuste de la ganancia. ± 3.5 % (± o.3 dB)
- Error de offset (después de la calibración) ± 15 LSB máximo, ±5LSB típico.
- Linealidad de fase ± 0.5 °, de DC hasta 20 KHz
- Fase entre canales. ± 1° , de DC a 20 KHz.
- Retardo de la señal. 35.6 periodos de muestreo, a cualquier velocidad de muestreo.
- Distorsión Armónica Total (THD) -95 dB para 0 dB de entrada, de DC a 22 KHz.
- Señal a THD+ruido. 90 dB para 0 dB de entrada, de DC a 22 KHz.
- Distorsión de intermodulación (IMD)

  A 48 Khz de velocidad de muestreo

  SMPTE (60 Hz, 7 Khz) -85 dB.

  DIN (250 Hz, 8 Khz) -85 dB.

  CCIF (14 Khz, 15 Khz) -95 dB.
- Rango dinámico

  (máxima relación S/N) 93 dB.
- Separación entre canales (Crosstalk) -85 dB, DC a 22 Khz.
- Ancho de Banda (-3dB) -0.45 veces la velocidad de muestreo.
- Frecuencias de muestreo. 4, 5.5125, 6, 6.4, 8, 11.025, 12, 12.8, 16, 22.05, 24, 25.6, 32, 44.1, 48, 51.2 KHz.

SALIDA ANALÓGICA.

- Número de canales — dos.
- Impedancia de salida. — 51.1 Ω.
- Acoplamiento de salida. — AC o DC.
- Impedancia de la carga recomendada — 10 kΩ o superior.
- Resolución — 16 bits.
- Rango de señal — ± 2.828 V (2 Vrms)
- Rango de ajuste de la ganancia — ± 6 % (± 0.5 dB).
- Error de offset (después de la calibra.) — ± 3 mV.
- Linealidad de fase. — ± 0.5°, DC a 20 Khz.
- Fase Intercanal. — ± 1° , de DC a 20 KHz.
- Retardo de la señal. — 34.6 ± 0.5 periodos de muestreo, a cualquier frecuencia de conversión.
- Señal THD+ruido

  (20 Hz a 20 Khz, 0dB de salida, a 48 kHz).

  80 Khz de ancho de banda. — -75 dB.

  20 Khz de ancho de banda. — -80 dB.

- Distorsión de intermodulación.

  A 48 Khz de velocidad de muestreo

  SMPTE (60 Hz, 7 Khz) — -82 dB.

  DIN (250 Hz, 8 Khz) — -82 dB.

  CCIF (14 Khz, 15 Khz) — -90 dB.

• Rango dinámico.	
(máxima relación S/N)	92 dB (48 kH de frecuencia de conversión)
• Separación entre canales (Crosstalk)	-85 dB, DC a 22 Khz.
• Ancho de Banda (-3dB)	-0.45 veces la velocidad de muestreo.
Frecuencias de muestreo.	4, 5.5125, 6, 6.4, 8, 11.025, 12, 12.8, 16, 22.05, 24, 25.6, 32, 44.1, 48, 51.2 KHz.

TRIGGER DIGITAL.

• Nivel de Entrada	Compatible TTL.
• Respuesta.	Programable por flanco de caída o subida.
• Máximo ancho de banda del pulso.	50 nseg.
• Nivel de salida.	Compatible TTL.
• Corriente de salida de nivel alto	-3.2 mA.
• Corriente de salida de nivel bajo	24.0 mA.

ANEXO II

```
/*PROGRAMA PRINCIPAL Y MENU*/

#include <conio.h>
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <malloc.h>
#include <graph.h>
#include <stdlib.h>
#include "colores.h"
#include <dos.h>
#include <io.h>
#include "menup.h"  /*fichero con las definiciones de los menus*/
#define TRUE 1
#define FALSE 0
#define SLOT 1
void desactivaMenu(int ); /*función que desactiva un menu*/
void ActivaMenu(int);     /* función que activa un menu*/
void leeraton(int *, int *, BOOLEAN *,BOOLEAN *);  /* función que lee la
                                                        posición del ratón */
void punteroon(void); /* función que activa el puntero del ratón */
void punteroff(void); /* función que desactiva el puntero del ratón */
void Menu(WORD *, BOOLEAN * );  /* función que recoje el resultado de la
                                            elección del menú */
void InicMenu(void);          /* función que inicia el menú */
BOOLEAN si_raton(void);       /* función que detecta si hay conectado un ratón */
void recuperabajoventana(int, int, char * ); /* función que repone lo que hay bajo una
                                                    ventana cuando esta se cierrra */
void BarraMenu(void);         /* función que pone la barra del menú */
void PoneBarra(int );
void BorraBarra(int);
```

```
void DesliegaMenu(int *, int *, int * ,int *, char *); /* función que despliega el menú
                                                                    */
void Genera(void);  /* función que genera un fichero */
void poneaspa(int, int, int, int); /* función que pone la aspas cuando seleccionamos
                                                                algo */
void borraaspa( int, int ,int, int); /* lo contrario de la anterior */
void rectangulo(void); /* función que dibuja un rectángulo */
void dibuja(void); /* función que dibuja un fichero en la pantalla */
void zoom(); /* función que hace un zoom */
void reproducir(long ,long ,int ); /* función que permite reproducir un fichero */
void borrar(void); /* función que permite borrar un trozo de fichero */
void copiar(void); /* función que permite borrar un trozo de fichero */
int cargar2(void); /* función que nos permite cargar un fichero */
void pantalla(void); /* función que dibuja la pantalla de presentación */
void repro_rapida(void); /* función que permite reproducir de forma rápida un
                                                                fichero*/
void gen_fiche(void); /* función que nos permite generar un fichero */
void adqisicion(void); /* función que nos permite adquirir un fichero */
void dsp_filtro(int ); /* función de filtrado */

union REGS regs; /* unión para los registros del ratón */

main(int argc, char *argv[])  /* procedimiento principal */
{
  int
      posix,y,i,resulCompar,Presentacion,x1=0,x2=639,y1,y2,s,col,k,
      CodigoPlaca,DireccionBase,Interrup1,Interrup2,IrqTrigMode,DMA1,
      DMA2,daqMode,NoPonePantall=1,orig=0;
  short
      resultado=0;
  long
      salvacolor,tama_panta,r,TamaImag,puntei,puntef;
```

```
char
    huge *buffer,tecla1,tecla2;
BOOLEAN
      encontrado,izda,dcha,Salir,llave, Amuleto;
WORD
    CodeRetorno;
float
    esc_x,escalax;
FILE
    *pf;

/*inicializa la AT-DSP2200 y obtiene el tipo*/
Get_DA_Brds_info(SLOT, CodigoPlaca,&DireccionBase,&Interrup1,&Interrup2,
                &IrqTrigMode,&DMA1,&DMA2,&daqMode); /*obtiene el tipo de
                                                          tarjeta*/
system("cls"); /* borra la pantalla haciendo uso de una ruina DOS */

if(CodigoPlaca==18)  /* comprueba si es una AT-DSP2200, si no es sale*/
{
    printf("\n\n\n\n\n\n\t\t\tINICIALIZANDO");
    Init_DA_Brds(SLOT,&CodigoPlaca); /*INICIALIZA LA TARJETA*/
}
else
{
  printf("--- ERROR: LA AT-DSP2200 NO ESTA CONECTADA!!!");
  exit(1);
}
resulCompar=strcmp(argv[1],"sp"); /* comprueba si el argumento es sp */
if((argc==2) && (resulCompar==0))
      Presentacion=FALSE;

if((argc==2) && (resulCompar!=0))
```

```
        {
         printf("\n\nEl argumento correcto es <sp> ");
         exit(1);
        }

  if(argc!=2)
      Presentacion=TRUE;

  if(_setvideomode(_MAXRESMODE)==0) /* inicializa el modo de video */
  {
      printf("No se puede cambiar a ese modo de video");
      exit(1);
  }

 if(_registerfonts("TMSRB.FON")<=0)  /*carga los tipos de letras*/
 {
      _setvideomode(3);
      printf("\nNo puedo cargar los tipos de letras");
      exit(0);
  }
 if(_registerfonts("COURB.FON")<=0)
    {
       _setvideomode(3);
      printf("\nNo puedo cargar los tipos de letras");
      exit(1);
    }
_setfont("t'Tms Rmn' h13w13b"); /*activa el tipo tmsrmn*/
 if(Presentacion==TRUE)
 pantalla(); /* llama a la pantalla de presentación */
  llave=FALSE;
  _setcolor(9);
  _rectangle(_GFILLINTERIOR,0,16,639,479); /*pone la pantalla azul*/
```

```
_setcolor(0);
if (si_raton())  /*si hay ratón inicia el menú */
  {
   InicMenu();
   punteroon(); /*activa el puntero*/
  }

do   /*bucle de espera que conforma el nucleo principal del programa*/
 {
   leeraton(&posix,&y,&izda,&dcha);
   if ((izda) && (y<12))
       {
        for(r=0;r<200000;r++) /*retardo*/
         {
          /* retardo al desplegar el menú */
         }
        Menu(&CodeRetorno,&Amuleto);   /* código de retorno que indica la
                                                              opción elegida */
        if (CodeRetorno!=0)
            switch (CodeRetorno)   /* selección de opciones */
                {
                   case 25: Salir=TRUE;break;
                   case 24:
                        llave=TRUE;
                         break;
                   case 41: repro_rapida();
                         CodeRetorno=0;
                         break;
                   case 42: acquisicion();
                         CodeRetorno=0;
                          break;
```

```
case 43: gen_fiche();
        CodeRetorno=0; /* reseteamos el codigo */
        break;
case 51:dsp_filtro(0);
        CodeRetorno=0;
        break;
 case 19: /*salir al DOS*/
        punteroff();
        if((pf=tmpfile())==NULL)
              {
               printf("Error al abrir el fichero");
               exit(1);
              }
        y1=0;
        y2=20;
        /* salva la pantalla en el disco duro */
        TamaImag= _imagesize(x1,y1,x2,y2);
        buffer= (char huge *)halloc(TamaImag,sizeof(char));
       for(s=0;s<480;s+=20)
            {
              _getimage(x1,s,x2,s+20,buffer);
              fwrite(buffer,sizeof(char),(int)TamàImag,pf);
            }
        _getimage(x1,459,x2,479,buffer);
       fwrite(buffer,sizeof(char),(int)TamaImag,pf);
       _setvideomode(3);
       printf("Pulse EXIT para volver al programa");
       /* ejecuta otro command.com */
       system("command.com");
        _setvideomode(_MAXRESMODE);
       rewind(pf);
      for(s=0;s<480;s+=20)
```

```
            {
              fread(buffer,sizeof(char),(int)TamaImag,pf);
              _putimage(0,s,buffer,_GPSET);
            }
        fread(buffer,sizeof(char),(int)TamaImag,pf);
        _putimage(0,459,buffer,_GPSET);
        hfree(buffer);
        fclose(pf);
        punteroon;
        CodeRetorno=0;
        break;
case 52:dsp_filtro(1);
        CodeRetorno=0;
        break;
case 53:dsp_filtro(2);
        CodeRetorno=0;
        break;
case 21:resultado=cargar2();
        CodeRetorno=0;
        if(resultado==0)
        DesMenu[1].Activo=TRUE; /* activa menú uno */
        break;
case 31:reproducir(puntei,puntef,orig);
        CodeRetorno=0;
        break;
case 34:zoom();
        CodeRetorno=0;
        break;
case 32:borrar();
        CodeRetorno=0;
        break;
case 33:copiar();
```

```
                        CodeRetorno=0;
                        break;
                    case 23: /* borrar */
                        punteroff();
                        _setcolor(9);
                        _rectangle(3,0,17,639,479);
                        desactivaMenu[1];
                        punteroon();
                        CodeRetorno=0;
                            /* borra el fichero de intercanbio */
                        remove("inter.fil");
                        break;
                } /*del case*/
            if (Amuleto)
                break;
        }

        if (llave) break;

 } /* fin del do */
while(1);
        punteroff();
       _unregisterfonts(); /* descarga los tipos de letra */
       _setvideomode(3); /* restaura el modo de video a modo texto */
} /* fin del procedimiento principal */

void InicMenu(void)     /*inicia menú */
{
  int i;
  char tecla,tecla2;
  char *buffer;
  long tam;
```

```
  _setcolor(7);
  _rectangle(_GFILLINTERIOR,0,0,639,15);  /*Barra de menu*/
  _setcolor(NEGRO);
   for(i=0;i<4;i++)
    {
      _moveto(DesMenu[i].XInin,0);
      _outgtext(DesMenu[i].Titulo);
    }
} /* fin de la función */


void  Menu(WORD *CodeRetorno, BOOLEAN *Amuleto)
{
 WORD
        posix,posiy,Tomar,LimiteSup;
 int
        i,j,AnchoMenu,M1X,M2X,M1Y,M2Y,SalvarColor,LimiteInf;
 BOOLEAN
        Encontrado,izda,dcha;
 char
        *BajoMenu,*BarraMovil,*BajoMenuS, *Amuleto=FALSE;

  SalvarColor=_getcolor();  /*salva el color de fondo*/
  _setcolor(15);
   leeraton(&posix,&posiy,&izda,&dcha);
    /* el ratón se encuentra en esta posición actúa */
   if ((posix>1) && (posix<13) && (posiy>0) && (posiy<10))
      {
        *Amuleto=TRUE;
        _setcolor(SalvarColor);
        _setvideomode(3);
```

```
        exit(1);
    }
    i=1;

 do /*pone los titulos del menu*/
  {
     i=i+1;
     if ((posix >= DesMenu[i].XInin) && (posix <= DesMenu[i].XFinal) &&
       (DesMenu[i].Activo))
        Encontrado=TRUE;
    else
         Encontrado=FALSE;
    }
while ((!Encontrado) && (i<4) && (strlen(DesMenu[i].Titulo)!=0));

 if (Encontrado)
    {
         punteroff();
         AnchoMenu=0;   /* calcula el tamaño del menu a desplegar en
                           función del conenido del mismo */
         for(j=0;j<((DesMenu[i].Elecs)-1);j++)
               if(strlen(DesMenu[i].ListaOpciones[j].Opcion)>AnchoMenu)
         AnchoMenu=strlen(DesMenu[i].ListaOpciones[j].Opcion);
         AnchoMenu=AnchoMenu*12;
         M1X=DesMenu[i].XInin;
         M1Y=14;
         M2X=DesMenu[i].XInin+AnchoMenu+6;
         M2Y=((DesMenu[i].Elecs+1) * 17);

          /* espacio para el rectangulo del menu */
          DesMenu[i].TamanoMenu=_imagesize(M1X,M1Y,M2X,M2Y);
          BajoMenu=(char *) malloc((DesMenu[i].TamanoMenu));
```

```
_getimage(M1X,M1Y,M2X,M2Y,BajoMenu);

/* espacio para el rectangulo de la sombra */
DesMenu[i].TamanoMenu=_imagesize(M1X+5,M1Y+5,M2X+5,M2Y+5);
BajoMenuS=(char *) malloc((DesMenu[i].TamanoMenu));
_getimage(M1X+5,M1Y+5,M2X+5,M2Y+5,BajoMenuS);
 _setcolor(0); /* color de la sombra */
 _rectangle(_GFILLINTERIOR,M1X+5,M1Y+5,M2X+5,M2Y+5);
 _setcolor(0); /* color del rectangulo del menu */
 _rectangle(_GFILLINTERIOR,M1X,M1Y,M2X,M2Y);
    /* color de la barra */
 BarraMovil=(char *) malloc((int)_imagesize(M1X+1,M1Y+1,M2X-
                                                1,M1Y+17));
 _setcolor(7);
 _rectangle(_GFILLINTERIOR,M1X+1,M1Y+1,M2X-1,M1Y+17);
 _getimage(M1X+1,M1Y+1,M2X-1,M1Y+17,BarraMovil);

if(DesMenu[i].ptr==NULL) /* comprueba si estan activos */
    {
      _rectangle(_GBORDER,M1X,M1Y,M2X,M2Y);
      _setcolor(0);
     if (DesMenu[i].ListaOpciones[0].OpcionActiva)
       {
         _moveto(DesMenu[i].XInin+3,15);
         _outgtext(DesMenu[i].ListaOpciones[0].Opcion);
       }
       _setcolor(7);
       for(j=1;j<((DesMenu[i].Elecs));j++)
           if (DesMenu[i].ListaOpciones[j].OpcionActiva)
             {
               _moveto(DesMenu[i].XInin+3,15+(j*17));
               _outgtext(DesMenu[i].ListaOpciones[j].Opcion);
```

```
            }
        DesMenu[i].ptr=(char *) malloc(DesMenu[i].TamanoMenu);
        _getimage(M1X,M1Y,M2X,M2Y,DesMenu[i].ptr);
    }
    /* traer la imagen del menu desde la memoria */
    _putimage(M1X,M1Y,DesMenu[i].ptr,_GPSET);
    punteroon(); /* puntero sobre la barra movil */
/* bucle que mueve la barra */
LimiteSup=15;
LimiteInf=32;
Tomar=0;

do
    {
    leeraton(&posix,&posiy,&izda,&dcha);

    /* si el puntero abandona el menu salir */
    /* lo mismo que pulsar el boton derecho */
    if ((posix<M1X) || (posix>M2X) || (posiy >M2Y))
        break;
    else
     {
      if(posiy<LimiteSup)   /*barra hacia arriba*/
        if (posiy>17)    /* si todavia no ha llegado al tope */
            {
             punteroff();
             /*si opcion activa borrar barra */
             if( DesMenu[0].ListaOpciones[Tomar].OpcionActiva)

             _putimage(M1X+1,LimiteSup,BarraMovil,_GXOR);

              /* decrementar los limites y elegir el numero */
```

```
                    LimiteSup=LimiteSup-17;
                    LimiteInf=LimiteInf-17;
                    Tomar=Tomar-1;
                  /* si opcion activa mostrar barra en nueva posicion */

                    if (DesMenu[i].ListaOpciones[Tomar].OpcionActiva)
                        _putimage(M1X+1,LimiteSup,BarraMovil,_GXOR);
                    punteroon();
                  }
                  if ((posiy>LimiteInf) && (posiy<=(17*DesMenu[i].Elecs)))
                    {
                     punteroff();
                /* si opcion activa borra barra en la posicion actual */
                     if (DesMenu[i].ListaOpciones[Tomar].OpcionActiva)
                         _putimage(M1X+1,LimiteSup,BarraMovil,_GXOR);

                     /* incrementar limites y elegir numero */
                     LimiteSup=LimiteSup+17;
                     LimiteInf=LimiteInf+17;
                     Tomar=Tomar+1;

                   /* si opcion activa mostrar barra en nueva posicion */
                   if (DesMenu[i].ListaOpciones[Tomar].OpcionActiva)
                    _putimage(M1X+1,LimiteSup,BarraMovil,_GXOR);
                    punteroon();
              }
          }
      if(kbhit() && getch()==27) break;
      }
while (!izda);
```

```
        recuperabajoventana(M1X,M1Y,BajoMenu);
        punteroff();
        _putimage(M1X+5,M1Y+5,BajoMenuS,_GPSET);
        punteroon();
        free(BajoMenu);
        free(BajoMenuS);

                /* Inicializamos el codigo de retorno de la funcion */
                /* EL boton derecho indica siempre "escape", es decir */
                /* No elegir ninguna accion.Elegir una opcion inactiva */
                /* tambien da como resultado un cero.Una opcion activa */
                /* devuelve como resultado de la funcion el codigo de la
                opcion */
        if (izda)
            *CodeRetorno=DesMenu[i].ListaOpciones[Tomar].CodOpcion;
         else
            *CodeRetorno=0;
        punteroon();
    }
     _setcolor(SalvarColor);
    free(BarraMovil);
} /* fin */

void BarraMenu(void)
{
     char tecla,tecla1;
    int i=0;
     punteroff();/* apagamos raton */
    PoneBarra(i);
    getch();
```

```
 do
   {
          tecla=getch();
          switch(tecla)
            {
              case 77:
                    BorraBarra(i);
                    i++;
                    if (i>3)
                    i=0;
                    PoneBarra(i);
                    break;

            case 75:
                    BorraBarra(i);
                    i--;
                    if(i==1)
                    i=3;
                    PoneBarra(i);
                    break;
            }

     }
  while ((tecla != 13) && (tecla !=27));
  BorraBarra(i);
  punteroon();
}


void BorraBarra(int y) /* Función que borra la barra*/
{
  _setcolor(7);
```

```
  _rectangle(_GFILLINTERIOR,DesMenu[i].XInin,0,DesMenu[i].XFinal,14);
  _setcolor(0);
  _moveto(DesMenu[i].XInin,0);
  _outgtext(DesMenu[i].Titulo);
}
void PoneBarra(int i)
{
  _setcolor(0); /*nuevo rectangulo*/;
  _rectangle(_GFILLINTERIOR,DesMenu[i].XInin,0,DesMenu[i].XFinal,14);
  _setcolor(7);
  _moveto(DesMenu[i].XInin,0);
  _outgtext(DesMenu[i].Titulo);
}
void leeraton(int *XX,int *YY,BOOLEAN *IZDA,BOOLEAN *DCHA)
{
 union REGS regs;

    regs.x.ax=0x03;
    int86(0x33,&regs,&regs);
    *XX=regs.x.cx;
    *YY=regs.x.dx;
    if( regs.x.bx==1)
         *IZDA=TRUE;
        else
          *IZDA=FALSE;
     if(regs.x.bx==2)
         *DCHA=TRUE;
        else
          *DCHA=FALSE;

}
void punteroon(void)
```

```
{

   regs.x.ax=0x01;
   int86(0x33,&regs,&regs);
}
void punteroff(void)
{
  regs.x.ax=0x02;
  int86(0x33,&regs,&regs);
}

BOOLEAN si_raton(void)
{
   regs.x.ax=0x00;
   int86(0x33,&regs,&regs);
   if(regs.x.ax=0xffff)
     {
        return(TRUE);
     }
   else
        return(FALSE);

}
void recuperabajoventana(int M1X,int M1Y, char *BajoMenu)
{
 punteroff();
 _putimage(M1X,M1Y,BajoMenu,_GPSET);
 punteroon();
}

void desactivaMenu(int numenu)
{
```

```
DesMenu[numenu].Activo=FALSE;
}
void ActivaMenu(int numen)
{
DesMenu[numen].Activo=TRUE;
}
```

```
/* EDITOR DE FICHEROS */
typedef unsigned char BOOLEAN;

#include <stdio.h>
#include <stdlib.h>
#include "colores.h"
#include <graph.h>
#include <malloc.h>
#include <dos.h>
#include <ctype.h>
#include <direct.h>
#include <bios.h>

#define TAB 15
#define RELLENO 3
#define HUECO 2
#define MAX 20
#define NO 0
#define SI 1
#define PAG_ARRIBA  73
#define PAG_ABAJO 81
#define SI 1
#define NO 0
#define F1 59

/* función que permite escribir en modo gráfico*/
char *escribir(int x1,int y1,int x2,int y2,int largo);

/* función que lee la posición del ratón */
void leeraton(int *, int *, BOOLEAN *,BOOLEAN *);

/* función que dibuja la carátula del cargar de de ficheros */
```

```
void DibujaCaratula(int, int, int, int);

/* función que dibuja un boton */
void boton(int, int ,int, int, char *,BOOLEAN);

/* función que dibuja el fichero en pantalla */
int dibuja(char *);

int cargar2(void)
{
 int
x1=50,y1=50,x2=320,y2=300,contador=0,nu_linea=0,SalirYCargar=NO,
directorios=0,fich=0,px,py,SalirSinCargar=NO,EsDSP,NumeroLineas=10,cfree=0;

 BOOLEAN
      ida,dha;
 long
      TamaImage;
 unsigned int
            error;
 unsigned
      memoria;
 char
      *cadena,*tamano;
 typedef struct find_t *p_ffblk; /* estructura de las varibles de directorio */
 char
direc_activo[_MAX_DIR],direc_original[_MAX_DIR],completa[_MAX_PATH];

 char *DireRaiz = "c:\\..\\";
 FILE *pf;
 char
  cambio, fin, aceptable;
```

```
int
  i, j, k=0, car, m,  max_lineas = 10, activa,incr=0,inicio=0,pagina=0,l;
long
  capacidad;
struct find_t
     fichero;
p_ffblk
      *linea,  *aux, inter;
static char huge *SalvaPantalla=NULL;

 _setfont("t 'courier',h13w10bf");  /* activa la letra courrier */

/*puntero para salvar el trozo de pantalla donde dibujaremos la carátula*/
 TamaImage=_imagesize(x1,y1,x2,y2);  /*TAMAÑO DE LA IMAGEN*/
 if((SalvaPantalla=(char huge *) halloc(TamaImage,sizeof(char)))==NULL) /*CREA
                                        UN BUFFER PARA SALVAR LA IMAGEN*/
  {
    _setvideomode(3);
    printf("NO HAY SUFICIENTE MEMORIA REINICIALICE EL PC");
    exit(1);
  }
 _getimage(x1,y1,x2,y2,SalvaPantalla);
 DibujaCaratula(x1, y1, x2, y2);
_setcolor(NEGRO);
_rectangle(HUECO,x1+10,y1+218,x2-5,y1+238);
_setcolor(AZULCLARO);
_rectangle(RELLENO,x1+11,y1+219,x2-6,y1+237);
if(getcwd(direc_original,_MAX_DIR)==NULL)  /*GUARDA EL DIRECTORIO
                                                    ACTUAL*/
  {
     printf("ERROR CAMBIANDO DIRECTORIO");
     exit(1);
```

```
    }
      _setcolor(NEGRO);
      _moveto(x1+12,y1+220);
      _outgtext(direc_original);

/* asigna memoria para la tabla de punteros*/
     if((linea=(p_ffblk  *)malloc(NumeroLineas*sizeof(p_ffblk)))==NULL)
        {
          printf("INPOSIBLE ASIGNAR MEMORIA");
          exit(1);
        }
      i=0;
     _moveto(x1+12,y1+21);
     _outgtext("*.*"); /* buscar todos */

     do
       {
         /* búsqueda del primer fichero */
         j= _dos_findfirst("*.*",_A_SUBDIR |_A_ARCH,&fichero);
         if(j!=0)
           cadena=escribir(x1+11,y1+21,x1+219,y1+39,MAX);
         }
     while(j!=0); /* hasta que lo encuentra */
    for(m=0;m<strlen(cadena);m++)
       cadena[m]=toupper(cadena[m]); /* lo pone en mayúsculas */
    do
      {
       if(j==0)
         do
          {
           if(i==max_lineas)
               {
```

```
        max_lineas +=10; /*numero de lineas que hay en pantalla*/
         /* si hay más de 10 ficheros reasigna el tamaño de la tabla */
if((aux =(p_ffblk  *)realloc (linea,max_lineas * sizeof(p_ffblk)))==NULL)
        {
          printf("INPOSIBLE ASIGNAR MEMORIA");
          exit(1);
         }
      linea=aux;
  }

if((linea[i] =(p_ffblk) malloc (sizeof(struct find_t)))==NULL)
        {
          printf("INPOSIBLE ASIGNAR MEMORIA");
          exit(1);
         }
  *linea[i++]=fichero;
  j=_dos_findnext(&fichero); /*búsca el resto de los ficheros si los hay*/
  if(j!=0)
       {
        _setcolor(AZULCLARO);
        _rectangle(RELLENO,x1+11,y1+71,+x1+170,y1+209);
        for(k;k<i;k++,contador++)
            {
             if(linea[k]->attrib & _A_SUBDIR) /* si es un directorio va en
                                                          rojo*/
                 {
                   _setcolor(ROJO);
                   _moveto(x1+12,y1+72+incr);
                   _outgtext(linea[k]->name);
                   _moveto(x1+12+100,y1+72+incr);
                    _outgtext("<DIR>");
                 }
```

```
                    else /* son ficheros */
                    {
                      _setcolor(NEGRO);
                      _moveto(x1+12,y1+72+incr);
                      _outgtext(linea[k]->name);
          }
          incr+=13;
          if(contador==9)
                break;
      }/*del for*/
          incr=0;
          contador=0;
          k++;

   }/* del if(j==0) */
  }/* del do */
while(j==0);
leeraton(&px,&py,&ida,&dha); /* obtiene la posición del ratón */

if(j!=0);
 {
   if (kbhit()) /* si se pulsan teclas que tienen doble código */
     {
      car=getch();
      switch(car)
     }
    case 0:car=getch();
         switch(car)
         {
         case PAG_ABAJO: /* si se pulsa PAG_ABAJO*/
      if(k<i)
        {
```

```
      pagina++;
      _setcolor(AZULCLARO);
      _rectangle(RELLENO,x1+11,y1+71,+x1+179,y1+209);
      for(k;k<i;k++,contador++)
       {
         if(linea[k]->attrib & _A_SUBDIR)
         {
          _setcolor(ROJO);
          _moveto(x1+12,y1+72+incr);
          _outgtext(linea[k]->name);
          _moveto(x1+12+100,y1+72+incr);
          _outgtext("<DIR>");
         }
        else
         {
          _setcolor(NEGRO);
          _moveto(x1+12,y1+72+incr);
          _outgtext(linea[k]->name);
         }
       incr+=13;
       if(contador==9)
        {
         break;
        }
       }/*del for*/
     incr=0;
     contador=0;
     k++;
   }/*del if<k)*/
  break;
case PAG_ARRIBA:
   if(k>0)
```

```
{
 _setcolor(AZULCLARO);
 _rectangle(RELLENO,x1+11,y1+71,+x1+179,y1+209);
 if(k % 10 ==0)
    k=k-20;
 if(k % 10 !=0)
    k=k-(10+(k % 10));
  if(k<0)
    k=0;
 for(k;k<i;k++,contador++)
   {
   if(linea[k]->attrib & _A_SUBDIR)
     {
      _setcolor(ROJO);
      _moveto(x1+12,y1+72+incr);
      _outgtext(linea[k]->name);
      _moveto(x1+12+100,y1+72+incr);
      _outgtext("<DIR>");
     }
   else
     {
      _setcolor(NEGRO);
      _moveto(x1+12,y1+72+incr);
      _outgtext(linea[k]->name);
     }
   incr+=13;
   if(contador==9)
     {
     break;
     }
  }/*del for*/
 incr=0;
```

```
        contador=0;
        k++;
      } /*del if*/
      break;
  case TAB:
      _setcolor(ROJO);
      _rectangle(HUECO,x1+10,y1+20,x1+220,y1+40);

      for(l=0;l<i;l++)
      {
       cfree++;
       free(linea[l]);
      }
       cfree=0;
      cadena=escribir(x1+11,y1+21,x1+219,y1+39,MAX);
      _setcolor(NEGRO);
      _rectangle(HUECO,x1+10,y1+20,x1+220,y1+40);
      i=0;
      j= _dos_findfirst(cadena,_A_SUBDIR ,&fichero);
      k=0;
      for(m=0;m<strlen(cadena);m++)  /*tranforma las minúsculas en mayúsculas
                                                                    */
      cadena[m]=toupper(cadena[m]);
      break;

case F1:  /* si se pulsa F1 */
      _setcolor(ROJO);
      _rectangle(HUECO,x1+10,y1+218,x2-5,y1+238);
       cadena=escribir(x1+11,y1+220,x2-6,y1+237,MAX);
      _setcolor(AZULCLARO);
      _rectangle(RELLENO,x1+11,y1+219,x2-6,y1+237);
      _setcolor(NEGRO);
```

```
 _rectangle(HUECO,x1+10,y1+218,x2-5,y1+238);

 if(_fullpath(completa,cadena,_MAX_PATH)!=NULL)
 {
  chdir(completa);
  if(getcwd(direc_activo,_MAX_DIR)==NULL)
    {
       printf("ERROR CAMBIANDO DIRECTORIO");
        exit(1);
    }
 if(strlen(direc_activo) >21)
  {
   strcat(DireRaiz,cadena);
   _setcolor(NEGRO);
   _moveto(x1+12,y1+220);
   _outgtext(DireRaiz);
  }
 else
  {
  _setcolor(NEGRO);
  _moveto(x1+12,y1+220);
  _outgtext(direc_activo);
  }
 }
else
 printf("ERROR");
 _setcolor(AZULCLARO);
 _rectangle(RELLENO,x1+11,y1+21,x1+219,y1+39);
 _setcolor(NEGRO);
 _moveto(x1+12,y1+21);
 _outgtext("*.*");
for(l=0;l<i;l++)
```

```
        {
         free(linea[l]);
          }
        i=0;
        j= _dos_findfirst("*.*",_A_SUBDIR ,&fichero);
        k=0;
        for(m=0;m<strlen(cadena);m++)
        cadena[m]=toupper(cadena[m]);
         break;
    }

   }/*del switch*/
  }
 } /*del if*/

for(l=0;l<i;l++)
{
  if((strcmp(cadena,linea[l]->name)==0) && (linea[l]->size>1024))/*!=0*/
  {
   SalirYCargar=SI;
  }
 else
   SalirYCargar=NO;
}
if((px > x1+10) && (px < x2-6))
 if ((py >y1+218) && (py < y1+238) && (ida))
 {
        punteroff();
        _setcolor(ROJO);
        _rectangle(HUECO,x1+10,y1+218,x2-5,y1+238);
        cadena=escribir(x1+11,y1+220,x2-6,y1+237,MAX);
        _setcolor(AZULCLARO);
```

```
 _rectangle(RELLENO,x1+11,y1+219,x2-6,y1+237);
 _setcolor(NEGRO);
 _rectangle(HUECO,x1+10,y1+218,x2-5,y1+238);
 if(_fullpath(completa,cadena,_MAX_PATH)!=NULL)
 {
  chdir(completa);
  if(getcwd(direc_activo,_MAX_DIR)==NULL)
    {
       printf("ERROR CAMBIANDO DIRECTORIO");
        exit(1);
    }
 if(strlen(direc_activo) >21)
  {
   strcat(DireRaiz,cadena);
   _setcolor(NEGRO);
   _moveto(x1+12,y1+220);
   _outgtext(DireRaiz);
  }
 else
  {
  _setcolor(NEGRO);
  _moveto(x1+12,y1+220);
  _outgtext(direc_activo);
  }
 }
else
 printf("ERROR");
 _setcolor(AZULCLARO);
 _rectangle(RELLENO,x1+11,y1+21,x1+219,y1+39);
 _setcolor(NEGRO);
 _moveto(x1+12,y1+21);
 _outgtext("*.*");
```

```
        for(l=0;l<i;l++)
          free(linea[l]);
        i=0;
        j= _dos_findfirst("*.*",_A_SUBDIR ,&fichero);
        k=0;
        for(m=0;m<strlen(cadena);m++)
            cadena[m]=toupper(cadena[m]);
        punteroon();
 }
if((px >x1+10) && (px < x1+220))
 if ((py >y1+20) && (py < y1+40) && (ida))
  {
        punteroff();
        _setcolor(ROJO);
        _rectangle(HUECO,x1+10,y1+20,x1+220,y1+40);
        for(l=0;l<i;l++)
          free(linea[l]);
        cadena=escribir(x1+11,y1+21,x1+219,y1+39,MAX);
        _setcolor(NEGRO);
        _rectangle(HUECO,x1+10,y1+20,x1+220,y1+40);
        i=0;
        j= _dos_findfirst(cadena,_A_SUBDIR ,&fichero);
        k=0;
        for(m=0;m<strlen(cadena);m++)
        cadena[m]=toupper(cadena[m]);
        punteroon();
  }
  if((px >x1+185) && (px < x1+294))
    if ((py >y1+120) && (py < y1+140) && (ida))
    {
 while(ida)
```

```
        {
          leeraton(&px,&py,&ida,&dha);
          _setfont("t'Tms Rmn' h13w13b");
          boton(x1+197, y1+120 ,x1+260,y1+140 ,"SALIR",NO);
          _setfont("t'courier' h13w13bf");
        }

          _setfont("t'Tms Rmn' h13w13b");
          boton(x1+197, y1+120 ,x1+260,y1+140 ,"SALIR",SI);
          _setfont("t'courier' h13w13bf");
          SalirSinCargar=SI;
          /*break;*/
       }

} /* do*/

while((SalirYCargar==NO) && (SalirSinCargar==NO));
 chdir(direc_original); /* repone le directorio original */
 punteroff(); /* desactiva el puntero del ratón */
 _putimage(x1,y1,SalvaPantalla,_GPSET); /* pone la pantalla salvada */
 punteroon(); /* activa el puntero del ratón */
 hfree(SalvaPantalla); /* libera el buufer de memoria */
 for(l=0;l<i;l++)
   free(linea[l]);
 free(linea);

 _setfont("t'Tms Rmn' h13w13b"); /*ponemos el tipo de leta Tms Rmn */

if(SalirYCargar==SI)
{
   if((pf=fopen("inter.fil","w+b"))==NULL)
    {
```

```
        printf("Error no se puede abrir el fichero");
        exit(1);
       }
      fwrite(cadena,sizeof(cadena),3,pf);
      fclose(pf);
      EsDSP=dibuja(cadena);  /* si el fichero es del tipo DSP lo dibuja */
      if(EsDSP==0)
        return(0); /* activar menu de edición */
      else
        return(1); /* no activar menu edición*/
   }
   if(SalirSinCargar==SI)
   {
     return(1); /* no activar menu de edición*/
   }
}
```

```
/* PROCEDIMIENTO QUE DIBUJA EL FICHERO Y PONE LAS PANTALLAS*/

#include <stdio.h>
#include <graph.h>
#include <dos.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
#include "colores.h"
#define TAMA 1024

void PoneEscala(int);
int dibuja(char * nomfile)
{
 float
        LongSegundos,Periodo;
 unsigned long
             tafi,pts_to_plot,xcount,tama;
 long
       max=0,min=0,pui,puf,punf,pufa,puia;
 int
       fh,x1,x2,conta=0,tecla,decimal,signo,NoesDSP=0;
 static int
         dato[TAMA];
 double
         x_scale,y_scale;
 unsigned int
             x,y,j,i;
struct
{
 int nu_canales;
```

```
 unsigned int fm;
}cabecera;
char Buffer[15];
struct videoconfig vconfig;
 FILE *pf0;
 punteroff();
 _getvideoconfig(&vconfig);
  titulos(1,21,640,450,NEGRO);
  titulos(2,23,638,35,AZULCLARO);
  PoneCadena(210,21,NEGRO,"VISOR DE FICHEROS");
  titulos(2,40,638,55,AZULCLARO);
  PoneCadena(6,40,NEGRO,"CANAL DERECHO");
  titulos(2,185,638,200,AZULCLARO);
  titulos(1,345,640,450,GRIS);
  titulos(4,455,635,475,AZULCLARO);
  PoneEscala(179);
if((fh=open(nomfile,O_RDONLY,O_BINARY))<0)
{
   printf("No se Puede abrir el fichero");
   exit(1);
}
tafi=filelength(fh); /* logitud del fichero */
pts_to_plot=tafi/2;
close(fh);


if((pf0=fopen(nomfile,"r+b"))==NULL)
{
 printf("No se Puede abrir el fichero");
 exit(1);
}
```

```
fread(&cabecera,sizeof(cabecera),1,pf0);   /*lee la cabecera*/

if((cabecera.nu_canales==0) || (cabecera.nu_canales>2) || (cabecera.nu_canales<0))
  {
   NoesDSP=1;
   PoneCadena(5,457,NEGRO,"EL FICHERO NO TIENE EL FORMATO
                                        ADECUADO. <PULSE UNA TECLA>");
   getch();
   _setcolor(AZULCLARO);
   _rectangle(3,0,16,639,479);
   punteroon();
   return(NoesDSP);
  }
LongSegundos=(float)((float)((float)1/cabecera.fm)*(float)tafi)/((float)cabecera.nu_c
                                                              anales*(float)2);

    if(cabecera.nu_canales==2)
    PoneCadena(6,184,NEGRO,"CANAL IZQUIERDO");
    titulos(1,345,640,360,GRIS);
    PoneCadena(6,345,AMARILLOCLARO,"NOMBRE DEL FICHERO:");
    PoneCadena(226,345,VERDE,nomfile);
    titulos(1,370,640,385,GRIS);
    PoneCadena(6,370,AMARILLOCLARO,"NUMERO CANALES:");
    itoa(cabecera.nu_canales,Buffer,10);
    PoneCadena(185,370,VERDE,Buffer);
    PoneCadena(210,370,AMARILLOCLARO,"DURACION EN SEGUNDOS:");
    gcvt((double)LongSegundos,5,Buffer);
    strcat(Buffer," Segundos");
    PoneCadena(462,370,VERDE,Buffer);
    titulos(1,395,640,410,GRIS);
    PoneCadena(6,395,AMARILLOCLARO,"TAMA¥O DEL FICHERO:");
```

```
ltoa((long)tafi,Buffer,10);
strcat(Buffer," Bytes");
PoneCadena(230,395,VERDE,Buffer);
titulos(1,420,640,435,GRIS);
PoneCadena(6,420,AMARILLOCLARO,"FRECUENCIA DE MUESTREO:");
ultoa((unsigned long)cabecera.fm,Buffer,10);
strcat(Buffer," Hz");
PoneCadena(265,420,VERDE,Buffer);

/*encuentra el maximo y el minimo*/
while(fread(&dato,sizeof(int),TAMA,pf0)!=0)
  {
   for(i=0;i<TAMA;i++)
     {
      if(dato[i]<min) min=dato[i];
      if(dato[i]>max) max=dato[i];
     }
  }
fseek(pf0,4L,0);/*situamos el puntero despues de la cabecera*/

/*define las escalas x e y*/
if(cabecera.nu_canales==2)
    x_scale=(double)(((2*vconfig.numxpixels))-2)/(double)(pts_to_plot);
else
   x_scale=(double)((vconfig.numxpixels)-2)/(double)(pts_to_plot);

y_scale=(double)((vconfig.numypixels/2)-150)/(double)(max-min);
_setlogorg(1,42);
_moveto(0,0);
```

```
for(i=0;i<cabecera.nu_canales;i++)
        {
         fread(&dato,sizeof(int),1,pf0);
         x=xcount=0; /*contadores a cero*/
         y=(int)((dato[i]-min)*y_scale);
         y=((vconfig.numypixels/2)-150)-y;
         _setcolor(4+(int)i);  /*color del dibujo*/
         _moveto(x,y+35);
          fseek(pf0,4L,0);
         while(fread(&dato,sizeof(int),TAMA,pf0)!=0)
              {
                   for(j=i+cabecera.nu_canales;j<TAMA;j+=cabecera.nu_canales)
                      {
                        x=(int)((double)(++xcount)*x_scale);
                        y=(int)((dato[j]-min)*y_scale);
                        y=((vconfig.numypixels/2)-150)-y;
                        _lineto(x,y+35);
                       }
               }

    if(cabecera.nu_canales==2)
       {
        fseek(pf0,4L,0);
        _setlogorg(1,185); /*185*/
       }
 }
     fclose(pf0);
    _setlogorg(0,0);
    _setfont("t'Tms Rmn' h8w8b");
```

```
        gcvt((((float)2.828*(float)max)/(float)32768),4,Buffer);
        strcat(Buffer," Volts");
        PoneCadena(2,61,VERDE,Buffer);
        if(cabecera.nu_canales==2)
          PoneCadena(2,206,VERDE,Buffer);
        gcvt((((float)2.828*(float)min)/(float)32768),4,Buffer);
        strcat(Buffer," Volts");
        PoneCadena(2,168,VERDE,Buffer);
        if(cabecera.nu_canales==2)
              {
              PoneCadena(2,328,VERDE,Buffer);
              PoneEscala(340);
              }
         _setfont("t'Tms Rmn' h13w13b");

 /*} del if no es dsp*/
 punteroon();
 return(NoesDSP);
}
void PoneEscala(int CorY) /* pone las escalas */
{
  _setcolor(VERDE);
  _moveto(0,CorY);
  _lineto(638,CorY);
  _moveto(1,CorY);
  _lineto(1,CorY-5);
  _moveto(320,CorY);
  _lineto(320,CorY-5);
  _moveto(638,CorY);
  _lineto(638,CorY-5);
}
```

```
/*PROCEDIMEINTO QUE PERMITE OIR UN TROZO DE FICHERO
SELECCIONADO*/
typedef unsigned int WORD;
typedef unsigned char BOOLEAN;

#include <graph.h>
#include <fcntl.h>
#include <stdio.h>
#include "nidaq.h"
#include <stdlib.h>
#include <malloc.h>
#include <ctype.h>
#include <io.h>
#include <dos.h>
#include "colores.h"
#define RELLENO 3
#define TRUE 1
#define FALSE 0
extern void ErrPrint();  /* definición de códigos de error */
void Cuadrado(int, int,int, int,int,int); /* función que dibuja un cuadrado */

/* función que nos permite indicar el número de iteraciones */
unsigned long NumeroIteraciones(void);
void DibujaFlecha(int, int, int,int); /* función que dibuja un aflecha */
void boton(int,int,int,int,char * ,BOOLEAN); /* función que dibuja un botón */
void leeraton(int *, int *, BOOLEAN *,BOOLEAN *);

void reproducir(unsigned long puiz,unsigned long pufz,int origen)
{
 int
      estado, fh,x1,x2,chanVect[2],codigo,fila1,columna,fila2;
 unsigned long
```

```
            pts_to_plot,tafi,pui,puf,iteraciones;
double
      x_scale,rate;
FILE
      *pfi;
char
     fichero[13];

struct /* estructura de la cabecera del fichero DSP*/
{int nu_canales;
unsigned int fm;
}cabecera;

Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
Ponecadena(5,457,NEGRO,"ESPERE, POR FAVOR");
estado=Init_DA_Brds(1,&codigo);
ErrPrint("Init_DA_Brds",estado);
iteraciones=1;
Cuadrado(RELLENO,5,455,634,475,AZULCLARO);
 if((pfi=fopen("inter.fil","r+b"))==NULL)  /* abre el fichero de intercanbio*/
  {
   printf("EL FIECHERO NO EXISTE");
   exit(1);
  }
  fread(&fichero,sizeof(fichero),1,pfi);
  fclose(pfi);
 if((fh=open(fichero,O_RDONLY,O_BINARY))<0)
 {
     printf("No se puede abrir el fichero");
     exit(1);
  }
 tafi=filelength(fh);  /* calcula la longitud del fichero */
```

```
  pts_to_plot=tafi/2;
  read(fh,&cabecera,4); /* lee la cabecera */
   rate=(double)cabecera.fm;
  close(fh);
  x_scale=(double)(640)/(double)(pts_to_plot); /* cácula la escala del eje X */
  if(origen==0) /* 0= Se llama del menu principal*/
    {
     Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
     Ponecadena(5,457,NEGRO,"MARQUE LA ZONA A REPRODUCIR");
     marcar(&x1,&x2,fichero); /* llama al procedimiento marcar */
    if(x1==x2)
       exit(1);
     pui=(long)(x1/x_scale);
     puf=(long)(x2/x_scale);
     if(pui<4L)
        {
          pui=4L;
        }
  }

if(origen==1) /*se llama desde el ZOOM*/
       {
        pui=puiz;
        puf=pufz;
       }

  if((pui % 2)==0)
      {
        pui=pui+1;
       }
  else
     pui=pui+1;
```

```
 if(cabecera.nu_canales==1)
 chanVect[0]=0;
 if(cabecera.nu_canales==2)
 {
  chanVect[0]=0;
  chanVect[1]=1;
 }
 if((pui % 2==0) && (puf % 2==0)) /*ambos pares*/
      pui=pui+1;
 if((pui % 2 !=0) && (puf % 2 !=0 )) /*ambos impares*/
     pui=pui+1;

 iteraciones=NumeroIteraciones();

 estado=WFM_from_Disk(1,cabecera.nu_canales,chanVect,fichero,(unsigned
long)pui,(unsigned long)puf,(unsigned long)iteraciones,rate);

ErrPrint("WFM_fron_Disk",estado);
estado=Init_DA_Brds(1,&codigo);
ErrPrint("Init_DA_Brds",estado);

/* borra lo marcado si hemos hecho la llamada desde oir */
  Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
      if(origen==0)
        {
         punteroff();
         for(fila1=76;fila1<169;fila1++)
         for(columna=x1;columna<x2;columna++)
           {
            if(_getpixel(columna,fila1)==CIAN)
               {
                 _setcolor(NEGRO);
```

```
                    _setpixel(columna,fila1);
                  }
              else
                 if (_getpixel(columna,fila1)==ROJO)
                    {
                     _setcolor(ROJO);
                     _setpixel(columna,fila1);
                    }
     }
    if(cabecera.nu_canales==2)
              {
               for(fila2=220;fila2<325;fila2++)
                 for(columna=x1;columna<x2;columna++)
                    {
                     if(_getpixel(columna,fila2)==CIAN)
                       {
                        _setcolor(NEGRO);
                        _setpixel(columna,fila2);
                        }
              else
                 if (_getpixel(columna,fila2)==MAGNETA)
                    {
                     _setcolor(MAGNETA);
                     _setpixel(columna,fila2);
                    }
        }
   }
    punteroon();
  }
}
unsigned long NumeroIteraciones(void)
{
```

```
int es_x1=200,es_x2=370,es_y1=100,es_y2=175,codigo,estado,contador=0;
static char huge *BufferImagen=NULL;
long TamaPantalla,i;
unsigned long Itera;
 char *tabla[10]={"1","2","3","4","5","6","7","8","9","10"}; /* máximo número de
                                                                iteraciones */
 int posix,posiy,indice=0,NumeroIteraciones;
 BOOLEAN dcha,izda,oir=FALSE;
TamaPantalla=_imagesize(es_x1-5,es_y1-5,es_x2+5,es_y2+5);
BufferImagen=(char huge *) halloc(TamaPantalla,sizeof(char));
if(BufferImagen==NULL)
{
     estado=Init_DA_Brds(1,&codigo);
     ErrPrint("Init_DA_Brds",estado);
     _setvideomode(3);
     printf("HA OCURRIDO UN ERROR EN LA APLICACION");
     exit(1);
 }

 _getimage(es_x1-5,es_y1-5,es_x2+5,es_y2+5,BufferImagen);
 punteroff();
 titulos(es_x1,es_y1,es_x2,es_y2,BLANCO);
 titulos(es_x1+6,es_y1+6,es_x2-6,es_y1+26,AZUL);
 titulos(es_x1+60,es_y1+36,es_x1+110,es_y1+50,9);
 PoneCadena(es_x1+20,es_y1+10,NEGRO,"N§ de iteraciones");
 _setcolor(8);
 boto1(es_x1+114,es_y1+32,es_x1+126,es_y1+42,1,"");
 boto1(es_x1+114,es_y1+44,es_x1+126,es_y1+54,1,"");
 DibujaFlecha(es_x1+120,es_x1+120,es_y1+35,1);
 DibujaFlecha(es_x1+120,es_x1+120,es_y1+51,0);
 boton(es_x1+65,es_y1+55,es_x1+110,es_y1+73,"OIR" ,TRUE);
 _moveto(es_x1+67,es_y1+36);
```

```
_outgtext("1");
punteroon();

    do
    {
    leeraton(&posix,&posiy,&dcha,&izda);
    if((posix > es_x1+115) && (posix < es_x1+125))
        if((posiy>es_y1+32) && (posiy<es_y1+40) && (dcha) )

    {
            _moveto(es_x1+67,es_y1+36);
            if(indice<10)
            {
            CuaDrado(RELLENO,es_x1+60,es_y1+36,es_x1+109,es_y1+50,9);
            _setcolor(0);
            _outgtext(tabla[indice++]);
            NumeroIteraciones=atof(tabla[indice-1]);
             for(i=0;i<320000;i++)
             {
              }
            }
            else
            indice=0;
            contador++;
     }
    if((posix > es_x1+115) && (posix < es_x1+125))
       if((posiy > es_y1+46) && (posiy < es_y1+54) && (dcha))
       {

         _moveto(es_x1+67,es_y1+36);
         if(indice>0)
```

```
            CuaDrado(RELLENO,es_x1+60,es_y1+36,es_x1+109,es_y1+50,9);
            _setcolor(0);
            _outgtext(tabla[--indice]);
            NumeroIteraciones=atof(tabla[indice]);
              for(i=0;i<320000;i++)
               {
               }
            }
            else
            indice=10;
          contador++;
        }
 if((posix > es_x1+65) && (posix <es_x1+110))
   if((posiy> es_y1+55) && (posiy< es_y1+73) &&(dcha))
    {
       while(dcha)
       {
        leeraton(&posix,&posiy,&dcha,&izda);
        boton(es_x1+65,es_y1+55,es_x1+110,es_y1+73,"OIR" ,FALSE);
       }
       boton(es_x1+65,es_y1+55,es_x1+110,es_y1+73,"OIR" ,TRUE);
       oir=TRUE;
    }
 }
 while(oir==FALSE);
punteroff();
_putimage(es_x1-5,es_y1-5,BufferImagen,_GPSET);
hfree(BufferImagen);
punteroon();
 if(contador==0)
    {
```

```
    Itera=(unsigned long)1;
    return(Itera);
    }
  else
    {
    Itera=(unsigned long)indice;
    return(Itera);
    }

}
```

```
/* FICHERO QUE COPIA UN TROZO DE FICHERO EN OTRO */

typedef char BOOLEAN;
#include <stdio.h>
#include <fcntl.h>
#include <graph.h>
#include <io.h>
#include <stdlib.h>
#include <malloc.h>
#include "colores.h"
#define TRUE 1
#define FALSE 0
#define RELLENO 3
void leeraton (int *,int *, BOOLEAN *,BOOLEAN *);
void boton(int,int,int,int,char *,BOOLEAN);
void Cuadrado(int,int,int,int,int,int);
char * NuevoFile();
int ExisFile(char *);
int ModiFile(int,int,int,int);

void copiar(void)
{
 int
x1,x2,x3,fh,dato,llave0,xv1=100,yv1=100,xv2=300,yv2=175,resulta,fila1,fila2,column
a,insertar=0,pox,poy;
 double
      x_scale;
 unsigned long
            pts_to_plot,tafi;
 long
      pui,puf,i,pins,TamaMenu;
```

```
char
    fichero[12],new[]="xxxx.xx",*NomFile;

FILE
    *pf0,*pf1,*pf2,*pfi,*pfn;

static char huge * BufMenu=NULL;

BOOLEAN
     dcha,izda;
struct
{
 int NumCanales;
 unsigned int Fr;
 }cabecera;

Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
llave0=ModiFile(xv1,yv1,xv2,yv2);  /*indica si se modificar  el fichero original*/
if((pfi=fopen("inter.fil","r+b"))==NULL)
{
   _setvideomode(3);
   printf("Ha ocurrido un error en la aplicación");
   getch();
   exit(1);
 }
fread(&fichero,sizeof(fichero),1,pfi);
fclose(pfi);
if(llave0==1) /* si se modifica el fichero*/
{
 do
 {
  NomFile=NuevoFile();
```

```
  if(*NomFile=='\0')
    {
     return;
    }
  resulta=ExisFile(NomFile);
  if(resulta==1)
  {
   Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
   PoneCadena(5,457,NEGRO,"EL FICHERO YA EXISTE");
  }
 }
 while(resulta==1);
 Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
 if((pfn=fopen(NomFile,"w+b"))==NULL)
  {
   _setvideomode(3);
   printf("\t\t\n\nHa ocurrido un error en la aplicación");
   getch();
   exit(1);
  }
}
if((fh=open(fichero,O_RDONLY,O_BINARY))<0)
{
   _setvideomode(3);
   printf("\t\t\n\nHa ocurrido un error en la aplicación");
   getch();
   exit(1);
}
tafi=filelength(fh);
pts_to_plot=tafi/2;
close(fh);
x_scale=(double)(640/(double)(pts_to_plot));
```

```
 if((pf0=fopen(fichero,"r+b"))==NULL)
 {
    _setvideomode(3);
    printf("\t\t\n\nHa ocurrido un error en la aplicaci¢n");
    getch();
    exit(1);
 }
 fread(&cabecera,sizeof(int),1,pf0);
 rewind(pf0);
if(llave0==1)
 {
  TamaMenu=_imagesize(xv1,yv1,xv2-10,yv1+20);
  BufMenu=(char huge *) halloc(TamaMenu,sizeof(char));
  punteroff();
  _getimage(xv1,yv1,xv2-10,yv1+20,BufMenu);
  boton(xv1,yv1,xv2-10,yv1+20,"ESPERE POR FAVOR",TRUE);

  while(fread(&dato,sizeof(int),1,pf0)==1)
  fwrite(&dato,sizeof(int),1,pfn);
  rewind(pf0);
  _putimage(xv1,yv1,BufMenu,_GPSET);
  hfree(BufMenu);
  punteroon();
}
 if((pf1=tmpfile())==NULL)
  {
    _setvideomode(3);
    printf("\t\t\n\nHa ocurrido un error en la aplicaci¢n");
    getch();
    exit(1);
  }
 if((pf2=tmpfile())==NULL)
```

```
{
   _setvideomode(3);
   printf("\t\t\n\nHa ocurrido un error en la aplicaci¢n");
   getch();
   exit(1);
}
Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
PoneCadena(5,457,NEGRO,"MARQUE LA ZONA A CORTAR BOTON
                                    DERECHO, BOTON IZQUIERDO");

marcar(&x1,&x2,fichero);
pui=(long)(x1/x_scale);
puf=(long)(x2/x_scale);
if(pui<4L) /*si marcamos la cabecera evita que se corte esta*/
pui=4L;
Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
PoneCadena(5,457,NEGRO,"MARQUE LA ZONA PARA INSERTAR,  BOTON
                                                DERECHO");

do
{
  leeraton (&pox,&poy,&dcha,&izda);
  if((pox>1) && (pox<638))
    if((poy>62) && (poy<179) && (dcha))
    {
     x3=pox;
     insertar=1;
     }
  if((pox>1) && (pox<638))
    if((poy>207) && (poy<334) && (dcha))
    {
     x3=pox;
```

```
    insertar=1;
    }

  if(insertar==1) break;
}
 while(insertar==0);
 TamaMenu=_imagesize(xv1,yv1,xv2-10,yv1+20);
 BufMenu=(char huge *) halloc(TamaMenu,sizeof(char));
 punteroff();
 _getimage(xv1,yv1,xv2-10,yv1+20,BufMenu);
 boton(xv1,yv1,xv2-10,yv1+20,"ESPERE POR FAVOR",TRUE);
 pins=(long)(x3/x_scale);
 fseek(pf0,(long)(2*pui),0);
 for(i=0;i<(puf-pui);i++)
  {
   fread(&dato,sizeof(int),1,pf0); /*salva el trozo selecionado en pf1*/
   fwrite(&dato,sizeof(int),1,pf1);
  }
 rewind(pf0);
for(i=0;i<pins;i++)
{
 fread(&dato,sizeof(int),1,pf0); /*copiamos el fichero hasta el punto de*/
 fwrite(&dato,sizeof(int),1,pf2);  /*insertar*/
}
rewind(pf1);
while(fread(&dato,sizeof(int),1,pf1)==1) /*insertamos*/
 fwrite(&dato,sizeof(int),1,pf2);
while(fread(&dato,sizeof(int),1,pf0)==1)
 fwrite(&dato,sizeof(int),1,pf2);
if((pf0=fopen(fichero,"w+b"))==NULL)   /*borra el fichero*/
{
 printf("Error no se puede abrir el fichero");
```

```
 exit(1);
}
rewind(pf2);
while(fread(&dato,sizeof(int),1,pf2)==1) /*copiamos el fichero total*/
 fwrite(&dato,sizeof(int),1,pf0);
 fclose(pf0);   /* CERRAMOS LOS FICHEROS */
 fclose(pf1);
 fclose(pf2);
 if(llave0==1)
 {
  fclose(pfn);
  resulta=rename(NomFile,new);   /* renombra los ficheros */
  resulta=rename(fichero,NomFile);
  resulta=rename(new,fichero);
 }
 _putimage(xv1,yv1,BufMenu,_GPSET);
 hfree(BufMenu);
 punteroon();
 Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
/*borra la marca*/
punteroff();
 for(fila1=76;fila1<169;fila1++)
      for(columna=x1;columna<x2;columna++)
      {
       if(_getpixel(columna,fila1)==CIAN)
       {
         _setcolor(NEGRO);
         _setpixel(columna,fila1);
       }
       else
        if (_getpixel(columna,fila1)==ROJO)
         {
```

```
            _setcolor(ROJO);
            _setpixel(columna,fila1);
          }
        }
 if(cabecera.NumCanales==2)
  {
    for(fila2=220;fila2<325;fila2++)
        for(columna=x1;columna<x2;columna++)
        {
         if(_getpixel(columna,fila2)==CIAN)
          {
           _setcolor(NEGRO);
           _setpixel(columna,fila2);
          }
         else
          if (_getpixel(columna,fila2)==MAGNETA)
            {
             _setcolor(MAGNETA);
             _setpixel(columna,fila2);
            }
        }
    }
    punteroon();

if(llave0==1)

      dibuja(NomFile);
 else
      dibuja(fichero);
}
```

```
/* PROCEDIMIENTO QUE BORRA UN TROZO DE FICHERO */

typedef char BOOLEAN;
typedef unsigned int WORD;
#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#include <malloc.h>
#include <graph.h>
#include <stdlib.h>
#include "colores.h"
#define RELLENO 3
#define TRUE 1
#define FALSE 0
void Cuadrado(int,int,int,int,int,int);
void leeraton(int *, int *, BOOLEAN *,BOOLEAN *);
void boton(int,int,int,int,char *,BOOLEAN);
void marcar(int *,int *, char *);
char * NuevoFile();
int ExisFile(char *);
char *escribe(int,int,int,int,int,char *,char *);
int ModiFile(int,int,int,int);
void dibuja( char *);

void borrar(void)
{
 int
      x1,x2,fh,dato,resultado,xv1=100,yv1=100,xv2=300,yv2=175;
 short
      llave0;
 double
      x_scale;
```

```
unsigned long
        pts_to_plot,tafi;
long
    pui,puf,i,tamaMenu;
char
    fichero[12],new[]="xxxx.xx",*NomFile;
FILE
    *pf0,*pf1,*pfi,*pfn;

static char huge *BufMenu=NULL;

BOOLEAN
     si,no,dcha,izda;

/*RUTINA QUE GESTIONA LA OPCION DE MODIFICAR EL FICHERO
                                                ORIGINAL O NO*/

Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
llave0=ModiFile(xv1,yv1,xv2,yv2);
if((pfi=fopen("inter.fil","r+b"))==NULL)    /*lee fichero de intercanbio*/
{
   _setvideomode(3);
   printf("Ha ocurrido un error en la aplicaci¢n");
   getch();
    exit(1);
 }
fread(&fichero,sizeof(fichero),1,pfi);
fclose(pfi);
if (llave0==1) /*se modifica*/
{
 do
  {
```

```
  NomFile=NuevoFile();
  if(*NomFile=='\0')
  {
   return;
  }
  resultado=ExisFile(NomFile);
  if(resultado==1)
  {
     Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
     PoneCadena(8,455,NEGRO,"EL FICHERO YA EXISTE");
  }
}

if((pfn=fopen(NomFile,"w+b"))==NULL)    /*abre el fichero para guardar*/
  {                                     /*la modificacion*/
     _setvideomode(3);
     printf("Ha ocurrido un error en la aplicación");
     getch();
     exit(1);
  }
}

if((fh=open(fichero,O_RDONLY,O_BINARY))<0) /*determina el tamaño del */
{
     _setvideomode(3);
     printf("Ha ocurrido un error en la aplicaci¢n");
     getch();
     exit(1);
}
tafi=filelength(fh);
pts_to_plot=tafi/2;
close(fh);
```

```
x_scale=(double)(640)/(double)(pts_to_plot);
if((pf0=fopen(fichero,"r+b"))==NULL)
{
     _setvideomode(3);
     printf("Ha ocurrido un error en la aplicaci¢n");
     getch();
     exit(1);
}
if(llave0==1)
  {
    tamaMenu=_imagesize(xv1,yv1,xv2-10,yv1+20);
    BufMenu=(char huge *) halloc(tamaMenu,sizeof(char));
     punteroff();
     _getimage(xv1,yv1,xv2-10,yv1+20,BufMenu);
     boton(xv1,yv1,xv2-10,yv1+20,"ESPERE POR FAVOR",TRUE);
     while(fread(&dato,sizeof(int),1,pf0)==1) /*copia voz.bin a voz1.bin*/
          fwrite(&dato,sizeof(int),1,pfn);
     rewind(pf0);
     _putimage(xv1,yv1,BufMenu,_GPSET);
     hfree(BufMenu);
     punteroon();
 }
if( (pf1=tmpfile())==NULL)
{
     _setvideomode(3);
     printf("Ha ocurrido un error en la aplicaci¢n");
     getch();
     exit(1);
}
 Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
 PoneCadena(5,457,NEGRO,"MARQUE LA ZONA A BORRAR, BOTON
                                        DERCHO, BOTON IZQUIERDO");
```

```
  marcar(&x1,&x2,fichero);
  tamaMenu=_imagesize(xv1,yv1,xv2-10,yv1+20);
  BufMenu=(char huge *) halloc(tamaMenu,sizeof(char));
  punteroff();
  _getimage(xv1,yv1,xv2-10,yv1+20,BufMenu);
  boton(xv1,yv1,xv2-10,yv1+20,"ESPERE POR FAVOR",TRUE);
  pui=(long)(x1/x_scale);
  puf=(long)(x2/x_scale);
  if(pui<4L)
    pui=4;
  for(i=0;i<pui;i++)
   {
    fread(&dato,sizeof(int),1,pf0);
    fwrite(&dato,sizeof(int),1,pf1);
   }
  fseek(pf0,(long)(2*puf),0);
  while(fread(&dato,sizeof(int),1,pf0)==1)
  fwrite(&dato,sizeof(int),1,pf1);
  if((pf0=fopen(fichero,"w+b"))==NULL)
     {
      _setvideomode(3);
      printf("Ha ocurrido un error en la aplicaci¢n");
      getch();
      exit(1);
     }
rewind(pf1);
 while(fread(&dato,sizeof(int),1,pf1)==1)
 fwrite(&dato,sizeof(int),1,pf0);
 fclose(pf0);
 fclose(pf1);
 if(llave0==1) /*se elige guardar los cambios*/
 {
```

```
    fclose(pfn);    /* renombra los ficheros */
    resultado=rename (NomFile,new);
    resultado=rename (fichero,NomFile);
    resultado=rename (new,fichero);
  }
  _putimage(xv1,yv1,BufMenu,_GPSET);
  hfree(BufMenu); /* libera la memoria */
  punteroon();
  _setcolor(0);
  _rectangle(3,0,448,639,467); /*borra la parte baja*/
  if(llave0==1)
      dibuja(NomFile);
  else
      dibuja(fichero);
}

char * NuevoFile()  /* permite escribir un nuevo fichero si ya existe */
{
  char *NomFich;
  NomFich=escribe(100,200,350,280,10,"Nombre del Fichero","");
  if(*NomFich=='\0')
  _cexit();
  return(NomFich);
}

int ExisFile(char *NomFile)  /*procedimiento para saber si un fichero existe*/
{
 char NombreRuta[_MAX_PATH];
 int existe;
 char entorno[]="COMPILAR";
 _searchenv(NomFile,entorno,NombreRuta);
 if(*NombreRuta !='\0')
```

```
{
 existe=1; /*el fichero no existe*/
 }
 else
 {
  existe=0; /*el fichero existe*/
 }
 return(existe);
}
int ModiFile(int xv1,int yv1,int xv2,int yv2)
{
 short sw0;
 int posix,posiy;
 static char huge *BufMenu=NULL;
 BOOLEAN si,no,dcha,izda;
 long tamaMenu;

 tamaMenu=_imagesize(xv1,yv1,xv2,yv2);
 BufMenu=(char huge *) halloc(tamaMenu,sizeof(char));
 punteroff();
 _getimage(xv1,yv1,xv2,yv2,BufMenu);
 boton(xv1,yv1,xv2,yv2-10,"Quiere guardar los",TRUE);
 _moveto(xv1+10,yv1+25);
 _outgtext("cambios en otro fichero");
 si=FALSE;
 no=FALSE;
 boton(xv1+20,yv1+45,xv1+50,yv1+63,"SI",TRUE);
 boton(xv1+145,yv1+45,xv1+180,yv1+63,"NO",TRUE);
 punteroon();

 do
 {
```

```
leeraton(&posix,&posiy,&dcha,&izda);
if((posix> xv1+20) && (posix < xv1+50))
 if((posiy >yv1+45) && (posiy <yv1+63) && (dcha))
 {
  while(dcha)
   {
    leeraton(&posix,&posiy,&dcha,&izda);
    boton(xv1+20,yv1+45,xv1+50,yv1+63,"SI",FALSE);
   }
    boton(xv1+20,yv1+45,xv1+50,yv1+63,"SI",TRUE);
    si=TRUE;  /* se ha pulsado si */
 }

if((posix>xv1+145) &&(posix < xv1+180))
 if((posiy >yv1+45) && (posiy <yv1+63) &&(dcha))
 {
  while(dcha)
   {
    leeraton(&posix,&posiy,&dcha,&izda);
    boton(xv1+145,yv1+45,xv1+180,yv1+63,"NO",FALSE);
   }
    boton(xv1+145,yv1+45,xv1+180,yv1+63,"NO",TRUE);
    no=TRUE;
 }
 if(si)
    {
     sw0=1;
     break;
     }
 if(no)
    {
     sw0=0;
```

```
        break;
        }
   }
  while(1);
punteroff();
  _putimage(xv1,yv1,BufMenu,_GPSET);
  punteroon();
  hfree(BufMenu);
if(sw0==1)
  return(1);
  if(sw0==0)
  return(0);
}
```

```
/*PROCEDIMIENTO QUE PERMITE HACER UN ZOOM*/

typedef unsigned char BOOLEAN;
typedef unsigned int WORD;
#include <graph.h>
#include <malloc.h>
#include <fcntl.h>
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include "c:\atdsp\dsp2200.h"
#include <io.h>
#include "colores.h"
#define FALSE 0
#define TRUE 1
#define RELLENO 3
#define HUECO 2

void marcar(int *, int *, char *);
extern void reproducir(long , long ,int );
int dialogo(void);
void boton(int,int,int,int,char * ,BOOLEAN);
int mas();
void amplia(unsigned long,unsigned long,char *);

void zoom(void)
{
   int
        x1,x2,fh,conta=0,temp,psx,psy,sw=0,ori=1,estado,salida,i;
   unsigned long
                tafi,pts_to_plot;
```

```
    long
          pui,puf,punf,tamaImag;
    double
           x_scale;
    BOOLEAN
            izda,dcha;
    char
           fichero[12],car,huge * buffer;
    FILE
          *pfi,*pfp;
    float
           xt;
        /*leeo en fichero de intercambio de fichero para saber el nimbre del fichero*/
if((pfi=fopen("inter.fil","r+b"))==NULL) /* abre el fichero de intercambio */
  {
   printf("ERROR AL ABRIR EL FICHERO");
   exit(1);
  }
  fread(&fichero,sizeof(fichero),1,pfi);
  fclose(pfi);

   if((fh=open(fichero,O_RDONLY,O_BINARY))<0)
    {
     printf("No se puede abrir el fichero");
     exit(1);
    }
   tafi=filelength(fh); /* calcula la longitud del fichero */
   pts_to_plot=tafi/2;
   close(fh);
   x_scale = (double)(640)/(double)(pts_to_plot);
  punteroff();
  Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
```

```
 PoneCadena(5,457,NEGRO,"MARQUE LA ZONA A AMPLIAR: BOTON
                                    DERECHO, BOTON IZQUIERDO");

 punteroon();
 /*guarda la señal para luego restaurarla*/
if((pfp=tmpfile())==NULL)
{
 printf("ERROR AL ABRIR EL FICHERO p00000.000");
 exit(1);
}
 tamaImag=_imagesize(1,0,638,12);

/* buffer de memoria para ser usado para guardar la señal al disco */
 buffer=(char huge *) halloc(tamaImag,sizeof(char));
 if(buffer==NULL)
 {
  printf("ERROR DE ASIGNACION DE MEMORIA");
  exit(1);
  }
 punteroff();
 for(i=60;i<179;i+=12)
 {
   _getimage(1,i,638,i+12,buffer);
   fwrite(buffer,sizeof(char),(int)tamaImag,pfp);
 }

 for(i=208;i<334;i+=12)
 {
   _getimage(1,i,638,i+12,buffer);
   fwrite(buffer,sizeof(char),(int)tamaImag,pfp);
 }
 punteroon(
```

```
do
{
 if(conta>0)
 {
  Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
  PoneCadena(5,457,NEGRO,"MARQUE LA ZONA A AMPLIAR: BOTON
                                        DERECHO, BOTON IZQUIERDO");
 }
 marcar(&x1,&x2,fichero); /* llama a marcar */
 Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
 leeraton(&psx,&psy,&izda,&dcha); /*rutina de salida*/
 if(izda)
   {
        sw=1;
        break;
   }
 if(x1>x2)
  {
   temp=x2;
   x2=x1;
   x1=temp;
  }
 if(x1==x2)
  {
   sw=1;
   break;
  }
if((puf-pui>20) || (conta==0))
 { /*if1*/
   if(conta==0)
       {
         pui=(long)(x1/x_scale);   /* calcula el desplazamientos dentro del fichero */
```

```
        puf=(long)(x2/x_scale);
      }
    else
      {
        puf=(long)(x2/(double)(640/(double)(puf-pui))+pui);
        pui=(long)(x1/(double)(640/(double)(punf-pui))+pui);
      }
    punf=puf;
    punteroff();
    Cuadrado(RELLENO,1,62,638,179,NEGRO); /*borra antes de dibujar la
                                                    ampilacion*/
    PoneEscala(179);
    Cuadrado(RELLENO,1,207,638,334,NEGRO); /*borra antes de dibujar la
                                                    ampilacion*/
    PoneEscala(340);
    punteroon();
    amplia(pui,puf,fichero);
    salida=dialogo();        /*colocamos la llamada*/
    if(salida==2)
      {
       reproducir(pui,puf,ori);
       sw=1;
       break;
       /*despues de reproducir regresa aqui*/
      }
    if(salida==3)
      {
       sw=1;
       break;
      }
       conta++;
} /* fin del if1*/
```

```
        if(kbhit() && getch()==27) break;
  } /* fin del do*/
while((puf-pui>20) ||(izda));   /* ampliar mientras las muestras sean mayor de 20 */
if(sw==0)
  {
   punteroff();
   Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
   PoneCadena(5,457,NEGRO,"IMPOSIBLE AMPLIAR MAS, PULSE UNA
                                                          TECLA");
   punteroon();
   getch(); /* espera a que se pulse una tecle */
   }
  rewind(pfp);
   punteroff();
   for(i=60;i<179;i+=12)
     {
        fread(buffer,sizeof(char),(int)tamaImag,pfp);
        _putimage(1,i,buffer,_GPSET);
      }
  for(i=208;i<334;i+=12)
  {
   fread(buffer,sizeof(char),(int)tamaImag,pfp);
   _putimage(1,i,buffer,_GPSET);
  }
   hfree(buffer);
   fclose(pfp);
   punteroon();
}

/* procedimiento que hace las ampliaciones */
void amplia(unsigned long punteroi,unsigned long punterof,char *nomfile)
{
```

```
unsigned long tafi,
        pts_to_plot,
            xcount,
                i,
              tama,l,k;
 double   x_scale,
           y_scale;

 long max=0,min=0;
unsigned int x, y;
        int j,dato;

        char *maximo,*minimo;
 FILE *pf;
  struct videoconfig vconfig;

  /* estructura que contiene los parámetros de la cabecera */
  struct
  {
   int num_cana;
   unsigned int fm;
   }cabecera;

 _getvideoconfig(&vconfig); /* obtiene la configuración de video */
 punteroff();
 if((pf=fopen(nomfile,"r+b"))==NULL)
 {
  printf("ERROR AL ABRIR EL FICHERO");
  exit(1);
 }
 fread(&cabecera,sizeof(cabecera),1,pf);
 rewind(pf);
```

```
tama=(long)(punterof-punteroi);
   fseek(pf,(long)(2*punteroi),0);
   for(i=punteroi;i<punterof;i++)
   {
    fread(&dato,sizeof(int),1,pf);
    if(dato<min) min=dato;
    if(dato>max) max=dato;
   }
   /* calculo de las escalas de los ejes x e y */
   x_scale = (double)((vconfig.numxpixels)-2)/(double)(tama);
   y_scale = (double)((vconfig.numypixels/2)-150)/(double)(max-min);
   _setlogorg(1, 42);
   _setcolor (4);
  _moveto(0,0);
   x = 0;
   xcount = 0;
   fseek(pf,(2*punteroi),0); /* desplaza el puntero hacia la nueva zona a ampliar */
   for(j=0;j<cabecera.num_cana;j++)
      {
       x=xcount=0;
       y=(int)((dato-min)*y_scale);
       y=((vconfig.numypixels/2)-150)-y;
       _setcolor(4+j);
       _moveto(x,y+35);
      for(i=punteroi;i<punterof;i++)
        {
          fread(&dato,sizeof(int),1,pf);
          x = (int) ( (double)(++xcount) * x_scale );
          y = (int) ( (dato- min) * y_scale );
          y = ((vconfig.numypixels/2)-150) - y;
         _lineto( x, y+35);
   }
```

```
if(cabecera.num_cana==2)
   {
    fseek(pf,(long)(2*punteroi),0);
    _setlogorg(1,185);
    }
}
  _settextposition(1,20);
  fclose(pf);
   _setlogorg(0,0);
   punteroon();
}

int dialogo(void)
{
 long tamaMenu;
 int x1=190,y1=115,x2=376,y2=135;
 BOOLEAN dcha,izda;
 WORD posix,posiy;
 static char huge *bufMenu=NULL;
 BOOLEAN go,zom,oir,salir;

/* reserva espacio para los botones */
  tamaMenu=_imagesize(x1,y1,x2,y2);
  bufMenu=(char huge *) halloc(tamaMenu,sizeof(char));
   go=FALSE;
   zom=FALSE;
   oir=FALSE;
   salir=FALSE;
    punteroff();
    _getimage(x1,y1,x2,y2,bufMenu);
```

```
  /* dibuja los botones */

  boton(x1,y1,x1+55,y2,"Zoom",TRUE);
  boton(x1+65,y1,x1+120,y2,"Oir",TRUE);
  boton(x1+130,y1,x1+185,y2,"Salir",TRUE);
  punteroon();

/* espera a que se pulse un boton */
 do
  {
  leeraton(&posix,&posiy,&dcha,&izda);
  if((posix >x1) && (posix <x1+55))
    if((posiy>y1) && (posiy<y2) &&(dcha))
     {
          while(dcha)
          {
           leeraton(&posix,&posiy,&dcha,&izda);   /* se pulsa el boton de zoom */
           boton(x1,y1,x1+55,y2,"Zoom",FALSE);
           }
          boton(x1,y1,x1+55,y2,"Zoom",TRUE);
          go=TRUE; /* se ha pulsado zoom */
          zom=TRUE;
     }
  if((posix >x1+65) && (posix <x1+120))
    if((posiy>y1) && (posiy<y2) &&(dcha))
     {
          while(dcha)
          {
           leeraton(&posix,&posiy,&dcha,&izda);
           boton(x1+65,y1,x1+120,y2,"Oir",FALSE); /* se pulsa el boton de oir */

           }
```

```
        boton(x1+65,y1,x1+120,y2,"Oir",TRUE);
        go=TRUE;
        oir=TRUE;  /* se pulsó oir */
    }
if((posix >x1+130) && (posix <x1+185))
  if((posiy>y1) && (posiy<y2) &&(dcha))
   {
        while(dcha)
        {
         leeraton(&posix,&posiy,&dcha,&izda);
         boton(x1+130,y1,x1+185,y2,"Salir",FALSE); /* se pulsa salir */
        }
        boton(x1+130,y1,x1+185,y2,"Salir",TRUE);
        go=TRUE;
        salir=TRUE; /* se pulsó salir */
   }
   if(go)  /* si se ha pulsado alguna de las teclas salir del bucle */
   {
       break;
   }
  }
while(1); /* repetir indefinidamente */
punteroff();
_putimage(x1,y1,bufMenu,_GPSET); /* borra los botenes restaurando el fondo */
punteroon();
hfree(bufMenu);
if(zom)
 {
      punteroon();
      return(1); /* si ha pulsado Zoom retornamos al procedimiento zoom */
```

```
    }
    if(oir)
    {
        punteroon();
        return(2); /* si se ha pulsado oir retorna al procedimiento oir */
    }
    if(salir)
    {
     punteroon();
        return(3); /* si salir retornamos al procedimiento salir */
    }
}
```

```
/*DIBUJA LA PANTALLA DE ADQUISICION*/

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned char BOOLEAN;

#include <malloc.h>
#include <ctype.h>
#include "nidaq.h"
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>
#include "colores.h"

#define SLOT 1
#define TRUE 1
#define FALSE 0
#define MAX 15
#define RELLENO 3
#define HUECO 2

void aspa(int,int,int,int,int); /* función para poner las aspas cuando marcamos */
char *escribe(int,int,int,int,int, char *,char *);
void boton(int,int,int,int,char *, BOOLEAN);
void texto(int,int,int,char *);
void boton(int, int ,int, int, char *,BOOLEAN);
void menu1(int,int,int,int);

char * NuevoFile();
int ExisFile(char *);
```

```
void leer (BOOLEAN,BOOLEAN,BOOLEAN,BOOLEAN,int,unsigned int,char
*,char *, char *,BOOLEAN, BOOLEAN, BOOLEAN, BOOLEAN, BOOLEAN,
char *, BOOLEAN, BOOLEAN, BOOLEAN);

/* función prototipo de la llamada a la función que realiza la adquisición */
void adquisicion(void)
{
 WORD
        posix,posiy,x1p,y1p,x2p,y2p,x1,y1,x2,y2;
 BOOLEAN
        canal1,dcha,izda,canal0,aspa0,aspa1,triger,asptr,salir,go,
        acoplo_dc,acoplo_ac,llave0,llave1,llave2,llave3,llave4,trigerd,trigera,subia,
        bajaa,dicad,dicai,subid,bajad;
 static char huge *panta=NULL;
 static char huge *panta1=NULL;
 static char huge *panta2=NULL;
 long
        tama_panta,tama_panta1,tama_panta2;
 int
        i,resultado,contador=0;
 unsigned int
        division;
 int
        frecuencia;
 char
        *cad0,*cad2,*cad4,*cad6;
 static char cad1[5]="0";
 static char cad3[5]="0";
 static char cad5[5]="0";
 static char cad7[15]="sonido.dat";
 char
    *NomFile;
```

```
/* iniciación de variables */
 x1p=100;
 y1p=20;
 x2p=300;
 y2p=275;
 x1=100;
 y1=300;
 x2=440;
 y2=450;

 punteroff();

/* crea el buffer de memoria para salvar el fondo */
 tama_panta=_imagesize(x1p,y1p,x2p,y2p);
 panta=(char huge *) halloc(tama_panta,sizeof(char));
 _getimage(x1p,y1p,x2p,y2p,panta);

/* crea la pantalla de selección */
 _setcolor(BLANCOBRILLANTE);
 _rectangle(_GFILLINTERIOR,x1p,y1p,x2p,y2p);
  _setcolor(NEGRO);
  _rectangle(2,x1p,y1p,x2p,y2p);

  _setcolor(7);
  _rectangle(_GFILLINTERIOR,x1p+2,y1p+2,x2p-2,y2p-2);
  _setcolor(NEGRO);
  _rectangle(2,x1p+2,y1p+2,x2p-2,y2p-2);

 _setcolor(NEGRO);
 _moveto(x1p+2,y1p+85);  /* lineas que separan opcines*/
 _lineto(x2p-2,y1p+85);     /*primera linea separadora*/
```

```
_moveto(x1p+2,y1p+102); /*segunda linea separadora*/
_lineto(x2p-2,y1p+102);

_moveto(x1p+2,y1p+188);  /*tercera linea separadora*/
_lineto(x2p-2,y1p+188);

_moveto(x1p+2,y1p+205);  /*cuarta linea separadora*/
_lineto(x2p-2,y1p+205);

_moveto(x1p+110,y1p+85); /*linea vertical separadora*/
_lineto(x1p+110,y1p+205);

_moveto(x1p+2,y1p+222); /*qinta linea separadora*/
_lineto(x2p-2,y1p+222);

 for(i=10;i<71;i+=20)   /* pone los cuadrados pequeños */
 _rectangle(2,x1p+10,y1p+i,x1p+20,y1p+i+10);

 _ellipse(2,x1p+10,y1p+192,x1p+20,y1p+202);  /* cuadrado de pretrigger*/

 _ellipse(2,x1p+120,y1p+192,x1p+130,y1p+202);  /* cuadrado de retardo*/

 _ellipse(2,x1p+10,y1p+209,x1p+20,y1p+219); /* cuadra de fichero*/

texto(NEGRO,x1p+24,y1p+7,"Canal Derecho");
texto(NEGRO,x1p+24,y1p+26,"Canal Izquierdo");
texto(NEGRO,x1p+24,y1p+47,"Acoplo AC");
texto(NEGRO,x1p+24,y1p+67,"Acoplo DC");
texto(NEGRO,x1p+10,y1p+85,"Frecuencia");
texto(NEGRO,x1p+120,y1p+85,"Divisi¢n");
texto(NEGRO,x1p+24,y1p+104,"51.2 KHz");
texto(NEGRO,x1p+24,y1p+124,"48.0 KHz");
```

```
 texto(NEGRO,x1p+24,y1p+144,"44.1 KHz");
 texto(NEGRO,x1p+24,y1p+164,"32.0 KHz");
 texto(NEGRO,x1p+134,y1p+104,"x 1");
 texto(NEGRO,x1p+134,y1p+124,"x 2");
 texto(NEGRO,x1p+134,y1p+144,"x 4");
 texto(NEGRO,x1p+134,y1p+164,"x 8");
 texto(NEGRO,x1p+24,y1p+188,"Pretrigger");
 texto(NEGRO,x1p+134,y1p+188,"Retardo");
 texto(NEGRO,x1p+24,y1p+207,"Nombre del Fichero");

 for(i=107;i<168;i+=20)
 _rectangle(2,x1p+10,y1p+i,x1p+20,y1p+i+10);

 for(i=107;i<168;i+=20)
 _rectangle(2,x1p+120,y1p+i,x1p+130,y1p+i+10);

 boton(x1p+10,y1p+227,x1p+60,y1p+244,"GO !",TRUE);
 boton(x2p-60,y1p+227,x2p-10,y1p+244,"EXIT",TRUE);

/*localizacion de coordenadas*/
 aspa(x1p+10,y1p+10,x1p+20,y1p+20,0);
 aspa(x1p+10,y1p+30,x1p+20,y1p+40,0);
 aspa(x1p+10,y1p+70,x1p+20,y1p+80,0);
 aspa(x1p+10,y1p+167,x1p+20,y1p+177,0); /*marcada la frecuencia de 32.0 KHz*/
 aspa(x1p+120,y1p+107,x1p+130,y1p+117,0); /*marca la division por dos*/

/* inicializa las variables */
 frecuencia=3;
 division=1;
 salir=FALSE;
 go=FALSE;
 trigerd=FALSE;
```

```
trigera=FALSE;
llave0=FALSE;
llave1=FALSE;
llave2=FALSE;
llave3=FALSE;
llave4=FALSE;
acoplo_ac=FALSE;
acoplo_dc=TRUE;
aspa0=TRUE;
aspa1=TRUE;
canal0=TRUE;
canal1=TRUE;
triger=FALSE;
asptr=FALSE;
subia=FALSE;
bajaa=FALSE;
dicad=FALSE;
dicai=FALSE;
subid=FALSE;
bajad=FALSE;

punteroon();
do /* espera a que elijan las opciones */
  {
   leeraton(&posix,&posiy,&dcha,&izda);

    if((posix >x1p+10) && (posix <x1p+20))
     {
      if((posiy >y1p+10) && (posiy < y1p+20) && (izda) &&(aspa0))
        {
         aspa(x1p+10,y1p+10,x1p+20,y1p+20,7);
         canal0=FALSE;
```

```
 aspa0=FALSE;
 }
if((posiy >y1p+10) && (posiy < y1p+20) && (dcha) &&(!aspa0))
 {
 aspa(x1p+10,y1p+10,x1p+20,y1p+20,0);
 canal0=TRUE;
 aspa0=TRUE;
 }
if((posiy >y1p+30) && (posiy < y1p+40) && (izda) && (aspa1))
 {
  aspa(x1p+10,y1p+30,x1p+20,y1p+40,7);
  canal1=FALSE;
  aspa1=FALSE;
 }
if((posiy >y1p+30) && (posiy < y1p+40) && (dcha) && (!aspa1))
 {
  aspa(x1p+10,y1p+30,x1p+20,y1p+40,0);
  canal1=TRUE;
  aspa1=TRUE;
 }
if((posiy >y1p+192) && (posiy < y1p+202) && (dcha) && (!asptr))
 {
   punteroff();
   _setcolor(0);
   _ellipse(3,x1p+12,y1p+194,x1p+18,y1p+200);
   triger=TRUE;
   asptr=TRUE;
  /* crea la pantalla para seleccionar los trigger */
   tama_panta1=_imagesize(x1,y1,x2,y2);
   panta1=(char  huge *) halloc(tama_panta1,sizeof(char));
   _getimage(x1,y1,x2,y2,panta1);
   menu1(x1,y1,x2,y2);
```

```
    punteroon();
   }
 if((posiy >y1p+192) && (posiy < y1p+202) && (izda) && (asptr))
   {
    punteroff();
   _setcolor(7);
   _ellipse(3,x1p+12,y1p+194,x1p+18,y1p+200);
    triger=FALSE;
    asptr=FALSE;
   _putimage(x1,y1,panta1,_GPSET);
    hfree(panta1);
    punteroon();
   }
 /* pone las aspas en la seleccion de frecuencia*/

if((posiy >y1p+50) && (posiy < y1p+60) && (dcha) )
  {
  aspa(x1p+10,y1p+50,x1p+20,y1p+60,0);
  aspa(x1p+10,y1p+70,x1p+20,y1p+80,7);
  acoplo_ac=TRUE;
  acoplo_dc=FALSE;
  _moveto(0,460);
  _outgtext("Seleciona Acoplo en DC");
  }
 if((posiy >y1p+70) && (posiy < y1p+80) && (dcha))
  {
  aspa(x1p+10,y1p+50,x1p+20,y1p+60,7);
  aspa(x1p+10,y1p+70,x1p+20,y1p+80,0);
  acoplo_ac=FALSE;
  acoplo_dc=TRUE;
  }
 if((posiy >y1p+107 ) && (posiy < y1p+117) && (dcha))
```

```
  {
  aspa(x1p+10,y1p+107,x1p+20,y1p+117,0);
  aspa(x1p+10,y1p+127,x1p+20,y1p+137,7);
  aspa(x1p+10,y1p+147,x1p+20,y1p+157,7);
  aspa(x1p+10,y1p+167,x1p+20,y1p+177,7);
  frecuencia=0;
  }
if((posiy >y1p+127 ) && (posiy < y1p+137) && (dcha))
 {
  aspa(x1p+10,y1p+107,x1p+20,y1p+117,7);
  aspa(x1p+10,y1p+127,x1p+20,y1p+137,0);
  aspa(x1p+10,y1p+147,x1p+20,y1p+157,7);
  aspa(x1p+10,y1p+167,x1p+20,y1p+177,7);
  frecuencia=1;
 }
if((posiy >y1p+147 ) && (posiy < y1p+157) && (dcha))
 {
  aspa(x1p+10,y1p+107,x1p+20,y1p+117,7);
  aspa(x1p+10,y1p+127,x1p+20,y1p+137,7);
  aspa(x1p+10,y1p+147,x1p+20,y1p+157,0);
  aspa(x1p+10,y1p+167,x1p+20,y1p+177,7);
  frecuencia=2;
 }
if((posiy >y1p+167 ) && (posiy < y1p+177) && (dcha))
 {
  aspa(x1p+10,y1p+107,x1p+20,y1p+117,7);
  aspa(x1p+10,y1p+127,x1p+20,y1p+137,7);
  aspa(x1p+10,y1p+147,x1p+20,y1p+157,7);
  aspa(x1p+10,y1p+167,x1p+20,y1p+177,0);
  frecuencia=3;
 }
if((posiy > y1p+209) && (posiy < y1p+219) &&(dcha))
```

```
    {
     punteroff();

        do
         {
        cad6=escribe(x1p-30,y1p+90,x2p+30,y2p-90,15,"Nombre del Fichero"," ");
          resultado=ExisFile(cad6);
          if(resultado==1) /*cuando resultado=0 el fichero no existe*/
                {
                 titulos(4,455,635,475,BLANCO);
                 Cuadrado(RELLENO,5,455,634,475,BLANCO);
                 PoneCadena(8,455,NEGRO,"EL FICHERO YA EXISTE");
                 contador++;
                }
          }
          while(resultado==1);
            if(contador>0)
            Cuadrado(RELLENO,5,455,634,475,BLANCO);
          strcpy(cad7,cad6);
          llave2=TRUE;
          punteroon();
    }

  } /*fin de las casillas de la izda*/

if((posix >x1p+120) && (posix < x1p+130))
  {
   if((posiy >y1p+107 ) && (posiy < y1p+117) && (dcha))
    {
     aspa(x1p+120,y1p+107,x1p+130,y1p+117,0);
     aspa(x1p+120,y1p+127,x1p+130,y1p+137,7);
     aspa(x1p+120,y1p+147,x1p+130,y1p+157,7);
```

```
  aspa(x1p+120,y1p+167,x1p+130,y1p+177,7);
  division=1;
  }

if((posiy >y1p+127 ) && (posiy < y1p+137) && (dcha))
  {
  aspa(x1p+120,y1p+107,x1p+130,y1p+117,7);
  aspa(x1p+120,y1p+127,x1p+130,y1p+137,0);
  aspa(x1p+120,y1p+147,x1p+130,y1p+157,7);
  aspa(x1p+120,y1p+167,x1p+130,y1p+177,7);
  division=2;
  }

if((posiy >y1p+147 ) && (posiy < y1p+157) && (dcha))
  {
  aspa(x1p+120,y1p+107,x1p+130,y1p+117,7);
  aspa(x1p+120,y1p+127,x1p+130,y1p+137,7);
  aspa(x1p+120,y1p+147,x1p+130,y1p+157,0);
  aspa(x1p+120,y1p+167,x1p+130,y1p+177,7);
   division=4;
  }
if((posiy >y1p+157 ) && (posiy < y1p+177) && (dcha))
  {
  aspa(x1p+120,y1p+107,x1p+130,y1p+117,7);
  aspa(x1p+120,y1p+127,x1p+130,y1p+137,7);
  aspa(x1p+120,y1p+147,x1p+130,y1p+157,7);
  aspa(x1p+120,y1p+167,x1p+130,y1p+177,0);
   division=8;
  }
if((posiy > y1p+192) && (posiy < y1p+202) &&(dcha))
  {
  punteroff();
```

```
      if(!llave3)
       {
       cad4=escribe(x1p-40,y1p+90,x2p+40,y2p-90,15,"Introduzca el retardo","0");
        strcpy(cad5,cad4);
        llave3=TRUE;
        }
       else
        {
     cad4=escribe(x1p-40,y1p+90,x2p+40,y2p-90,15,"Introduzca el retardo",cad5);
         strcpy(cad5,cad4);
        }
      punteroon();
     }
  }
if((posix > x2p-60) && (posix < x2p-10))
    if( (posiy >y1p+227) && (posiy <y1p+244) && (dcha))
     {
      while(dcha)
        {
        leeraton(&posix,&posiy,&dcha,&izda);
        boton(x2p-60,y1p+227,x2p-10,y1p+244,"EXIT",FALSE);
        }
        boton(x2p-60,y1p+227,x2p-10,y1p+244,"EXIT",TRUE);
        salir=TRUE;
     }

if((posix > x1p+10) && (posix < x1p+60))
    if( (posiy >y1p+227) && (posiy <y1p+244) && (dcha))
     {
      while(dcha)
        {
        leeraton(&posix,&posiy,&dcha,&izda);
```

```
      boton(x1p+10,y1p+227,x1p+60,y1p+244,"GO !",FALSE);
      }
      boton(x1p+10,y1p+227,x1p+60,y1p+244,"GO !",TRUE);
      if(llave2==TRUE)
            go=TRUE;
   }

if((posix > x1+10) && (posix < x1+20))
{
   if((posiy > y1+25) && (posiy < y1+35) && (dcha))
    {
     aspa(x1+10,y1+25,x1+20,y1+35,0);
     aspa(x1+10,y1+50,x1+20,y1+60,0);
     aspa(x1+10,y1+110,x1+20,y1+120,0);
     subia=TRUE;
     dicai=TRUE;
     cad3[0]='0';
     cad3[1]='\0';
     trigera=TRUE;
    }

   if((posiy > y1+25) && (posiy < y1+35) && (izda))
    {
     aspa(x1+10,y1+25,x1+20,y1+35,7);
     aspa(x1+10,y1+50,x1+20,y1+60,7);
     aspa(x1+10,y1+70,x1+20,y1+80,7);
     aspa(x1+10,y1+90,x1+20,y1+100,7);
     aspa(x1+10,y1+110,x1+20,y1+120,7);
     subia=FALSE;
     bajaa=FALSE;
     dicad=FALSE;
     dicai=FALSE;
```

```
   cad3[0]='0';
   cad3[1]='\0';
   trigera=FALSE;
  }
if((posiy >y1+50) && (posiy < y1+60) && (dcha) && (trigera))
 {
  aspa(x1+10,y1+50,x1+20,y1+60,0);
  aspa(x1+10,y1+70,x1+20,y1+80,7);
  bajaa=FALSE;
  subia=TRUE;
 }
 if((posiy >y1+70) && (posiy <y1+80) && (dcha) && (trigera))
 {
  aspa(x1+10,y1+50,x1+20,y1+60,7);
  aspa(x1+10,y1+70,x1+20,y1+80,0);
  subia=FALSE;
  bajaa=TRUE;
 }
 if((posiy > y1+90) && (posiy <y1+100) && (dcha) && (trigera) &&
    ((subia) || (bajaa)))
 {
  aspa(x1+10,y1+90,x1+20,y1+100,0);
  aspa(x1+10,y1+110,x1+20,y1+120,7);
  dicai=FALSE;
  dicad=TRUE;
 }
if((posiy > y1+110) && (posiy <y1+120) && (dcha) && (trigera) &&
   ((subia) || (bajaa)))
   {
  aspa(x1+10,y1+90,x1+20,y1+100,7);
  aspa(x1+10,y1+110,x1+20,y1+120,0);
  dicad=FALSE;
```

```
  dicai=TRUE;
 }
if((posiy >y1+130) && (posiy <y1+140) && (dcha) && (trigera) &&
   ((subia) || (bajaa)) && ((dicad) || (dicai)))
 {
  punteroff();
  if(!llave1)
    {
     cad2=escribe(x1+30,y1+35,x2-30,y2-35,4,"Tension Umbral","0");
     strcpy(cad3,cad2);
     llave1=TRUE;
    }
  else
    {
    cad2=escribe(x1+30,y1+35,x2-30,y2-35,4,"Tension Umbral",cad3);
    strcpy(cad3,cad2);
    }
  punteroon();
 }
/*numero de pretrigger*/
if((posiy >y1+5) && (posiy <y1+15) && (dcha))
 {
  punteroff();
  if(!llave0)
   {
    cad0=escribe(x1+30,y1+35,x2-30,y2-35,4,"Numero de pretrigger","0");
    strcpy(cad1,cad0);
    llave0=TRUE;
   }
  else
   {
    cad0=escribe(x1+30,y1+35,x2-30,y2-35,4,"Numero de pretrigger",cad1);
```

```
      strcpy(cad1,cad0);
     }
   punteroon();
    }

}
if((posix > x1+180) && (posix <x1+190))
  {
   if((posiy > y1+25) && (posiy < y1+35) && (dcha))
    {
     aspa(x1+180,y1+25,x1+190,y1+35,0);
     aspa(x1+180,y1+50,x1+190,y1+60,0);
     subid=TRUE;
     trigerd=TRUE;
    }
   if((posiy > y1+25) && (posiy < y1+35) && (izda))
    {
     aspa(x1+180,y1+25,x1+190,y1+35,7);
     aspa(x1+180,y1+50,x1+190,y1+60,7);
     aspa(x1+180,y1+110,x1+190,y1+120,7);
     subid=FALSE;
     bajad=FALSE;
     trigerd=FALSE;
    }

   if((posiy > y1+50) && (posiy < y1+60) && (dcha) && (trigerd))
   {
     aspa(x1+180,y1+50,x1+190,y1+60,0);
     aspa(x1+180,y1+110,x1+190,y1+120,7);
     bajad=FALSE;
     subid=TRUE;
   }
```

```
        if((posiy > y1+110) && (posiy < y1+120) && (dcha) && (trigerd))
        {
         aspa(x1+180,y1+50,x1+190,y1+60,7);
         aspa(x1+180,y1+110,x1+190,y1+120,0);
         subid=FALSE;
         bajad=TRUE;
        }

    }

    if(kbhit() && getch()==27) break; /* salir si pulsamos escape */
    if(salir) break;
      if((go) && (llave2)) break; /* si se han elegido todas las opciones podemos
llamar al procedimientro leer*/
  }
while(1);
 punteroff();
 _putimage(x1p,y1p,panta,_GPSET); /* repone el fondo */
 _putimage(x1,y1,panta1,_GPSET);
  hfree(panta1);  /* libera la memoria */
 hfree(panta);
 punteroon();
 if((go) && (llave2))
 leer(canal0,canal1,acoplo_dc,acoplo_ac,frecuencia,division,cad7,cad5,
        cad3,trigerd,subid,bajad,dicad,dicai,cad1,trigera,subia,bajaa);
/* llama a la función que se encarga de realizar la adquisición  */
 }

 void texto(int color,int x,int y, char *text)
 {
  _setcolor(color);
  _moveto(x,y);
```

```
 _outgtext(text);
 }

void aspa(int x1,int y1, int x2, int y2, int color) /* función que pone las aspas */
{
 punteroff();
 _setcolor(color);
 _moveto(x1,y1);
 _lineto(x2,y2);
 _moveto(x2,y1);
 _lineto(x1,y2);
 punteroon();
}

void menu1(int x1s,int y1s,int x2s,int y2s)
{
 int i;
 _setcolor(BLANCOBRILLANTE);
 _rectangle(_GFILLINTERIOR,x1s,y1s,x2s,y2s);
 _rectangle(2,x1s,y1s,x2s,y2s);
 _setcolor(7);
 _rectangle(_GFILLINTERIOR,x1s+2,y1s+2,x2s-2,y2s-2);
 _setcolor(0);
 _rectangle(2,x1s+2,y1s+2,x2s-2,y2s-2); /*rectangulo grande*/
 _ellipse(2,x1s+10,y1s+5,x1s+20,y1s+15);
 _ellipse(2,x1s+10,y1s+130,x1s+20,y1s+140);
 _rectangle(2,x1s+10,y1s+25,x1s+20,y1s+35);
 _rectangle(2,x1s+180,y1s+25,x1s+190,y1s+35);
 _rectangle(2,x1s+180,y1s+50,x1s+190,y1s+60);
 _rectangle(2,x1s+180,y1s+110,x1s+190,y1s+120);

 for(i=50;i<+121;i+=20)
```

```
 _rectangle(2,x1s+10,y1s+i,x1s+20,y1s+i+10);
_setcolor(NEGRO);
_moveto(x1s+2,y1s+20); /*primera linea horzontal*/
_lineto(x2s-2,y1s+20);

_moveto(x1s+2,y1s+40); /*segunda linea horizontal*/
_lineto(x2s-2,y1s+40);

_moveto(x1s+172,y1s+20); /*linea vertical*/
_lineto(x1s+172,y2s-2);


 texto(NEGRO,x1s+24,y1s+2,"Numero de Pretrigger <1000");
 texto(NEGRO,x1s+24,y1s+22,"Trigger Analogico");
 texto(NEGRO,x1s+24,y1s+47,"Rampa de Subida");
 texto(NEGRO,x1s+24,y1s+67,"Rampa de Bajada");
 texto(NEGRO,x1s+24,y1s+87,"Disp. Canal Dcho");
 texto(NEGRO,x1s+24,y1s+107,"Disp. Canal Izdo");
 texto(NEGRO,x1s+24,y1s+127,"Umbral");

 texto(NEGRO,x1s+194,y1s+22,"Triger Digital");
 texto(NEGRO,x1s+194,y1s+47,"Flanco de Subida");
 texto(NEGRO,x1s+194,y1s+107,"Flanco de Bajada");

}
```

```
/* PROGRAMA QUE ADQUIERE LOS DATOS Y LOS GUARDA EN EL
DISCO*/

typedef unsigned char BOOLEAN;

#include <stdio.h>
#include <malloc.h>
#include <graph.h>
#include <ctype.h>
#include <stdlib.h>
#include "nidaqerr.h"
#include "nidaqdsp.h"
#include <time.h>

#define AC 0
#define DC 1
#define SCANS 0
#define FRAMES 1
#define FRAMES_IN_BUF 60L /* INDICA 60 TRAMAS EN BUFFER*/
#define GET 0
#define TAP 1
#define FULL_CHECK 1
#define AT_DSP2200 18  /* tipo de tarjeta */
#define TRUE 1
#define FALSE 0
extern void ErrPrint(); /* fichero  de errores */
void boton(int,int,int,int, char *,BOOLEAN);
void ErrClean(char *,int , int ,int huge *,int huge *);
static int huge *getBuffer=NULL;
/* inicio */
void leer(BOOLEAN cana0,BOOLEAN cana1,BOOLEAN dc,BOOLEAN ac,int
frecu, unsigned int divi,char *cadena7, char *cadena5,char *cadena3,
```

```
        BOOLEAN digtri,BOOLEAN disub,BOOLEAN dibaja,BOOLEAN canad,
        BOOLEAN canai, char *cadena1,BOOLEAN analogotri, BOOLEAN anasubi,
        BOOLEAN anabajada)
{
 int getBoardToUse();
 clock_t inicio,final;
 int brd=1,
     div,
     boardCode,
       errNum,
     numChans,
     coupling[2],
     chanList[2],
     timebase,
     digTrigMode,
     anlTrigMode,
     anlTrigSlope,
     anlTrigLevel,
     anlTrigChan,
     digTrigEdge,
     posix,
     posiy,
     extMemUsed,
     acqDone,
     x1,
     y1,
     x2,
     y2,
     i,
     j,
     nTimebase;
```

```
static char huge *pantalla=NULL;
unsigned int
   intervalo,
   buf_ini,
   numcanales,
   pts_to_plot,
   xcount,
   x,
   y,
   k,
   frameToPlot;

unsigned long
   preTrigScans,
   postTrigScans,
   scansPerFrame,
   samplesPerFrame,
   framesDone,
   scansDone,
   numGotten,
   lastFrame,
   lastScan,

   extMemAddr;

double
   scanRate,
   posTrigDelaySecs,
   audioTimebase[4],
   duracion,
   tam;
```

```
char
   ch,
  *fin='\0',
   cr;
FILE *pf;

float
   z;
long
   min,
   max,
   tama_pantalla;
unsigned char color;
BOOLEAN stop,dcha,izda;

struct CABECERA  /* estructura para la cabecera */
{
 int nu_canales;
 unsigned int fm;
}cabeza;

/*---------------numero de canales y acoplamiento-------------------*/
if((cana0) && (!cana1))
 {
  numChans=1;
  chanList[0]=0;
 }
if((cana1) && (!cana0))
 {
  numChans=1;
  chanList[0]=1;
 }
```

```
 if((cana0) && (cana1))
  {
   numChans=2;
   chanList[0]=0;
   chanList[1]=1;
  }
 if((dc) && (!ac))
  {
   coupling[0]=DC;
   coupling[1]=DC;
  }
 if((ac) && (!dc))
  {
   coupling[0]=AC;
   coupling[1]=AC;
  }
errNum=Init_DA_Brds(brd,&boardCode);
ErrPrint("Init_DA_Brds",errNum);
  if(errNum!=0)
  {
      hfree(pantalla);
      hfree(getBuffer);
      return;
      exit(1);
   }
errNum=MAI_Coupling(brd,2,coupling);
ErrPrint("MAI_Coupling",errNum);
  if(errNum!=0)
  {
      hfree(pantalla);
      hfree(getBuffer);
      return;
```

```
        exit(1);
    }
errNum=MAI_Setup(brd,numChans,chanList,0L,0,0,0);
ErrPrint("MAI_Setup",errNum);
  if(errNum!=0)
  {
        hfree(pantalla);
        hfree(getBuffer);
        return;
        exit(1);
    }

/*----------------------------fin---------------------------------*/

/*----------------asignamos la frecuencia de muestreo----------------*/

intervalo=divi;
timebase=frecu;
errNum=MDAQ_ScanRate(brd,intervalo,timebase);
ErrPrint("MDAQ_ScanRate",errNum);
  if(errNum!=0)
  {
        hfree(pantalla);
        hfree(getBuffer);
        return;
        exit(1);
    }
/*----------------------------fin---------------------------------*/

/*-------------------------abrir el fichero----------------------*/
if( (pf=fopen(cadena7,"w+b"))==NULL)
  {
```

```
  printf("Error al abrir el fichero");
  exit(1);
  }
/*-------------------------fin-----------------------------------------------------------*/

/*----------------------ponemos la cabecera al programa-------------------------------*/
cabeza.nu_canales=numChans;
switch(frecu)
{
  case 0: cabeza.fm=(unsigned int)(51200/divi);
        break;
  case 1: cabeza.fm=(unsigned int)(48000/divi);
        break;
  case 2: cabeza.fm=(unsigned int)(44100/divi);
        break;
  case 3: cabeza.fm=(unsigned int)(32000/divi);
        break;
  default: cabeza.fm=(unsigned int)(32000/divi);
 }
if(cabeza.fm==5512)
  cabeza.fm=cabeza.fm+1;
 fwrite(&cabeza,sizeof(cabeza),1,pf);
 fseek(pf,4L,0);/*despazamiento desde el principio*/
 fread(&cabeza,sizeof(cabeza),1,pf);

/*-----------------------------fin--------------------------*/

/*---------determina la cantidad de datos pretigger y postrigger----*/

if((cabeza.nu_canales==1) && (cabeza.fm==51200))
      postTrigScans=12288;
else
```

```
      postTrigScans=14336; /*cada trama contiene 15000 muestras*/
            /*esto es una optimización para un PC 386 a 40 MHz */

preTrigScans=strtoul(cadena1,&fin,10);
scansPerFrame   =postTrigScans+preTrigScans;
samplesPerFrame =scansPerFrame*(unsigned long)numChans;
getBuffer=(int huge *) halloc(samplesPerFrame,sizeof(int));
if(getBuffer==NULL)
  {
   printf("No hay suficiente memoria");
   exit(1);
  }
/*-------------------------fin-----------------------------------------*/
/*-----------------Almacenamos los datos en memoria extendida---------*/
/*los datos seran almacenados en memoria extendida a partir de la direccion*/
/*0x00100000 es decir en el borde del segundo mega*/

extMemAddr=0x00100000;
/* reserva la memoria extendida */
errNum=MDAQ_Ext_Setup(brd,FRAMES_IN_BUF,FRAMES,preTrigScans,postTri
gScans,  extMemAddr);
ErrPrint("MDAQ_Ext_Setup",errNum);
  if(errNum!=0)
  {
     hfree(pantalla);
     hfree(getBuffer);
     return;
     exit(1);
  }
/*-------------------------fin-----------------------------------------*/

/*-----------------asigna los trigger hardware-----------------------*/
```

```
if(digtri)
 {
  digTrigMode=1;
  if(disub)
    {
    digTrigEdge=1;
    }
  if(dibaja)
    {
    digTrigEdge=0;
    }
 }
else
   digTrigMode=0;
if(analogotri)
{
 anlTrigMode=1;
 anlTrigLevel=atoi(cadena3);
 if(anasubi)
  anlTrigSlope=1;
 if(anabajada)
  anlTrigSlope=0;
 if(canad)
  anlTrigChan=0;
 if(canai)
  anlTrigChan=1;
 }
else
  anlTrigMode=0;
errNum=MDAQ_Trig_Select(brd,digTrigMode,digTrigEdge,anlTrigMode,
                        anlTrigSlope,anlTrigLevel,anlTrigChan);
```

```
ErrPrint("MDAQ_Trig_Select",errNum);
  if(errNum!=0)
  {
      hfree(pantalla);
      hfree(getBuffer);
      return;
      exit(1);
   }

intervalo=atoi(cadena5);
errNum=MDAQ_Trig_Delay(brd,intervalo,0);/*timebase es ignorado*/
ErrPrint("MDAQ_Trig_Delay",errNum);
  if(errNum!=0)
  {
      hfree(pantalla);
      hfree(getBuffer);
      return;
      exit(1);
   }

/*-------------------------------fin-----------------------------------*/

/*--------------------Dibuja la Ventana de Acqusicion-------------------*/
x1=170;
y1=190;
x2=470;
y2=290;
punteroff();
tama_pantalla=_imagesize(x1,y1,x2,y2);
pantalla=(char huge *) halloc(tama_pantalla,sizeof(char));
_getimage(x1,y1,x2,y2,pantalla);
```

```
_setcolor(15);
_rectangle(3,x1,y1,x2,y2);

_setcolor(7);
_rectangle(3,x1+2,y1+2,x2-2,y2-2);

_setcolor(0);
_rectangle(2,x1,y1,x2,y2);
_rectangle(2,x1+2,y1+2,x2-2,y2-2);

_moveto(x1+2,y1+18);
_lineto(x2-2,y1+18);

_moveto(x1+10,y1+2);
_outgtext("Fichero:");

_moveto(x1+80,y1+2);
_outgtext(cadena7);

_moveto(x1+10,y1+30);
_outgtext("TRAMAS");
_moveto(x1+100,y1+30);

_outgtext("TAMAÑO F.");
_moveto(x1+100,y1+50);
_outgtext("Megas");
_moveto(x2-90,y1+30);

_outgtext("TIEMPO");
_moveto(x2-90,y1+47);
_outgtext("Seg");
```

```
_setcolor(0);
_rectangle(3,x1+10,y1+47,x2-10,y1+66);

boton(x1+125,y1+70,x1+179,y1+88,"STOP",TRUE);

inicio=clock();

/*-----------------Modulo de acquisicion----------------------------*/
stop=FALSE;
errNum=MDAQ_Start(brd,0L);

ErrPrint("MDAQ_Start",errNum);
  if(errNum!=0)
  {
     hfree(pantalla);
     hfree(getBuffer);
     return;
     exit(1);
   }
j=1;
punteroon();

do  /* comienza la adquisción */
{
 leeraton(&posix,&posiy,&dcha,&izda);
errNum=MDAQ_Check(brd,FULL_CHECK,&acqDone,&framesDone,&scansDone);
 if(framesDone>=j)
 {
  errNum=MDAQ_Get(brd,FRAMES,GET,1L,0L,1L,0L,getBuffer,
                  &numGotten,&lastFrame,&lastScan,&acqDone);
  fwrite(getBuffer,sizeof(int),(int)samplesPerFrame,pf);
```

```
 j++; /* indica la adquisición de una trama */
 _settextposition(16,25);
 printf("%lu\r",lastFrame); /* pone el número tramas adquiridos */
 final=clock();
 duracion=(double)(final-inicio)/CLOCKS_PER_SEC;
 _settextposition(16,50);
 printf("%2.1f\r",duracion); /* pone la duración */
 tam=lastFrame*14336*2*numChans; /* calcula la longitud del dichero */
 _settextposition(16,35);
 printf("%.0lf\n",tam); /* indica el tamaño del fichero */
 if( (posix >x1+125) && (posix < x1+179))
  if((posiy > y1+70) &&  (posiy <y1+88) && (dcha))
  {
   while(dcha)
   {
    leeraton(&posix,&posiy,&dcha,&izda);
    boton(x1+125,y1+70,x1+179,y1+88,"STOP",FALSE);
   }
    boton(x1+125,y1+70,x1+179,y1+88,"STOP",TRUE);
    stop=TRUE;
  }
   /* sale si se pulsa ESC o el ratón */
  if(kbhit() && getch()==27) break;
 if(stop) break;
 }
 if(kbhit() && getch()==27) break;
 if(stop) break;
}
while(1);
ErrPrint("MDAQ_Get",errNum);
 if(errNum!=0)
 {
```

```
        hfree(pantalla); /* libera la memoria */
        hfree(getBuffer);
        return;
        exit(1);
    }

getch();
punteroff();
_putimage(x1,y1,pantalla,_GPSET); /* restaura el fondo de la pantalla */
hfree(pantalla);
hfree(getBuffer);
punteroon();
MDAQ_Clear(brd); /* limpia los buffer de adquisición */
fclose(pf);
}
/*-------------------------fin----------------------*/
```

```
/*PROGRAMA QUE PERMITE GENENERAR UN FICHERO DE FORMA
CONTROLADA*/

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned char BOOLEAN;

#include <malloc.h>
#include <ctype.h>
#include "nidaq.h"
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>
#include "colores.h"
#define SLOT 1
#define TRUE 1
#define FALSE 0
#define MAX 15
#define RELLENO 3
#define HUECO 2
extern void ErrPrint();
char * NuevoFile();
int ExisFile(char *);
char *escrib(int ,int, int);
char *escribe(int ,int, int,int,int, char *, char *);
void boton(int, int, int, int, char *,BOOLEAN);
void iniciar(BOOLEAN ,BOOLEAN,char *,double);

void gen_fiche(void)
{
 WORD
        posix,posiy,f;
```

```
BOOLEAN
            canal1,dcha,izda,canal0,aspa0,aspa1,salir,todo,go,llave2;
int
      X1p,X2p,Y1p,Y2p,y2s,y1s,color,s,i,x1p,x2p,y1p,y2p,resultado,contador=0;
static char huge *panta=NULL;
char *cad0,*cadena1,*cade1;
static char cad1[15]="datos.bin";
double
       frecuencia,division,frecu;
char
    *NomFile;
long tama_panta;
punteroff();
x1p=350;
y1p=65;
x2p=550;
y2p=320;
llave2=FALSE;

/* buffer donde se salvará la el fondo de la pantalla */
 tama_panta=_imagesize(x1p,y1p,x2p,y2p);
 panta=(char huge *) halloc(tama_panta,sizeof(char));
 _getimage(x1p,y1p,x2p,y2p,panta);

/* dibuja la carátula de control */
 _setcolor(15);
 _rectangle(_GFILLINTERIOR,x1p,y1p,x2p,y2p);
 _setcolor(7);
 _rectangle(_GFILLINTERIOR,x1p+2,y1p+2,x2p-2,y2p-2);
 _setcolor(0);
 _rectangle(_GBORDER,x1p,y1p,x2p,y2p);
```

```
_setcolor(0);
_rectangle(_GBORDER,x1p+2,y1p+2,x2p-2,y2p-2);

_setcolor(0);
_rectangle(_GBORDER,x1p+10,y1p+10,x1p+20,y1p+20);
_rectangle(_GBORDER,x1p+10,y1p+30,x1p+20,y1p+40);

_moveto(x1p+24,y1p+7);
_outgtext("Canal Derecho");
_moveto(x1p+24,y1p+27);
_outgtext("Canal Izquierdo");

_moveto(x1p+2,y1p+50);
_lineto(x2p-2,y1p+50);

_rectangle(2,x1p+10,y1p+54,x1p+20,y1p+64);
_moveto(x1p+24,y1p+50);
_outgtext("Nombre Del Fichero");

_moveto(x1p+2,y1p+67);
_lineto(x2p-2,y1p+67);

_moveto(x1p+14,y1p+67);
_outgtext("Frecuencia");
_moveto(x1p+124,y1p+67);
_outgtext("Division");

_moveto(x1p+2,y1p+84);
_lineto(x2p-2,y1p+84);

for(i=94;i<155;i+=20)
   _rectangle(2,x1p+10,y1p+i,x1p+20,y1p+i+10);
```

```
_moveto(x1p+24,y1p+91);
_outgtext("51.2 KHz");
_moveto(x1p+24,y1p+111);
_outgtext("48.0 KHz");
_moveto(x1p+24,y1p+131);
_outgtext("44.0 KHz");
_moveto(x1p+24,y1p+151);
_outgtext("32.0 KHz");

for(i=94;i<155;i+=20)
_rectangle(2,x1p+120,y1p+i,x1p+130,y1p+i+10);
_moveto(x1p+134,y1p+91);
_outgtext("x 1");
_moveto(x1p+134,y1p+111);
_outgtext("x 2");
_moveto(x1p+134,y1p+131);
_outgtext("x 4");
_moveto(x1p+134,y1p+151);
_outgtext("x 8");

_moveto(x1p+2,y1p+175);
_lineto(x2p-2,y1p+175);

_moveto(x1p+110,y1p+67);
_lineto(x1p+110,y1p+175);

 boton(x1p+100,y1p+224,x1p+150,y1p+241,"EXIT",TRUE);
 boton(x1p+30,y1p+224,x1p+80,y1p+241,"GO !",TRUE);
X1p=360;
Y1p=75;
X2p=370;
Y2p=85;
```

```
y1s=105;
y2s=115;
aspa0=FALSE;
aspa1=FALSE;
salir=FALSE;
canal0=FALSE;
canal1=FALSE;
go=FALSE;
punteroon();

do /* bucle que lee el estado del ratón y nos permite seleccionar las distintas opciones
*/
{
 leeraton(&posix,&posiy,&dcha,&izda);
 if((posix > x1p+10) && (posix < x1p+20))
  {
   if((posiy > y1p+10) && (posiy < y1p+20) && (!aspa0) && (dcha))
     {
       color=0;
       aspa(x1p+10,y1p+10,x1p+20,y1p+20,color);
       canal0=TRUE;
       aspa0=TRUE;
     }
   if((posiy > y1p+10) && (posiy < y1p+20) && (aspa0) && (izda))
     {
       color=7;
       aspa(x1p+10,y1p+10,x1p+20,y1p+20,color);
       canal0=FALSE;
       aspa0=FALSE;
     }

   if ((posiy > y1p+30) && (posiy < y1p+40) && (dcha) && (!aspa1))
```

```
 {
  color=0;
  aspa(x1p+10,y1p+30,x1p+20,y1p+40,color);
  canal1=TRUE;
  aspa1=TRUE;
 }

if((posiy > y1p+30) && (posiy < y1p+40) && (aspa1) && (izda))
 {
  color=7;
  aspa(x1p+10,y1p+30,x1p+20,y1p+40,color);
  canal1=FALSE;
  aspa1=FALSE;
 }
if((posiy >y1p+54) && (posiy <y1p+64) && (dcha))
 {
  punteroff();
  do
   {
    cad0=escribe(x1p-30,y1p+90,x2p+30,y2p-90,15,"Nombre del fichero","");
    resultado=ExisFile(cad0);
    if(resultado==0) /*cuando resultado=0 el fichero no existe*/
      {
       titulos(4,455,635,475,BLANCO);
       Cuadrado(RELLENO,5,455,634,475,BLANCO);
       PoneCadena(8,455,NEGRO,"EL FICHERO NO EXISTE");
       contador++;
      }
   }
   while(resultado==0);
   if(contador>0)f(resultado==0) /*cuando resultado=0 el fichero no existe*/
     Cuadrado(RELLENO,5,455,634,475,BLANCO);
```

```
    llave2=TRUE;
    strcpy(cad1,cad0);
            punteroon();
  }
if((posiy >y1p+94) && (posiy < y1p+104) && (dcha))
  {
   aspa(x1p+10,y1p+94,x1p+20,y1p+104,0);
   aspa(x1p+10,y1p+114,x1p+20,y1p+124,7);
   aspa(x1p+10,y1p+134,x1p+20,y1p+144,7);
   aspa(x1p+10,y1p+154,x1p+20,y1p+164,7);
   frecuencia=51.2;  /* se selecciona la frecuencia de 51.200 */
   }
 if ((posiy >y1p+114) && (posiy <y1p+124) && (dcha))
  {
   aspa(x1p+10,y1p+94,x1p+20,y1p+104,7);
   aspa(x1p+10,y1p+114,x1p+20,y1p+124,0);
   aspa(x1p+10,y1p+134,x1p+20,y1p+144,7);
   aspa(x1p+10,y1p+154,x1p+20,y1p+164,7);
   frecuencia=48.0; /* se seleciona la frecuencia de 48.0 */
  }
 if((posiy >y1p+134) && (posiy < y1p+144) && (dcha))
  {
   aspa(x1p+10,y1p+94,x1p+20,y1p+104,7);
   aspa(x1p+10,y1p+114,x1p+20,y1p+124,7);
   aspa(x1p+10,y1p+134,x1p+20,y1p+144,0);
   aspa(x1p+10,y1p+154,x1p+20,y1p+164,7);
   frecuencia=44.1; /* se selecciona la frecuencia de 44.1 */
  }
 if((posiy >y1p+154) && (posiy < y1p+164) && (dcha))
  {
   aspa(x1p+10,y1p+94,x1p+20,y1p+104,7);
   aspa(x1p+10,y1p+114,x1p+20,y1p+124,7);
```

```
    aspa(x1p+10,y1p+134,x1p+20,y1p+144,7);
    aspa(x1p+10,y1p+154,x1p+20,y1p+164,0);
    frecuencia=32.0; /* se selecciona la frecuencia de 32.0 */
   }
}
if((posix > x1p+120) && (posix <x1p+130))
 {
  if((posiy >y1p+94) && (posiy < y1p+104) && (dcha))
   {
    aspa(x1p+120,y1p+94,x1p+130,y1p+104,0);
    aspa(x1p+120,y1p+114,x1p+130,y1p+124,7);
    aspa(x1p+120,y1p+134,x1p+130,y1p+144,7);
    aspa(x1p+120,y1p+154,x1p+130,y1p+164,7);
    division=1.0;  /* división por uno */
    }
  if ((posiy >y1p+114) && (posiy <y1p+124) && (dcha))
   {
    aspa(x1p+120,y1p+94,x1p+130,y1p+104,7);
    aspa(x1p+120,y1p+114,x1p+130,y1p+124,0);
    aspa(x1p+120,y1p+134,x1p+130,y1p+144,7);
    aspa(x1p+120,y1p+154,x1p+130,y1p+164,7);
    division=2.0;  /* división por dos */
   }
  if((posiy >y1p+134) && (posiy < y1p+144) && (dcha))
   {
    aspa(x1p+120,y1p+94,x1p+130,y1p+104,7);
    aspa(x1p+120,y1p+114,x1p+130,y1p+124,7);
    aspa(x1p+120,y1p+134,x1p+130,y1p+144,0);
    aspa(x1p+120,y1p+154,x1p+130,y1p+164,7);
    division=4.0; /* división por cuatro */
   }
  if((posiy >y1p+154) && (posiy < y1p+164) && (dcha))
```

```
    {
     aspa(x1p+120,y1p+94,x1p+130,y1p+104,7);
     aspa(x1p+120,y1p+114,x1p+130,y1p+124,7);
     aspa(x1p+120,y1p+134,x1p+130,y1p+144,7);
     aspa(x1p+120,y1p+154,x1p+130,y1p+164,0);
     division=8.0; /* división por ocho */
    }
 }
if ( (posix > x1p+100) && (posix < x1p+150))
 if ((posiy > y1p+224) && (posiy < y1p+241) && (dcha))
  {
   while(dcha)
   {
    leeraton(&posix,&posiy,&dcha,&izda);
    boton(x1p+100,y1p+224,x1p+150,y1p+241,"EXIT",FALSE);
   }
   boton(x1p+100,y1p+224,x1p+150,y1p+241,"EXIT",TRUE);
   salir=TRUE; /* se pulsa el botón salir */
  }
 if((posix > x1p+30) && (posix < x1p+80))
  if ((posiy > y1p+224) && (posiy <y1p+241) && (dcha))
   {
    while(dcha)
     {
      leeraton(&posix,&posiy,&dcha,&izda);
       boton(379,289,429,306,"GO !",FALSE);
       boton(x1p+30,y1p+224,x1p+80,y1p+241,"GO !",FALSE);
     }
    boton(x1p+30,y1p+224,x1p+80,y1p+241,"GO !",TRUE);
    go=TRUE; /* se pulsa el botón generar */
   }
```

```
    if (kbhit() && getch()==27) break; /* se se pulsa ESC salir */
    if(salir) break; /* si se selecciona salir se sale */
    if(go) break;
  }
 while(1);

 frecu=((frecuencia/division)*1000);
 punteroff();
 _putimage(350,65,panta,_GPSET);
 hfree(panta);
 if(go) iniciar(canal0,canal1,cad1,frecu); /* si hemos seleccionado GO se llama
reproducir */
 punteroon();
}

/* función reproducir */
void iniciar(BOOLEAN cana0, BOOLEAN cana1, char *cad0, double frecuencia)
{
 int   numChans,
    chanVect[2],
      boardCode,
        status;
 unsigned long startPt,
              endPt,
                i,
         iteraciones;
 double rate;

 if(cana0)
    {
     numChans=1;
     chanVect[0]=0;
```

```
    }
  if(cana1)
    {
     numChans=1;
     chanVect[1]=1;
    }
   if((cana0) && (cana1))
    {
     numChans=2;
     chanVect[0]=0;
     chanVect[1]=1;
     }
  startPt=1;/*comenzamos a leer despues de la cabecera*/
  endPt=0;
  iteraciones=1;
  rate=frecuencia;
  status=Init_DA_Brds(SLOT,&boardCode);
  ErrPrint("Init_DA_Brds",status);
/* función que genera los datos */
status=WFM_from_Disk(SLOT,numChans,chanVect,cad0,startPt,endPt,iteraciones,r
ate);
  ErrPrint("WFM_from_Disk",status);
  status=Init_DA_Brds(SLOT,&boardCode);
  ErrPrint("Init_DA_Brds",status);
}
```

```
/* RUTINA PARA REPRODUCIR RAPIDAMENTE UN FICHERO*/

#include <graph.h>
#include "colores.h"
#include <fcntl.h>
#include <stdio.h>
#include "nidaq.h"
#include <io.h>
#include <dos.h>

#define RELLENO 3

char *NuevoFile(void);
int ExisFile(char *);
extern void ErrPrint();
char *escribe(int,int,int,int,int,char *,char *);

void repro_rapida()
{
 int
      estado,fh,chanVect[2],CodigoPlaca,resultado;
 struct /* estructura que contiene la cabecera */
 {
  int nu_canales;
  unsigned int fm;
  }cabecera;

  char *nom_fiche;
  punteroff();
  punteroon();
  do
   {
```

```
 nom_fiche=NuevoFile();
 if(*nom_fiche=='\0')
   {
    return;
   }
 resultado=ExisFile(nom_fiche);
 if(resultado==0)
    {
     titulos(4,455,635,475,BLANCO);
     Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
     PoneCadena(8,455,NEGRO,"EL FICHERO NO EXISTE");
    }
 }
while(resultado==0);
Cuadrado(RELLENO,5,455,635,475,BLANCO);
PoneCadena(5,457,NEGRO,"ESPERE POR FAVOR");
 estado=Init_DA_Brds(1,&CodigoPlaca);
ErrPrint("Init_DA_Brds",estado);
/* abre le fichero para leer la cabecera */
if((fh=open(nom_fiche,O_RDONLY,O_BINARY))<0)
{
   _setvideomode(3);
   printf("Ha ocurrido un error en la aplicaci¢n");
   getch();
   exit(1);
 }
read(fh,&cabecera,4); /* lee la cabecera */
close(fh);
if(cabecera.nu_canales==1)
chanVect[0]=0;
if(cabecera.nu_canales==2)
{
```

```
  chanVect[0]=0;
  chanVect[1]=1;
 }
/* rutina que reproduce el fichero */
estado=WFM_from_Disk(1,cabecera.nu_canales,chanVect,nom_fiche,3L,0L,1,(doubl
e)cabecera.fm);
 ErrPrint("WFM_fron_Disk",estado);
 estado=Init_DA_Brds(1,&CodigoPlaca);
 ErrPrint("Init_DA_Brds",estado);
 Cuadrado(RELLENO,5,455,635,475,AZULCLARO);
}
```

```
/* PANTALLA DE INTRODUCCION DE DATOS PARA CALCULAR EL
FILTRO */
/* DE AQUI SE PASAN LOS DATOS A LOS PROCEDIMIENTOS DE
FILTRADO Y REPRESENTACIÓN */

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned char BOOLEAN;
#include <stdio.h>
#include <graph.h>
#include <ctype.h>
#include <dos.h>
#include "colores.h"
#include <stdlib.h> /* la librerias hay que ponerlas porque sino la conversión */
                    /* entre una cadena y un float no se realiza de forma adecuada */
#define TRUE 1
#define FALSE 0
#define RELLENO 3
#define HUECO  2

/* prototipos de las funciones */
char *dato(int,int,int);
void boton(int, int, int, int, char *, BOOLEAN);
void boto1(int,int,int,int,int,char *);
void PoneCadena(int, int, int, char * );
void CuaDrado(int,int,int,int,int,int);
void DibujaFlecha(int, int, int,int);
void coeficientes(float,float,float,int,int,int,float,float,int);
void desactivaMenu(int);
void titulos(int,int,int,int,int);

void dsp_filtro(int TipoFil )
```

```
{
 WORD posix,posiy;
 BOOLEAN dcha,izda,sw1,sw2,sw3,sw4,sw5,go,exit,sw6,sw7;
 int x1=0,y1=301,x2=639,y2=439,tipo,orden,tama_array,orden1;
 char *tabla[16]={"4000","5513","6000","6400","8000","11025","12000","12800",
          "16000","22050","24000","25600","32000","44100","48000","51200"  };
 char *car,tecla0;
 double in1=0,in2=0,in5,in6;
 int ind=0;
 long i;
 int j,in3=0,in4=-1;
 double fm,f_low,f_hi,rizado,atenuacion;
 char patron[]={0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};
 char *fs;
  punteroff();
 desactivaMenu(1); /*desactiva el menu de edici¢n*/
_setcolor(AZULCLARO);
_rectangle(3,0,16,639,300);
_setcolor(MARRON); /*borramos lo anterior dibujando un rectangulo marron*/
_rectangle(_GFILLINTERIOR,x1,y1,x2,y2+1);

_setcolor(ROJO); /*borde rojo*/
_rectangle(_GBORDER,x1,y1,x2,y2);
 punteroon();

 sw1=FALSE;
 sw2=FALSE;
 sw3=FALSE; /*inicializo las llaves*/
 sw4=FALSE;
 sw5=FALSE;
 sw6=FALSE;
 sw7=FALSE;
```

```
go=FALSE;
exit=FALSE;
 boton (x1+400,y1+96,x1+450,y1+113,"IR !",TRUE);
 boton (x1+500,y1+96,x1+550,y1+113,"EXIT",TRUE);
 titulos(x1+5,y1+145,x1+635,y1+172,BLANCOBRILLANTE); /*recuadro de
                                                        ayudas*/
  titulos(x1+24,y1+24,x1+276,y1+36,BLANCOBRILLANTE);
  PoneCadena(x1+26,y1+22+1,0,"Frecuencia de Muestreo:");
  titulos(x1+24,y1+60,x1+311,y1+72,BLANCOBRILLANTE);
  PoneCadena(x1+26,y1+59,0,"Frecuencia de Corte Inferior:");
  titulos(x1+24,y1+96,x1+316,y1+108,BLANCOBRILLANTE);
   PoneCadena(x1+26,y1+94,0,"Frecuencia de Corte Superior:");
   titulos(x1+404,y1+9,x1+576,y1+21,BLANCOBRILLANTE);
   PoneCadena(x1+406,y1+8,0,"Orden del Filtro:");
   titulos(x1+404,y1+39,x1+466,y1+51,BLANCOBRILLANTE);
    PoneCadena(x1+406,y1+37,0,"Tipo:");

 if((TipoFil==1) || (TipoFil==2)) /* pone esta cadena sólo si se se ha elejido */
                                  /* el filtro eliptico o el cheby */
 {
    titulos(x1+494,y1+39,x1+606,y1+51,BLANCOBRILLANTE);
    PoneCadena(x1+496,y1+38,0,"Rizado:");
  }
  if(TipoFil==2)
  {
    titulos(x1+404,y1+69,x1+551,y1+81,BLANCOBRILLANTE);
    PoneCadena(x1+406,y1+68,0,"Atenuación:");
  }

 if(TipoFil==0)
 {
  sw6=TRUE;
```

```
   sw7=TRUE;
}
if(TipoFil==1)
    sw7=TRUE;
 _setcolor(0);
 boto1(x1+282,y1+19,x1+294,y1+29,1,"");
 boto1(x1+282,y1+31,x1+294,y1+41,1,"");
  _setcolor(0);
  DibujaFlecha(x1+288,x1+288,y1+21,1);  /* dibuja las flecahas */
  DibujaFlecha(x1+288,x1+288,y1+38,0);
   boto1(x1+317,y1+55,x1+329,y1+65,1,"");
   boto1(x1+317,y1+67,x1+329,y1+77,1,"");
  _setcolor(0);
  DibujaFlecha(x1+323,x1+323,y1+57,1);
  DibujaFlecha(x1+323,x1+323,y1+74,0);
  boto1(x1+322,y1+91,x1+334,y1+101,1,"");
  boto1(x1+322,y1+103,x1+334,y1+113,1,"");
  _setcolor(0);
  DibujaFlecha(x1+328,x1+328,y1+93,1);
  DibujaFlecha(x1+328,x1+328,y1+110,0);

  boto1(x1+582,y1+4,x1+594,y1+14,1,"");
  boto1(x1+582,y1+16,x1+594,y1+26,1,"");

  _setcolor(0);
  DibujaFlecha(x1+588,x1+588,y1+6,1);
  DibujaFlecha(x1+588,x1+588,y1+23,0);

  boto1(x1+472,y1+34,x1+484,y1+44,1,"");
  boto1(x1+472,y1+46,x1+484,y1+56,1,"");

  _setcolor(0);
```

```
      DibujaFlecha(x1+478,x1+478,y1+36,1);
      DibujaFlecha(x1+478,x1+478,y1+53,0);

      if((TipoFil==1) || (TipoFil==2))
      {
         boto1(x1+612,y1+34,x1+624,y1+44,1,"");
         boto1(x1+612,y1+46,x1+624,y1+56,1,"");
         _setcolor(0);
          DibujaFlecha(x1+618,x1+618,y1+36,1);
          DibujaFlecha(x1+618,x1+618,y1+53,0);
       }
      if(TipoFil==2)
      {
        boto1(x1+557,y1+64,x1+569,y1+74,1,"");
        boto1(x1+557,y1+76,x1+569,y1+86,1,"");
        _setcolor(0);
        DibujaFlecha(x1+563,x1+563,y1+66,1);
        DibujaFlecha(x1+563,x1+563,y1+83,0);
      }

  do /*seleccionamos los lor apartados */
  {
   leeraton(&posix,&posiy,&dcha,&izda);

      do
      {
         leeraton(&posix,&posiy,&dcha,&izda);
          if((posix > x1+282) && (posix < x1+294))
            if((posiy>y1+19) && (posiy<y1+29) && (dcha)
               {
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
```

```
            PoneCadena(x1+7,y1+150,ROJO,"Indica la frecuencia de muestreo");

            _moveto(x1+219,y1+22);
            if(ind<16)
            {



CuaDrado(RELLENO,x1+219,y1+25,x1+274,y1+35,BLANCOBRILLANTE);

            _setcolor(0);
            _outgtext(tabla[ind++]);
            fm=atof(tabla[ind-1]);


             for(i=0;i<320000;i++)
             {
              }
            }
            else
            ind=0;

            sw1=TRUE;
      }
     if((posix > x1+282) && (posix < x1+294))
        if((posiy>y1+31) && (posiy<y1+41) && (dcha))
        {



CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
            PoneCadena(x1+7,y1+150,ROJO,"Indica la frecuencia de muestreo");
```

```
                _moveto(x1+219,y1+22);
                if(ind>0)
                   {
CuaDrado(RELLENO,x1+219,y1+25,x1+274,y1+35,BLANCOBRILLANTE);
                    _setcolor(0);
                    _outgtext(tabla[--ind]);
                    fm=atof(tabla[ind]);
                     for(i=0;i<320000;i++)
                            { /* bucle de retardo para el avance de los números*/
                            }
                   }
                 else
                  ind=16;
                  sw1=TRUE;
                }
        if((posix > x1+317) && (posix < x1+328))
           if((posiy>y1+55) && (posiy<y1+65) && (dcha) )  /*FRECUENCIA DE
                                                          CORTE BAJA*/

             {
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
            PoneCadena(x1+7,y1+150,ROJO,"Indica la frecuencia inferior de corte
                                                  del filtro (f_baja<=fs/2)");

                _moveto(x1+253,y1+58);
                if(in1<51200)
                    {
CuaDrado(RELLENO,x1+249,y1+61,x1+309,y1+72,BLANCOBRILLANTE);
                  _setcolor(0);
                  gcvt(in1+=0.5,6,car);
                   f_low=in1;
```

```
                    _outgtext(car);
                     for(i=0;i<40000;i++) /*retardo de avance de los numeros*/
                       {
                       }
            }
            else
             in1=0;
             sw2=TRUE;
      }
     if((posix > x1+317) && (posix < x1+328))
        if((posiy>y1+67) && (posiy<y1+77) && (dcha))
        {
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
            PoneCadena(x1+7,y1+150,ROJO,"Indica la frecuencia inferior de corte
del filtro (f_baja<=fs/2)");
                _moveto(x1+253,y1+58);
                 if(in1<0.1)
                 in1=0.0;
                if(in1>0)
                   {
CuaDrado(RELLENO,x1+249,y1+60,x1+309,y1+72,BLANCOBRILLANTE);
                      _setcolor(0);
                       gcvt(in1-=0.5,6,car);
                       f_low=in1;
                       _outgtext(car);
                      for(i=0;i<40000;i++)
                            {
                            }
            }
            else
             in1=0;
            sw2=TRUE;
```

```
        }
    if((posix > x1+322) && (posix < x1+333))
        if((posiy>y1+91) && (posiy<y1+101) && (dcha) ) /*FRECUENCIA DE
CORTE ALTA*/
    {
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
            PoneCadena(x1+7,y1+150,ROJO,"Indica la frecuencia superior de corte
del filtro (f_alta<=fs/2)");

              _moveto(x1+259,y1+94);
              if(in2<51200)
                    {
CuaDrado(RELLENO,x1+255,y1+96,x1+314,y1+105,BLANCOBRILLANTE);
                    _setcolor(0);
                    gcvt(in2+=0.5,6,car);
                    f_hi=in2;
                    _outgtext(car);
                   for(i=0;i<40000;i++)
                          {
                          }
            }
            else
            in2=0;
             sw3=TRUE;
    }
    if((posix > x1+322) && (posix < x1+333))
        if((posiy>y1+103) && (posiy<y1+113) && (dcha))
        {
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
            PoneCadena(x1+7,y1+150,ROJO,"Indica la frecuencia superior de corte
del filtro (f_alta<=fs/2)");
```

```
            _moveto(x1+259,y1+94);
            if(in2<0.1)
                  in2=0.0;

            if(in2>0)
            {
CuaDrado(RELLENO,x1+255,y1+96,x1+314,y1+107,BLANCOBRILLANTE);
             _setcolor(0);
             gcvt(in2-=0.5,6,car);
             f_hi=in2;
             _outgtext(car);
              for(i=0;i<40000;i++) /*retardo */
                        {
                        }
            }
            else
            in2=0;
            sw3=TRUE;
          }
      if((posix > x1+582) && (posix < x1+592))
         if((posiy>y1+4) && (posiy<y1+14) && (dcha) )  /*ORDEN DEL FILTRO*/
               {
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
             PoneCadena(x1+7,y1+150,ROJO,"Indica el orden del filtro debe ser
mayor de CERO y menor de CIEN");
             _moveto(x1+544,y1+7);
               if(in3<50)
                {
CuaDrado(RELLENO,x1+540,y1+9,x1+576,y1+20,BLANCOBRILLANTE);
                 _setcolor(0);
                 in3++;
                 itoa(in3,car,10);
```

```
                    _outgtext(car);
                    orden=in3;
                  for(i=0;i<200000;i++)
                        {
                         }
            }
            else
            in3=0;
            sw4=TRUE;
      }
     if((posix > x1+582) && (posix < x1+592))
        if((posiy>y1+16) && (posiy<y1+26) && (dcha))
        {
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
          PoneCadena(x1+7,y1+150,ROJO,"Indica el orden del filtro debe ser mayor
de CERO y menor de CIEN");
          _moveto(x1+544,y1+7);
          if(in3>1)
          {
CuaDrado(RELLENO,x1+540,y1+9,x1+576,y1+20,BLANCOBRILLANTE);
          _setcolor(0);
           --in3;
           orden=in3;
           itoa(in3,car,10);
           _outgtext(car);
            for(i=0;i<200000;i++)
            {
             }
     }
          else
          in3=1;
          sw4=TRUE;
```

```
        }
    if((posix > x1+472) && (posix < x1+481))
       if((posiy>y1+34) && (posiy<y1+44) && (dcha) ) /*Tipo de filtro*/
      {
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
          PoneCadena(x1+7,y1+150,ROJO,"Indica el tipo de filtro,0:Lowpass
1:Highpass 2:Bandpass 3:Bandstop");
          _moveto(x1+454,y1+37);
          if(in4<3)
             {
CuaDrado(RELLENO,x1+450,y1+39,x1+466,y1+50,BLANCOBRILLANTE);
             _setcolor(0);
              ++in4;
             itoa(in4,car,10);
             _outgtext(car);
              tipo=in4;
           for(i=0;i<300000;i++)
           {
            }
           }
           else
           in4==1;
          sw5=TRUE;
    }
    if((posix > x1+472) && (posix < x1+481))
       if((posiy>y1+46) && (posiy<y1+56) && (dcha))
       {
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
          PoneCadena(x1+7,y1+150,ROJO,"Indica el tipo de filtro,0:Lowpass
1:Highpass 2:Bandpass 3:Bandstop");
          _moveto(x1+454,y1+37);
          if(in4>0)
```

```
            {
CuaDrado(RELLENO,x1+450,y1+39,x1+466,y1+50,BLANCOBRILLANTE);
                    _setcolor(0);
                    --in4;
                    tipo=in4;
                    itoa(in4,car,10);
                    _outgtext(car);
                    for(i=0;i<300000;i++)
                       {
                       }
            }
            else
            in4=0;
            sw5=TRUE;
          }
      if((posix > x1+612) && (posix < x1+621))
         if((posiy>y1+34) && (posiy<y1+44) && (dcha) && ((TipoFil==1) ||
(TipoFil==2)))  /*rizado*/
      {
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
            PoneCadena(x1+7,y1+150,ROJO,"Indica el rizado en dBs de la banda de
paso 0 < Rizado < Atenuaci¢n");
            _moveto(x1+558,y1+37);
            if(in5<100)
            {
CuaDrado(RELLENO,x1+557,y1+39,x1+596,y1+50,BLANCOBRILLANTE);
                    _setcolor(0);
                    gcvt(in5+=0.5,3,car);
                    rizado=in5;
                    _outgtext(car);
                    for(i=0;i<100000;i++) /*retardo de avance de los numeros*/
                       {
```

```
                        }
            }
            else
              in5=0;
          sw6=TRUE;
      }
     if((posix > x1+612) && (posix < x1+621))
        if((posiy>y1+46) && (posiy<y1+56) && (dcha) && ((TipoFil==1) ||
(TipoFil==2)))
          {
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
            PoneCadena(x1+7,y1+150,ROJO,"Indica el rizado en dBs de la banda de
paso 0 < Rizado < Atenuación");
            _moveto(x1+558,y1+37);
            if(in5<0.1)
                 in5=0.0;
            if(in5>0)
            {
CuaDrado(RELLENO,x1+557,y1+39,x1+596,y1+50,BLANCOBRILLANTE);
            _setcolor(0);
             gcvt(in5-=0.5,3,car); /*incremento de 0.5*/
             rizado=in5;
             _outgtext(car);
              for(i=0;i<100000;i++)
                {
                }
      }
           else
            in5=0;
         sw6=TRUE;
        }
```

```
      if((posix > x1+557) && (posix < x1+566))
         if((posiy>y1+64) && (posiy<y1+74) && (dcha) &&  (TipoFil==2))
/*atenuación*/
      {
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
            PoneCadena(x1+7,y1+150,ROJO,"Indica la atenuación en dBs
Atenuación > 0");
            _moveto(x1+504,y1+67);
            if(in6<100)
            {
CuaDrado(RELLENO,x1+500,y1+69,x1+551,y1+80,BLANCOBRILLANTE);
            _setcolor(0);
             gcvt(in6+=0.5,3,car);
             atenuacion=in6;
            _outgtext(car);
             for(i=0;i<100000;i++) /*retardo de avance de los numeros*/
             {
              }
      }
            else
             in6=0;
          sw7=TRUE;
      }
     if((posix > x1+557) && (posix < x1+566))
        if((posiy>y1+76) && (posiy<y1+86) && (dcha) && (TipoFil==2))
        {
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
          PoneCadena(x1+7,y1+150,ROJO,"Indica la atenuación en dBs Atenuación
> 0");
          _moveto(x1+504,y1+67);
          if(in6<0.1)
               in6=0.0;
```

```
            if(in6>0)
            {
CuaDrado(RELLENO,x1+500,y1+69,x1+551,y1+80,BLANCOBRILLANTE);
                _setcolor(0);
                gcvt(in6-=0.5,3,car); /*incremento de 0.5*/
               atenuacion=in6;
                _outgtext(car);
                for(i=0;i<100000;i++)
                        {
                        }
           }
            else
             in6=0;
           sw7=TRUE;
          }
    if((posix > x1+220) && (posix < x1+276))      /*frecuecia de muestreo*/
      if((posiy>y1+24) && (posiy<y1+36) && (dcha))
       break;
    if((posix > x1+260) && (posix < x1+311))      /*frecuencia baja*/
      if((posiy>y1+60) && (posiy<y1+72) && (dcha))
       break;
    if((posix > x1+260) && (posix < x1+316))      /*frecuencia alta*/
      if((posiy>y1+92) && (posiy<y1+112) && (dcha))
       break;
    if((posix > x1+470) && (posix < x1+580))      /*orden del filtro*/
       if((posiy>y1+9) && (posiy<y1+21) && (dcha))
        break;

    if((posix > x1+450) && (posix < x1+470))      /*tipo recuadro*/
      if((posiy>y1+39) && (posiy<y1+51) && (dcha))
      break;
    if((posix > x1+500) && (posix < x1+551))      /*recuadro  atenuaci¢n*/
```

```
        if((posiy>y1+64) && (posiy<y1+81) && (dcha) && (TipoFil==2))
         break;
      if((posix > x1+550) && (posix < x1+606))      /* recuadro de rizado*/
        if((posiy>y1+39) && (posiy<y1+51) && (dcha) && ((TipoFil==1) ||
(TipoFil==2)))
        break;

      if((posix > x1+500) && (posix < x1+550) )
        if((posiy>y1+96) && (posiy<y1+113) && (dcha))    /*boton salir*/
         break;

      if((posix > x1+400) && (posix < x1+450) && (sw1) && (sw2) && (sw3) &&
(sw4) && (sw5) && (sw6) && (sw7) )
             if((posiy>y1+96) && (posiy<y1+113) && (dcha))    /*boton go*/
             break;
            }
       while(1);  /*del while*/
if((posix > x1+220) && (posix < x1+276))      /*primer recuadro*/
    if((posiy>y1+24) && (posiy<y1+36) && (dcha))
    {
       punteroff();
       CuaDrado(HUECO,x1+19,y1+18,x1+281,y1+43,NEGRO);
       CuaDrado(RELLENO,x1+219,y1+25,x1+273,y1+36,BLANCOBRILLANTE);
       CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
       PoneCadena(x1+7,y1+150,ROJO,"Indica la frecuencia de muestreo");
       fs=dato(219,22,5);
       CuaDrado(HUECO,x1+19,y1+18,x1+281,y1+43,MARRON);
       CuaDrado(RELLENO,x1+215,y1+24,x1+273,y1+36,BLANCOBRILLANTE);
       _setcolor(0); /*pone numeros*/
       _moveto(x1+219,y1+22);
       _outgtext(fs);
```

```
        fm=atof(fs);
        CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
        sw1=TRUE;
        punteroon();
      }
          if((posix > x1+260) && (posix < x1+311))      /*segundo recuadro*/
        if((posiy>y1+60) && (posiy<y1+72) && (dcha))
        {
         punteroff();
         CuaDrado(HUECO,x1+19,y1+54,x1+316,y1+79,NEGRO);
         CuaDrado(RELLENO,x1+249,y1+61,x1+308,y1+72,BLANCOBRILLANTE);
         CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
         PoneCadena(x1+7,y1+150,ROJO,"Indica la frecuencia inferior de corte del
filtro (f_baja<=fs/2)");
         fs=dato(253,58,5);
         CuaDrado(HUECO,x1+19,y1+54,x1+316,y1+79,MARRON);
         CuaDrado(RELLENO,x1+249,y1+61,x1+308,y1+72,BLANCOBRILLANTE);

         _setcolor(0);  /*posición del texto*/
         _moveto(x1+253,y1+58);
         _outgtext(fs);
         f_low=atof(fs);
         CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
         sw2=TRUE;
         punteroon();
        }
      if((posix > x1+260) && (posix < x1+316))
        if((posiy>y1+96) && (posiy<y1+108) && (dcha))   /*tercer recuadro*/
        {
         punteroff();
```

```
CuaDrado(HUECO,x1+19,y1+90,x1+321,y1+115,NEGRO);
CuaDrado(RELLENO,x1+255,y1+96,x1+313,y1+106,BLANCOBRILLANTE);
CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
        PoneCadena(x1+7,y1+150,ROJO,"Indica la frecuencia superior de corte del
filtro (f_alta<=fs/2)");
        fs=dato(259,94,5);

 CuaDrado(HUECO,x1+19,y1+90,x1+321,y1+115,MARRON);
CuaDrado(RELLENO,x1+255,y1+96,x1+313,y1+106,BLANCOBRILLANTE);
        _setcolor(0);    /*posición del texto*/
        _moveto(x1+259,y1+94);
        _outgtext(fs);
        f_hi=atof(fs);
        CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
         sw3=TRUE;
        punteroon();
      }
   if((posix > x1+470) && (posix < x1+580))      /*orden del filtro*/
    if((posiy>y1+9) && (posiy<y1+21) && (dcha))
     {
      punteroff();
       CuaDrado(HUECO,x1+399,y1+3,x1+581,y1+28,NEGRO);
       CuaDrado(RELLENO,x1+540,y1+9,x1+573,y1+19,BLANCOBRILLANTE);
       CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
       PoneCadena(x1+7,y1+150,ROJO,"Indica el orden del filtro debe ser mayor de
CERO y menor de CIEN");
       fs=dato(544,7,3);
       CuaDrado(HUECO,x1+399,y1+3,x1+581,y1+28,MARRON);
       CuaDrado(RELLENO,x1+540,y1+9,x1+573,y1+19,BLANCOBRILLANTE);

      _setcolor(0);       /*letras negras */
      _moveto(x1+544,y1+7);
```

```
        _outgtext(fs);
        orden=atoi(fs);
        CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
        sw4=TRUE;
        punteroon();
      }
     if((posix > x1+450) && (posix < x1+470))      /*tipo recuadro*/
       if((posiy>y1+39) && (posiy<y1+51) && (dcha))
       {
        punteroff();
CuaDrado(HUECO,x1+399,y1+33,x1+471,y1+58,NEGRO);
        CuaDrado(RELLENO,x1+450,y1+39,x1+463,y1+51,BLANCOBRILLANTE);
        CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
        PoneCadena(x1+7,y1+150,ROJO,"Indica el tipo de filtro,0:Lowpass
                                    1:Highpass 2:Bandpass 3:Bandstop");

        fs=dato(454,37,1);

        CuaDrado(HUECO,x1+399,y1+33,x1+471,y1+58,MARRON);
        CuaDrado(RELLENO,x1+450,y1+39,x1+463,y1+51,BLANCOBRILLANTE);

        _setcolor(0);/*posición del texto*/
        _moveto(x1+454,y1+37);
        _outgtext(fs);
        tipo=atoi(fs);

        CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
        sw5=TRUE;
        punteroon();
      }
    if((posix > x1+510) && (posix < x1+551))      /*recuadro  atenuaci¢n*/
      if((posiy>y1+64) && (posiy<y1+81) && (dcha) )
```

```
        {
         punteroff();
        CuaDrado(HUECO,x1+399,y1+63,x1+556,y1+88,NEGRO);
        CuaDrado(RELLENO,x1+500,y1+69,x1+549,y1+81,BLANCOBRILLANTE);
        CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
PoneCadena(x1+7,y1+150,ROJO,"Indica la atenuación en dBs Atenuación > 0");
         fs=dato(504,67,4);
        CuaDrado(HUECO,x1+399,y1+63,x1+556,y1+88,MARRON);
        CuaDrado(RELLENO,x1+500,y1+69,x1+549,y1+81,BLANCOBRILLANTE);
        _setcolor(0);/*posición del texto*/
        _moveto(x1+504,y1+67);
        _outgtext(fs);
        atenuacion=atof(fs);

        CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
        sw7=TRUE;
        punteroon();
       }

     if((posix > x1+550) && (posix < x1+606))      /* recuadro de rizado*/
       if((posiy>y1+39) && (posiy<y1+51) && (dcha)  )
       {
        punteroff();
       CuaDrado(HUECO,x1+489,y1+33,x1+611,y1+58,NEGRO);
       CuaDrado(RELLENO,x1+555,y1+39,x1+603,y1+51,BLANCOBRILLANTE);

       CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
       PoneCadena(x1+7,y1+150,ROJO,"Indica el rizado en dBs de la banda de paso
0 < Rizado < Atenuación");

        fs=dato(558,37,4);
```

```
        CuaDrado(HUECO,x1+489,y1+33,x1+611,y1+58,MARRON);

        CuaDrado(RELLENO,x1+555,y1+39,x1+603,y1+51,BLANCOBRILLANTE);

        _setcolor(0);/*posición del texto*/
        _moveto(x1+558,y1+37);
        _outgtext(fs);
        rizado=atof(fs);
        CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
        sw6=TRUE;
        punteroon();
      }

    if((posix > x1+400) && (posix < x1+450) && (sw1) && (sw2) && (sw3) &&
(sw4) && (sw5) && (sw6) && (sw7))
      if((posiy>y1+96) && (posiy<y1+113) && (dcha))    /*botón*/
      {
       while(dcha)
           {
           leeraton(&posix,&posiy,&dcha,&izda);
           boton (x1+400,y1+96,x1+450,y1+113,"IR !",FALSE);
           }
           boton  (x1+400, y1+96, x1+450, y1+113, "IR !", TRUE);
           go=TRUE;
       }
    if((posix > x1+500) && (posix < x1+550) )
      if((posiy>y1+96) && (posiy<y1+113) && (dcha))    /*boton*/
      {
       while(dcha)
           {
           leeraton(&posix,&posiy,&dcha,&izda);
           boton (x1+500,y1+96,x1+550,y1+113,"EXIT",FALSE);
```

```
            }
            boton  (x1+500, y1+96, x1+550, y1+113, "EXIT", TRUE);
            exit=TRUE;
          }
      if(kbhit() && getch()==27) break;
      if(exit)
      {
        CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
        _setcolor(AZULCLARO); /*borramos lo anterior dibujando un rectangulo
AZUL*/
        punteroff();
        _rectangle(_GFILLINTERIOR,0,16,639,479);
        punteroon();
        break;
      }

    if(go)
     {
      CuaDrado(RELLENO,x1+6,y1+146,x1+631,y1+171,BLANCOBRILLANTE);
      orden1=orden;
      switch(tipo)
       {
        case 0: tama_array=orden+1; break;
        case 1: tama_array=orden+1; break;
        case 2: tama_array=(2*orden+1); break;
        case 3: tama_array=(2*orden+1); break;
       }
      if(TipoFil==0)
       {
       coeficientes((float)fm,(float)f_low,(float)f_hi,tipo,tama_array,orden1,0,0,0);
       break;
       }
```

```
      if(TipoFil==1 || (TipoFil==2))
        {
coeficientes((float)fm,(float)f_low,(float)f_hi,tipo,tama_array,orden1,0,(float)rizado,1
);
        break;
        }
      if(TipoFil==2)
        {
coeficientes((float)fm,(float)f_low,(float)f_hi,tipo,tama_array,orden1,(float)atenuacio
n,(float)rizado,2);
        break;
        }
  }/*del if go*/
   }
  while(1);
}

char *dato(int px,int py,int largo)
{
 char car;
 static char frase[5];
 int i,c,j;
 int x1=0,y1=301,x2=639,y2=439;
 i=0;
 c=0;
 frase[0]='\0';
do
{
 car=getch();
 switch(car)
 {
  case  0: car=getch();break;
```

```
case 13: if (i==0)
           break;
         else
           frase[i]='\0';
         break;
case 8: if(i>0)   /*borrar*/
         {
         i--;
         c-=10;
         frase[i]=frase[i];
         frase[i+1]='\0';
         _moveto(x1+px+c,y1+py);
         _setcolor(15);              /*invisible*/
         _outgtext(frase+i);
         _moveto(x1+px+c,y1+(py-1));
         _setcolor(0);              /*visible*/
         _outgtext("");
         _setcolor(15);              /*invisible*/
         _outgtext("");
         }
         break;

 case 47:break;

default:  /*if(car=47) break;
         else*/
         if( (  (car>=46) && (car<=57) )   && (i<largo)  )
         {
          /*if(car=47) break;*/

          frase[i]=car;
          frase[i+1]='\0';
```

```
            _setcolor(0);                /*blanco*/
            _moveto(x1+px+c,y1+py);
            _outgtext(frase+i);
            _moveto(x1+px+c,y1+(py-1));
            _setcolor(15);
            _outgtext("");
            _setcolor(0);
            _outgtext("");
            i++;
            c+=10;
            }
           break;
      }
   }
 while (car!=13);
 return(frase);
}

void titulos(int xs1, int ys1, int xs2,int ys2,int fondo)
{
 ys1=ys1-1;
 ys2=ys2+2;
 _setcolor(7);
 _rectangle(3,xs1-4,ys1-4,xs2+4,ys2+4);
 _setcolor(fondo);
 _rectangle(3,xs1,ys1,xs2,ys2);
 _setcolor(0);
 _moveto(xs1,ys2);
 _lineto(xs1,ys1);
 _lineto(xs2,ys1);
 _setcolor(15);
 _moveto(xs1,ys2);
```

```
_lineto(xs2,ys2);
_lineto(xs2,ys1);
_setcolor(7);
_moveto(xs1-1,ys2-1);
_lineto(xs2-1,ys2-1);
_lineto(xs2-1,ys1-1);
}

void DibujaFlecha(int cordex1,int cordex2,int cordey,int arriba)
{
      int i,j,conta=0;
      _setcolor(0);
      if(arriba==1)
      {
       for(i=cordey;i<cordey+5;i++)
        {
          if(conta>0)
           {
            cordex1--;
            cordex2++;
           }
      for(j=cordex1;j<(cordex2+1);j++)
       {
         _setpixel(j,i);
         conta++;
       }
     }
 }

if(arriba==0)
{
  for(i=cordey;i>cordey-5;i--)
```

```
 {
  if(conta>0)
  {
   cordex1--;
   cordex2++;
   }
    for(j=cordex1;j<(cordex2+1);j++)
    {
     _setpixel(j,i);
     conta++;
    }
}
}

}
void PoneCadena(int xp1,int yp1, int color,char *texto)
{
        _moveto(xp1,yp1);
         _setcolor(color);
        _outgtext(texto);
}

void CuaDrado(int forma, int cx1, int cy1, int cx2, int cy2, int Color)
{
 _setcolor(Color);
 _rectangle(forma,cx1,cy1,cx2,cy2);
}
```

```
/* CALCULA Y DIBUJA LA REPUESTA DEL FILTRO */

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned char BOOLEAN;

#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>
#include <graph.h>
#include <malloc.h>
#include <dos.h>
#include <math.h>
#include "c:\atdsp\dsp2200.h"
#include "c:\nidsp\include\atdsp.h"
#include "c:\nidsp\include\dsp.h"
#include "colores.h"
#include <direct.h>
#include <string.h>

#define HUECO 2
#define RELLENO 3

#define SLOT 1
#define PI 3.141592645
#define TRUE 1
#define FALSE 0

void titulos(int,int,int,int,int);
int freqz(float *,float *,int,float,int);
void plot(float *,int,int, float,int);
```

```
char *escribe(int,int,int,int,int,char *,char *);
void filtro(char *,float *,float *,int,char *,int);
void PoneCoeficientes(float *,float *,int);
void PoneCadena(int,int,int,char *);
void CuaDrado(int,int,int,int,int,int);
int ExisteFile(char *NombreFile);
void boton(int,int,int,int,char *,BOOLEAN);
void ActivaMenu(int);

int coeficientes(float fs,float lo,float hi,int tipo,int ta_array,int orden,float
atenuacion,float rizado,int tipofiltro)
{
  static float *a=NULL;
  static float *b=NULL;
  char buffer[15];
  int
        i,j,x1=0,x2=0,retorno=0;
/* puteros a los coeficentes */
  a=(float *) malloc(ta_array*sizeof(float));
  b=(float *) malloc(ta_array*sizeof(float));
  desactivaMenu(2);  /* desactiva el menú de edición */
  punteroff();
/* pantalla donde se ven los coeficientes */
  CuaDrado(RELLENO,380,392,555,419,MARRON);
  titulos(4,23,635,294,AMARILLOCLARO);
  punteroon();
  _setcolor(NEGRO);
  _moveto(50,25);
  _outgtext("COEF A:");
  _moveto(180,25);
  _outgtext("COEF B:");
  switch(tipofiltro) /* se calculan los coeficintes en función del filtro seleccioando */
```

```
{
case 0:  DSP_Bw_Coef(SLOT,fs,lo,hi,orden,a,b,tipo); /* calcula los coeficientes
                                                    de un filtro Butterworth*/
         PoneCoeficientes(a,b,ta_array); /* llama a la rutina que dibuja los
                                            coeficientes */
         PoneCadena(7,451,ROJO,"PULSE UNA TECLA PARA VER LA
                                            REPUESTA DEL FILTRO");
         getch();
         punteroff();
         CuaDrado(RELLENO,6,447,631,472,BLANCOBRILLANTE);
         CuaDrado(RELLENO,0,16,639,300,AZULCLARO);
         punteroon();
           /* llama a la rutina encargada de dibujar la respuesta en frecuencia */
           /* del filtro. Si no se filtra un fichero cuando pulsamos salir retorna */
         if(freqz(a,b,ta_array,fs,tipo)==1)
         {
            retorno=1;
         }
         break;

case 1: DSP_Ch_Coef(SLOT,fs,lo,hi,orden,rizado,a,b,tipo); /* calcula los
                                            coeficientes de un filtro chebyshev */

         PoneCoeficientes(a,b,ta_array); /* pone los coeficientes */
         PoneCadena(7,451,ROJO,"PULSE UNA TECLA PARA VER LA
                                            REPUESTA DEL FILTRO");
         getch(); /* espera a que se pulse una tecla antes de dibujar la respuesta en
                                                    frecuencia */
         punteroff();
         CuaDrado(RELLENO,6,447,631,472,BLANCOBRILLANTE);
         CuaDrado(RELLENO,0,16,639,300,AZULCLARO);
         punteroon();
```

```
        /* llama a la rutina encargada de dibujar la respuesta en frecuencia */
        /* del filtro. Si no se filtra un fichero cuando pulsamos salir retorna */

      if(freqz(a,b,ta_array,fs,tipo)==1)
      {
        retorno=1;
      }
      break;


case 2: DSP_Elp_Coef(SLOT,fs,lo,hi,orden,rizado,atenuacion,a,b,tipo); /* calcula
                                        los coeficientes de un filtro elítico */
      PoneCoeficientes(a,b,ta_array);  /* pone los coeficientes */
      PoneCadena(7,451,ROJO,"PULSE UNA TECLA PARA VER LA
                                          REPUESTA DEL FILTRO");
      getch();/* espera a que se pulse una tecla antes de dibujar la respuesta en
                                                  frecuencia */

      punteroff();
      CuaDrado(RELLENO,6,447,631,472,BLANCOBRILLANTE);
      CuaDrado(RELLENO,0,16,639,300,AZULCLARO);
      punteroon();

        /* llama a la rutina encargada de dibujar la respuesta en frecuencia */
        /* del filtro. Si no se filtra un fichero cuando pulsamos salir retorna */
      if(freqz(a,b,ta_array,fs,tipo)==0)
      {
       retorno=1;
      }
      break;
}
```

```
free(a);
free(b);
if(retorno=1)
    {
     return(1);
     }
}
/* función que calcula la respuesta en frecuencia del filtro */
int freqz(float *a, float *b,int tamaA,float frm,int tipo)
{
  static float *x=NULL;
  static float *y=NULL;
  static float *z=NULL;
  static float *k=NULL;
  int
        i,j,min,max,resultado,filtrar=0;
  WORD posix,posiy;
  BOOLEAN dcha,izda;
  char *sx,car,*f_ori,*f_dest,origen[15];
  float fx,ffx;

  punteroff();
  boton(400,397,490,414,"FILTRAR",TRUE);
  boton(550,397,600,414,"EXIT",TRUE);
  titulos(4,23,635,296,AMARILLOCLARO);
  punteroon();
  x=(float *) malloc(256*sizeof(float)); /* reserva memoria para un array */
  if(x==NULL)
     printf("Error al asignar memoria para x");
  y=(float *) malloc(256*sizeof(float));
  if(x==NULL)
     printf("Error al asignar memoria para y");
```

```
  z=(float *) malloc(256*sizeof(float));
  if(x==NULL)
     printf("Error al asignar memoria para z");
  k=(float *) malloc(256*sizeof(float));
  if(x==NULL)
     printf("Error al asignar memoria para k");
for(i=0;i<tamaA;i++)
  {
  x[i]=a[i];
  z[i]=b[i];
 }
  DSP_ZeroPad(SLOT,x,tamaA,256); /* rellena de ceros el 256 - los coeficientesA */
  DSP_ReFFT(SLOT,x,y,256); /* calcula la transformada rápida de fourrier */
   DSP_ZeroPad(SLOT,z,tamaA,256); /* lo mismo para los coeficientes B*/
   DSP_ReFFT(SLOT,z,k,256);
   DSP_CxDiv(SLOT,x,y,z,k,128,z,k); /* divide la FFT A por la FFT B */
   DSP_ToPolar(SLOT,z,k,128,z,k); /* Calcula el módulo y el argumento */
   max=min=(k[i]*180/PI);

   for(i=0;i<128;i++)
    {
     if((int)(k[i]*180/PI) < min) min=(int)(k[i]*180/PI); /* calcula el máximo y el
                                                                    mínimo*/
     if((int)(k[i]*180/PI) > max) max=(int)(k[i]*180/PI);
    }

  if(min>=0) min-=(min % 10);
  else min-= (10+(min % 10));

  if(max>=0) max+= (10-(max % 10));
  else max-=(max % 10);
```

```
 plot(z,250,250,frm,0); /* dibuja la magnitud*/
 plot(k,550,250,frm,1); /* dibuja la fase*/
_moveto(100,22);
_outgtext("MAGNITUD");
_moveto(420,22);
_outgtext("FASE");
_setcolor(0);

do /* espera a que se pulse una tecla */
{
 leeraton(&posix,&posiy,&dcha,&izda);
 if((posix > 550) && (posix < 600))
  if((posiy > 397) && (posiy <414) && (dcha))
 {
 while(dcha)
      {
         leeraton(&posix,&posiy,&dcha,&izda);
         boton(550,397,600,414,"EXIT",FALSE);
      }
      boton(550,397,600,414,"EXIT",TRUE);
      free(x);
      free(y);
      free(k);
      free(z);
      punteroff();
      _setcolor(AZULCLARO);
      _rectangle(_GFILLINTERIOR,0,16,639,479);
      punteroon();
      ActivaMenu(2);
      return(1);
}
```

```
if((posix > 400) && (posix < 490))
 if((posiy > 397) && (posiy <414) && (dcha) )
   {
  while(dcha)
  {
   leeraton(&posix,&posiy,&dcha,&izda);
   boton(400,397,490,414,"FILTRAR",FALSE);
  }
   boton(400,397,490,414,"FILTRAR",TRUE);
   filtrar=1;
   break;
 }
}
while(1);

do
{
 punteroff();
 f_ori=escribe(130,110,350,185,12,"Fichero Origen","");
 punteroon();
   if(*f_ori=='\0')
   {
    free(x);
    free(y);
    free(k);
    free(z);
    return(1);
   }
 /*comprobar si existe el fichero y si no dar un mensaje de error*/
 do
 {
```

```
      resultado=ExisteFile(f_ori);
      if(resultado==0)
      {
       CuaDrado(RELLENO,6,447,631,472,BLANCOBRILLANTE);
       PoneCadena(7,451,ROJO,"ESCRIBA OTRO NOBRE DE FICHERO");
       punteroff();
       f_ori=escribe(130,110,350,185,12,"Fichero Origen","");
       punteroon();
       CuaDrado(RELLENO,6,447,631,472,BLANCOBRILLANTE);
       PoneCadena(7,451,ROJO,"EL FICHERO ORIGEN NO EXISTE");

      if(*f_ori=='\0')
      {
       free(x);
       free(y);
       free(k);
       free(z);
       return(1);
      }
     }
 }
while(resultado==0);

   CuaDrado(RELLENO,6,447,631,472,BLANCOBRILLANTE);
   strcpy(origen,f_ori);
   punteroff();
   f_dest=escribe(130,110,350,185,12,"Fichero Destino","");
   punteroon();
     if(*f_dest=='\0')
     {
      free(x);
      free(y);
```

```
        free(k);
        free(z);
        return(1);
       }

    do
    {
      resultado=ExisteFile(f_dest);
      if(resultado==1)
      {
       CuaDrado(RELLENO,6,447,631,472,BLANCOBRILLANTE);
       PoneCadena(7,451,ROJO,"EL FICHERO DESTINO YA EXISTE. SOBRE
ESCRIBIR: S/N");
       car=getch();
       if (_toupper(car)=='S')
       {
        resultado=1;
        break;
       }
       CuaDrado(RELLENO,6,447,631,472,BLANCOBRILLANTE);
       PoneCadena(7,451,ROJO,"ESCRIBA OTRO NOBRE DE FICHERO");
       punteroff();
       f_dest=escribe(130,110,350,185,12,"Fichero Destino","");
       punteroon();
       if(*f_dest=='\0')
       {
        free(x);
        free(y);
        free(k);
        free(z);
        return(1);
       }
```

```
    }
}
while(resultado==1);
CuaDrado(RELLENO,6,447,631,472,BLANCOBRILLANTE);
 filtro(origen,a,b,tamaA,f_dest,tipo); /*Esta es la llamada a la función que filtra el
                                                                    fichero */
    break;
}
while(1);
  free(x);
  free(y);
  free(k);
  free(z);
  getch();
  punteroff();
  _setcolor(9);
  _rectangle(_GFILLINTERIOR,0,16,639,479);
  punteroon();
  desactivaMenu(1);
  ActivaMenu(2);
}
void plot(float *p,int lx,int ly,float fr,int sw)
{
  int i,pts_to_plot,max,min,step_size,j;
  double x_scale,y_scale;
  unsigned int xx,yy,xcount;
  char *sx;
  float fx,ffx;
  _setcolor(0);
   punteroff();
  _moveto(lx-200,ly-5); /*linea horizontal*/
  _lineto(lx,ly-5);
```

```
 _moveto(lx+10,ly-5);
 _outgtext("Hz");
  fx=((fr/2)/128); /*pasos para la frecuencia del eje x*/
 for(i=0,j=0;i<220;i+=25,j++)
 {
       _moveto((lx-200)+i,ly-5); /*marcas en el eje x*/
       _lineto((lx-200)+i,ly-5+5);
       _moveto((lx-207)+i,ly-5+50);
       _setgtextvector(0,1);
        ffx=fx*16*j;
        itoa((int)ffx,sx,10);
       _outgtext(sx);
 }
 _setgtextvector(1,0);
 _moveto(lx-200,ly-5-200); /*linea vertical*/
 _lineto(lx-200,ly-5);
 if(sw==0)
 {
   for(i=0,j=0;i<220;i+=20,j++)
    {
      _moveto(lx-200,ly-5-i); /*marcas en el eje y*/
      _lineto(lx-205,ly-5-i);
      _moveto(lx-235,(ly-5-7)-i);
      ffx=(float) j/10;
      gcvt(ffx,3,sx);
     _outgtext(sx); /*numeros en el eje y*/
    }
}
if(sw==1)
{
 for(i=0;i<128;i++)
 {
```

```
  if((int)(p[i]*180/PI) < min) min=(int)(p[i]*180/PI);
  if((int)(p[i]*180/PI) > max) max=(int)(p[i]*180/PI);
  }
  if(min>=0) min-=(min % 10);
  else min-= (10+(min % 10));
  if(max>=0) max+= (10-(max % 10));
  else max-=(max % 10);
   j=max;
   for(i=0;i<225;i+=25)
    {
      _moveto(lx-200,ly-5-i); /*marcas en el eje y*/
      _lineto(lx-205,ly-5-i);
      _moveto(lx-238,((ly-5-7)-200)+i);

      gcvt((float)j,3,sx);
       _outgtext(sx); /*numeros en el eje y*/
       j=(j-(max/4));
    }
}
  pts_to_plot=128;
  min=max=p[0];
  for(i=0;i<pts_to_plot;i++)
  {
      if(p[i] < min) min=p[i];
      if(p[i] > max) max=p[i];
  }
  if((min==0) && (max==0) )
      {
          min=0; /*evita el error de divisi¢n por cero*/
          max=1;
      }
```

```
  x_scale=(double)(200)/(double)(128); /* cáculo de las escalas X e Y */
  y_scale=(double)(200)/(double)(max-min);
  _setlogorg(lx-200,50-5);
  step_size=pts_to_plot/100;
  if(!step_size) step_size=1;
  step_size=1;
  xx=xcount=0;
  yy=(int)((p[0]-min)*y_scale);
  yy=(200-yy);
  _setcolor(4);
  _moveto(xx,yy);
  for(i=1;i<pts_to_plot;i++)
  {
      xx=(int) ((double)(++xcount)*x_scale);
      yy=(int) ((p[i]-min)*y_scale);
      yy=200-yy;
      xx=abs(xx);
      yy=abs(yy);
      if(yy<15) yy=16;
       _lineto(xx,yy);
   }
  _setlogorg(0,0);
  punteroon();
}

void PoneCoeficientes(float * ca,float * cb,int TamaArray)
{

   char buffer[15], *ind1;
    int i,j,x1=0,x2=0,ContaPases=0;
   punteroff();
    for(i=0,j=0;i<TamaArray;i++,j+=14)
```

```
{
  _setcolor(NEGRO);
  _moveto(25+x1,50+j);
  _outgtext("a[    ]=");
  itoa(i,ind1,10);
  _moveto(39+x1,50+j);
  _outgtext(ind1);
  gcvt(ca[i],5,buffer);
  _moveto(75+x1,50+j);
  _outgtext(buffer);
  _moveto(175+x2,50+j);
  _outgtext("b[    ]=");
  _moveto(189+x1,50+j);
  _outgtext(ind1);
  gcvt(cb[i],5,buffer);
  _moveto(225+x2,50+j);
  _outgtext(buffer);
  if( (i!=0) && (i % 15)==0)
    {
     ContaPases++;
     if(ContaPases<2)
     {
      x1=300;
      x2=300;
      j=0;
     }
     else
      {
      if(i<TamaArray)
      {
        PoneCadena(7,451,ROJO,"PULSE UNA TECLA PARA VER EL RESTO
                                                DE LOS COEFICIENTES");
```

```
            getch();
           }
          CuaDrado(RELLENO,10,50,630,280,AMARILLOCLARO);
          CuaDrado(RELLENO,6,447,631,472,BLANCOBRILLANTE);
          x1=0;
          x2=0;
          j=0;
          ContaPases=0;
         }
       }/*del primer else*/
    }
    punteroon();
}

int ExisteFile(char *NombreFile)
{
 char NombreRuta[_MAX_PATH];
 char entorno[_MAX_PATH];
 int existe;

 if(getcwd(entorno,_MAX_PATH)==NULL)
 {
  printf("Falla en la funcion getcwd");
  exit(1);
 }
 _searchenv(NombreFile,entorno,NombreRuta);
 if(*NombreRuta!='\0')
    return(1);
 else
    return(0);
}
```

```
/* FUNCIÓN QUE PERMITE FILTRAR UN FICHERO*/

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <io.h>
#include <graph.h>
#include "colores.h"
#define TAMA 1024
#define RELLENO 3
#define HUECO 2

void DspAirr(char * ,char *, float *, float *, int);
void EstereoMono(char *);
void MonoEstereo(char *);
void boto1(int, int, int,int, int, char *);
void PoneCadena(int,int,int,char *);
void CuaDrado(int,int,int,int,int,int);

void filtro(char * FileOrigen, float *Array_A, float *Array_B,
        int TamanoArray,char *FileDestino, int TipoFiltro)

{
 FILE *pf,*pf1,*pf2;
 int NumeroCanales,i;
  static char huge *Buffer=NULL;
 long TamaImagen,j;
 /*1° vemos si es mono o estero*/

 if((pf=fopen(FileOrigen,"r+b"))==NULL)
 {
  printf("Error al abrir el fichero %s",FileOrigen);
```

```
 exit(1);
}
fread(&NumeroCanales,sizeof(int),1,pf);
fclose(pf);
TamaImagen=_imagesize(250,100,440,150);
  Buffer=(char huge *) halloc(TamaImagen,sizeof(char)); /* asigna el buffer para los
                                                                    datos*/
     if(Buffer==NULL)
     {
          printf("Error 3: No hay memeria suficiente para asignar a Buffer");
          exit(1);
     }
      punteroff();
     _getimage(250,100,440,150,Buffer);
      boto1(250,100,440,150,5,"ESPERE POR FAVOR");
      if(NumeroCanales==2) /* es un fichero estereo primero se separa */
         {
          EstereoMono(FileOrigen);
          if( (pf1=fopen("mono.fi0","w+b"))==NULL) /* se crean los ficheros para
                                                          separar los canales */
            {
              printf("Error al abrir el fichero mono.fi0");
               exit(1);
             }
           if( (pf2=fopen("mono.fi1","w+b"))==NULL)
              {
               printf("Error al abrir el fichero mono.fi1");
               exit(1);
               }
            fclose(pf1);
            fclose(pf2);
```

```
  /* llamada a la fucnión que implementa el algoritmo de filtrado de un filtro IIR */
  DspAirr("mono.ch0" ,"mono.fi0",Array_B, Array_A,TamanoArray);
  DspAirr("mono.ch1" ,"mono.fi1",Array_B, Array_A,TamanoArray);
  MonoEstereo(FileDestino); /* convierte el fichero en estereo otravez */
 }

 else /* si es un fichero mono se filtra directamente */

     DspAirr(FileOrigen ,FileDestino,Array_B, Array_A,TamanoArray);

 /* OJO QUE AQUI EL ARAY A Y EL B ESTA CAMBIADOS PARA
HACERLOS CONCIDIR*/
 /*CON PROCEDIMIENTO DEL FILTRO QUE YA TENIA HECHO*/
 /*filtrar*/
 _putimage(250,100,Buffer,_GPSET);
 hfree(Buffer);
 punteroon();
 remove("mono.ch0");
 remove("mono.ch1");
 remove("mono.fi0");
 remove("mono.fi1");
 for(j=0;j<80000;j++)
 {
 /*RETARDO*/
 }
 CuaDrado(RELLENO,6,447,631,472,BLANCOBRILLANTE);
 PoneCadena(7,451,ROJO,"EL FILTRADO HA TERMINADO, PULSE UNA
                                                    TECLA");
}

/* rutina que separa las muestras de un fichero estereo en dos ficheros que contienen
cada uno un canal */
```

```
void EstereoMono(char *FicheroOrigen)
{
 FILE *pf0,*pf1,*pf2;
 int *x1=NULL;
 int *y1=NULL;
 int *y2=NULL;
 int i,j,n,k,numero;
 char cr;
/*asignar buffer*/
      x1=(int *) malloc (TAMA*sizeof(int));
      if(x1==NULL)
      {
            printf("Error 1: No hay memeria suficiente");
            exit(1);
      }
      y1=(int *) malloc (TAMA*sizeof(int));
      if(y1==NULL)
      {
            printf("Error 2: No hay memeria suficiente");
            exit(1);
      }
      y2=(int *) malloc (TAMA*sizeof(int));
      if(y2==NULL)
      {
            printf("Error 3: No hay memeria suficiente");
            exit(1);
      }
      if( (pf0=fopen(FicheroOrigen,"r+b"))==NULL)
      {
       printf("Error al abrir el fichero %s",FicheroOrigen);
       exit(1);
      }
```

```
        if( (pf1=fopen("mono.ch0","w+b"))==NULL)
        {
         printf("Error al abrir el fichero mono.ch0");
         exit(1);
        }
        if( (pf2=fopen("mono.ch1","w+b"))==NULL)
        {
         printf("Error al abrir el fichero mono.ch1");
         exit(1);
        }
        boto1(250,100,440,150,5,"ESPERE POR FAVOR");
        while(!feof(pf0))
        {
          fread(x1,sizeof(int),TAMA,pf0);
          for(i=0,k=1,j=0;i<TAMA;i+=2,k+=2,j++)
          {
             y1[j]=x1[i];
             y2[j]=x1[k];
          }
         fwrite(y1,sizeof(int),TAMA/2,pf1);
         fwrite(y2,sizeof(int),TAMA/2,pf2);
       }
/* libera la memoria y cierra los ficheros */
  free(x1);
  free(y1);
  free(y2);
  fclose(pf0);
  fclose(pf1);
  fclose(pf2);
}
/* rutina de filtrado IIR */
```

```
void DspAirr(char *FICHEROORIGEN ,char *FICHERODESTINO, float *a, float
*b, int ORDEN)
{
 FILE *pf0,*pf1;

 int *x1=NULL;
 float *x=NULL;
 float *y=NULL;
 int *y1=NULL;
 float *cfa=NULL;
 float *cfb=NULL;
 float *cf=NULL;

 float ci,fs;
 int i,j,n,s,k,tipo,contador=0;
 char cr;
 struct CABECERA
 {
  int nu_cana;
  unsigned int fr;
  }cabeza;

/* asigna los buffer en memoria dinámica */
  x=(float *) malloc(TAMA*sizeof(float));
     if(x==NULL)
     {
          printf("Error 3: No hay memeria suficiente");
          exit(1);
     }

  y=(float *) malloc(TAMA*sizeof(float));
     if(y==NULL)
```

```
    {
        printf("Error 3: No hay memeria suficiente");
        exit(1);
    }

x1=(int *) malloc(TAMA*sizeof(int));
    if(x1==NULL)
    {
        printf("Error 3: No hay memeria suficiente");
        exit(1);
    }
y1=(int *) malloc(TAMA*sizeof(int));
    if(y1==NULL)
    {
        printf("Error 3: No hay memeria suficiente");
        exit(1);
    }
    if(y1==NULL)
    {
        printf("Error 3: No hay memeria suficiente");
        exit(1);
    }
cfa=(float *) malloc(TAMA*sizeof(float));
    if(cfa==NULL)
    {
        printf("Error 3: No hay memeria suficiente");
        exit(1);
    }

cfb=(float *) malloc(TAMA*sizeof(float));
    if(cfb==NULL)
    {
```

```
            printf("Error 3: No hay memeria suficiente");
            exit(1);
        }

    cf=(float *) malloc(TAMA*sizeof(float));
        if(cf==NULL)
        {
            printf("Error 3: No hay memeria suficiente");
            exit(1);
        }

 if((pf0=fopen(FICHEROORIGEN,"r+b"))==NULL)
 {
  printf("Error al abrir el fichero %s\n",FICHEROORIGEN);
  exit(1);
 }
 if((pf1=fopen(FICHERODESTINO,"w+b"))==NULL)
 {
  printf("Error al abrir el fichero %s\n",FICHERODESTINO);
  exit(1);
 }
 for(i=0;i<ORDEN;i++)
 {
   cfa[i]=0.0; /* inicializa las condiciones finales */
   cfb[i]=0.0;
   cf[i]=0.0;
 }
while(!feof(pf0))
{
 fread(x1,sizeof(int),TAMA,pf0);
 for(i=0;i<TAMA;i++)
  {
```

```
 if((contador==0) && (i<1))
   {
    cabeza.nu_cana=x1[0];
    cabeza.fr=(unsigned int)x1[1];
   }
 x[i]=(float)(x1[i]*2.828)/32768;  /* realiza la conversión entre enteros y reales */
 }/* del for*/
/* ---------------------------------rutina de filtrado-----------------------------------*/
 for(n=0;n<(ORDEN-1);n++)
 {
  y[n]=0;
      for(j=0;j<n;j++)
       y[n]=y[n]-(a[j+1]*y[n-j-1]);
      for(i=0;i<(n+1);i++)
       y[n]=y[n]+(b[i]*x[n+1-i-1]);

  y[n]=y[n]+cf[n]; /* suma las condiones finales */
 }

 for(n=(ORDEN-1);n<TAMA;n++)
 {
  y[n]=0;
     for(j=0;j<(ORDEN-1);j++)
       y[n]=y[n]-(a[j+1]*y[n-j-1]);
     for(i=0;i<ORDEN;i++)
       y[n]=y[n]+(b[i]*x[n+1-i-1]);
  if(y[n]>20)
    {
     PoneCadena(7,451,ROJO,"ERROR DE DESBORDAMIENTO, DISMINUYA
       EL ORDEN DEL FILTRO");

     /* libera los buffers */
```

```
      fclose(pf0);
      fclose(pf1);
      free(x);
      free(y);
      free(x1);
      free(y1);
      free(cfa);
      free(cfb);
      free(cf);
      return;
    }
 }  /*del for*/
/* Cálculo de las condiones finales */
 for(k=0;k<(ORDEN-1);k++)
 {
  cfa[k]=0;
     for(j=k,s=1;j<(ORDEN-1);j++,s++)
       cfa[k]=cfa[k]+(b[j+1]*x[TAMA-s]);

       cfb[k]=0;

     for(i=k,s=1;i<(ORDEN-1);i++,s++)
       cfb[k]=cfb[k]-(a[i+1]*y[TAMA-s]);

       cf[k]=cfa[k]+cfb[k];
 }

 for(i=0;i<TAMA;i++)
  y1[i]=(y[i]*32768)/2.828; /* realiza la conversión de reales a enteros y guarda los
                                                    datos filtrados en el fichero destino */

 if(contador==0)
```

```
  {
   y1[0]=cabeza.nu_cana;
   y1[1]=cabeza.fr;
  }

 fwrite(y1,sizeof(int),TAMA,pf1);

 contador++;
} /* fin del while*/
/* libera los buffers */
 fclose(pf0);
 fclose(pf1);
 free(x);
 free(y);
 free(x1);
 free(y1);
 free(cfa);
 free(cfb);
 free(cf);
}
/* junta dos las muestras de dos ficheros en uno solo */
void MonoEstereo(char *FileDESTINO)
{
 FILE *pf0,*pf1,*pf2;

  int *x1=NULL;
  int *y1=NULL;
  int *y2=NULL;
 int i,j,n,k;
 char cr;

     x1=(int *) malloc ((TAMA*2)*sizeof(int));
```

```
    if(x1==NULL)
    {
        printf("Error 1: No hay memeria suficiente");
        exit(1);
    }
    y1=(int *) malloc (TAMA*sizeof(int));
    if(y1==NULL)
    {
        printf("Error 1: No hay memeria suficiente");
        exit(1);
    }
    y2=(int *) malloc (TAMA*sizeof(int));
    if(y2==NULL)
    {
        printf("Error 1: No hay memeria suficiente");
        exit(1);
    }
if((pf0=fopen(FileDESTINO,"w+b"))==NULL)
  {
   printf("ERROR AL ABRIR EL FICHERO");
   exit(1);
  }
if((pf1=fopen("mono.fi0","r+b"))==NULL)
 {
  printf("ERROR AL ABRIR EL FICHERO");
  exit(1);
 }
if((pf2=fopen("mono.fi1","r+b"))==NULL)
{
 printf("ERROR AL ABRIR EL FICHERO");
 exit(1);
}
```

```
i=0;
while( (!feof(pf1)) && (!feof(pf2)) )
{
 fread(y1,sizeof(int),TAMA,pf1);
 fread(y2,sizeof(int),TAMA,pf2);
 for (j=0;j<TAMA;j++)
  {
   x1[i]=y1[j];
   i++;
   x1[i]=y2[j];
   i++;
   }
  i=0;
  fwrite(x1,sizeof(int),TAMA*2,pf0);
}
fclose(pf0);
fclose(pf1);
fclose(pf2);
}
```

```
/*PROGRAMA QUE PERMITE ESCRIBIR EN MODO GRAFICO*/

typedef unsigned char BOOLEAN;
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <graph.h>
#include "colores.h"
#define MAX 15
char *escribe(int x1,int y1,int x2,int y2,int largo,char *texto1,char*texto2)
{
  char  huge *panta,car;
  long tama_panta;
  static char frase[MAX];
  int i,c,escape=0;
  i=0;
  c=0;

  _setfont("t'courier'h13w10bf");

  tama_panta=_imagesize(x1,y1,x2,y2);
  panta=(char huge *) halloc(tama_panta,sizeof(char));
  _getimage(x1,y1,x2,y2,panta);



  _setcolor(MARRON);
  _rectangle(3,x1,y1,x2,y2);

  _setcolor(BLANCOBRILLANTE);
  _rectangle(2,x1+5,y1+5,x2-5,y2-5);
  _moveto(x1+10,y1+10);
  _outgtext(texto1);
```

```
  _moveto(x1+10,y1+50);
  _outgtext(texto2);
  _setcolor(MARRON);
  _rectangle(3,x1+10,y1+50,x1+152,y1+64);
do
   {
    car=getch();
    switch(car)
    {
     case 27:
            frase[0]='\0';
            break;
     case 0:car=getch(); /*esto evita qu se impriman los caracteres de comtrol*/
              break;
     case 13: if (i==0)
              break;
            else
              frase[i]='\0';
            break;
     case  8: if (i>0)
            {
              i--;
              c-=13;
              frase[i]=frase[i];
              frase[i+1]='\0';
              _moveto(x1+13+c,y1+50);
              _setcolor(MARRON);
              _outgtext(frase+i);
              _moveto(x1+13+c,y1+52);
            }
             break;
```

```
        default: if (i<largo)
                 {
                  frase[i]=car;
                  frase[i+1]='\0';
                  _setcolor(NEGRO);
                  _moveto(x1+13+c,y1+50);
                  _outgtext(frase+i);
                  _moveto(x1+13+c,y1+52);
                  i++;
                  c+=13;
                 }
                  break;
           }
      }
       while((car!=13) && (car!=27));
      _putimage(x1,y1,panta,_GPSET);
      hfree(panta);
      _setfont("t'Tms Rmn',h13w13b");
       return(frase);
}
```

```
/*PROCEDIMIENTO DE MARCAR EL TROZO SELECCIONADO*/

typedef unsigned char BOOLEAN;
#include <stdio.h>
#include <conio.h>
#include <graph.h>
#include <malloc.h>
#include "colores.h"
#define FALSE 0
#define TRUE 1
void marcar(int *x1,int *x2,char *Nfile)
{
 int     psx,
         psy,
     marca=0,c,sw1=0,sw2=0;

 BOOLEAN izda,
         dcha;

 long    tama,i;
 int columna,fila1,Xini,Xfin,Dato,fila2;

 FILE *pf;
struct
{
 int Canales;
 unsigned int Frecu;
 }cabecera;

  *x1=0;
  *x2=0;
```

```
if((pf=fopen(Nfile,"r+b"))==NULL)
{
 printf("ERROR AL ABRIR EL FICHERO %s",Nfile);
 exit(1);
}
fread(&cabecera,sizeof(int),1,pf);
fclose(pf);

do
 {
 leeraton(&psx,&psy,&izda,&dcha);
 if((psy>62) && (psy<179))
  if((psx>1) && (psx<638)  && (izda))
  {
   *x1=Xini=psx;
   sw1=1;
   }
 if((psy>62) && (psy<179))
  if((psx>1) && (psx<638) && (dcha))
  {
    *x2=Xfin=psx;
    sw2=1;
   }
 if((psy>207) && (psy<334))
  if((psx>1) && (psx<638)  && (izda) && (cabecera.Canales==2))
  {
   *x1=Xini=psx;
   sw1=1;
   }
 if((psy>207) && (psy<334))
  if((psx>1) && (psx<638) && (dcha) && (cabecera.Canales==2))
  {
```

```
    *x2=Xfin=psx;
    sw2=1;
   }
   if((sw1==1) && (sw2==1))
   break;
 }
while(1);
punteroff();
  for(fila1=76;fila1<169;fila1++)
       for(columna=Xini;columna<Xfin;columna++)
       {
        if((_getpixel(columna,fila1)==BLANCO) ||
(_getpixel(columna,fila1)==NEGRO))
        {
          _setcolor(CIAN);
          _setpixel(columna,fila1);
         }
        else
         if (_getpixel(columna,fila1)==ROJO)
          {
           _setcolor(ROJO); /*ROJO*/
           _setpixel(columna,fila1);
          }
       }
 if(cabecera.Canales==2)
 {
  for(fila2=220;fila2<325;fila2++)
       for(columna=Xini;columna<Xfin;columna++)
       {
        if((_getpixel(columna,fila2)==BLANCO) ||
(_getpixel(columna,fila2)==NEGRO))
        {
```

```
            _setcolor(CIAN);
            _setpixel(columna,fila2);
          }
         else
          if (_getpixel(columna,fila2)==MAGNETA)
            {
             _setcolor(MAGNETA);/* MAGNETA*/
             _setpixel(columna,fila2);
            }
         }
    }
punteroon();
 for(i=0;i<1500000;i++)
 {
 c=4*4;
 }
}
```

```
/* PROGRAMA QUE DIBUJA LOS BOTONES QUE APARECEN EN LOS
MENUS */

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned char BOOLEAN;

#include <malloc.h>
#include <ctype.h>
#include "nidaq.h"
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>

#define SLOT 1
#define TRUE 1
#define FALSE 0
#define NEGRO 0
#define BLANCO 15
#define MAX 15

void boton(int,int,int,int,char *, BOOLEAN);
void boton(int x1,int y1, int x2,int y2, char *cadena, BOOLEAN todo)
{
 if (todo)
   {
    _setcolor(0);
    _rectangle(_GBORDER,x1,y1,x2,y2);
    _setcolor(15);
    _rectangle(_GFILLINTERIOR,x1+1,y1+1,x2-1,y2-1);
    _remappalette(1,0x2e2e2e);
    _setcolor(1);
```

```
    _rectangle(_GFILLINTERIOR,x1+3,y1+3,x2-1,y2-1);
    _moveto(x1+5,y1+2);
    _setcolor(0);
    _outgtext(cadena);
   }
  else
   {
   _remappalette(1,0x2e2e2e);
   _setcolor(1);
   _rectangle(_GFILLINTERIOR,x1+1,y1+1,x2-1,y2-1);
   _moveto(x1+5,y1+2);
   _setcolor(0);
   punteroff();
   _outgtext(cadena);
    punteroon();
   }
}
```

```
/* FUNCIÓN QUE DIBUJA UN BOTON COMO EL DE WINDOWS */

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned char BOOLEAN;

#include <stdio.h>
#include <graph.h>
#include <dos.h>
#include <string.h>

void boto1(int,int,int,int,int,char *);

void boto1(int x1,int y1,int x2,int y2, int separacion,char *cadena)
{
 int i;

  for(i=0;i<separacion;i++) /*linea horizontal superior*/
    {
     _moveto(++x1,++y1);
     _lineto(--x2,y1);
    }

   x1=x1-separacion;
   x2=x2+separacion;
   y1=y1-separacion;

  for(i=0;i<separacion;i++)  /*linea vertical izda*/
    {
     _moveto(++x1,++y1);
     _lineto(x1,--y2);
    }
   x1=x1-separacion;
   y1=y1-separacion;
   y2=y2+separacion;

  _setcolor(8);

  for(i=0;i<separacion;i++)  /*linea horizontal inferior*/
    {
     _moveto(++x1,--y2);
     _lineto(--x2,y2);
    }
   x1=x1-separacion;
   x2=x2+separacion;
   y2=y2+separacion;
```

```
  for(i=0;i<separacion;i++) /*linea vertical dcha*/
    {
     _moveto(--x2,++y1);
     _lineto(x2,--y2);
     }


   x2=x2+separacion;
   y1=y1-separacion;
   y2=y2+separacion;
   _setcolor(7);
   _rectangle(3,x1+(separacion+1),y1+(separacion+1),x2-(separacion+1),y2-
(separacion+1));
   for(i=0;i<32000;i++)
   {
   }
     _setcolor(0);
     _moveto(x1+(separacion+2),y1+(separacion+4));
     _outgtext(cadena);

}
```

```
/*****************************************************************
        void Errprint(Nom_Proc,err_num)
        Esta función imprime el código de los mensajes de error de las funciones
        NI-DAQ de DOS

*****************************************************************
/
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include "colores.h"
#include <string.h>
void Mensajes(char *,char *,char *);

#include "nidaqerr.h"   /* Defenicion de codigos de error */

void ErrPrint(char *Nom_Proc,int err_num)
{
 char *numero;

        if (err_num != noErr)
                itoa(err_num,numero,10);

        switch (err_num)
        {
    case notEnoughExtMem:

 Mensajes(strcat("Error Numero",numero),"La BIOS Indica insuficiente Memoria
                                        Extendida","Para asignar
MDAQ_Ext_Setup");
        break;
```

```
case inputModeConflict:
    Mensajes(strcat("Error Numero",numero),"","Para asignar

   MDAQ_Ext_Setup");
    break;
   case DMAReprogramming:
    Mensajes(strcat("Error Numero",numero),"El buffer asignado requiere
                                    reprogramacion","de la DMA en run-time");
    break ;
    case pageBreakinDMAbuf:
    Mensajes(strcat("Error Numero",numero),"En modo DMA, la conmutación
                de paginas podría","causar glitches en la generación de ondas");
    break;

    case overWriteBeforeCopy:
    Mensajes(strcat("Error Numero",numero),"Los datos han sido
                                    sobrescritos","antes de comenzar la copia");
    break;
   case noPreTrigUnwrap:
    Mensajes(strcat("Error Numero",numero),"No encuentra suficiente memoria
                        para soportar","empaquetado de datos pre-trigger");
    break;
    case calibrationFailed:
    Mensajes(strcat("Error Numero",numero),"Falla la calibración de ganancia y
                                                    calibración"," ");
    break;
    case readOutputPort:
    Mensajes(strcat("Error Numero",numero),"Un puerto digital configurado
para                                                salida ha sido leído"," ");
    break;
   case dupDMALevels:
```

```
Mensajes(strcat("Error Numero",numero),"Dos o mas Placas tienen el mismo
                                                    nivel de DMA"," ");
break;

        case dupIntLevels:
         Mensajes(strcat("Error Numero",numero),"Dos o mas placas tienen el
                                                mismo","nivel de interrupción");
         break;
         case dupIOaddrRange:
        Mensajes(strcat("Error Numero",numero),"Dos o mas placas tienen
                                superpuesto el espacio","de direcciones I/O");
        break;
        case noErr:
        break;
        case notOurBrdErr:
        Mensajes(strcat("Error Numero",numero),"La placa en el slot
                especificado no es una","placa de la serie MC,AT o EISA");
break;
        case badBrdNumErr:
        Mensajes(strcat("Error Numero", numero),"El numero del slot esta
                                                    fuera de rango","
");
        break;
        case badGainErr:
        Mensajes(strcat("Error Numero",numero),"El valor de la ganancia esta
                                                    fuera de rango"," ");
        break;
       case badChanErr:
       Mensajes(strcat("Error Numero",numero),"El numero del canal esta
                                                    fuera de rango"," ");
       break;
      case noSupportErr:
```

```
                Mensajes(strcat("Error Numero",numero),"La función no esta
                                                    soportada por la placa"," ");
                break;
                case badPortErr:
                Mensajes(strcat("Error Numero",numero),"El numero del puerto esta
                                                    fuera de rango","");
                break;
                case badOutPortErr:
                Mensajes(strcat("Error Numero",numero),"El puerto especificado no
                esta","configurado como salida");
                break;
                case noLatchModeErr:
                Mensajes(strcat("Error Numero",numero),"El puerto no ha sido
                        configurado para","modo de cerrojo <<LATCHED>>");
                break;
                case noGroupAssign:
                Mensajes(strcat("Error Numero",numero),"El puerto no puede ser
                                                    asignado al grupo"," ");
                break;
                case badInputValErr:
                Mensajes(strcat("Error Numero",numero),"Uno o mas parámetros de
                                                    entrada fuera","de
rango.");
                break;
                case timeOutErr:
                Mensajes(strcat("Error Numero",numero),"Call timed out"," ");
                break;
                case outOfRangeErr:
                Mensajes(strcat("Error Numero",numero),"Un voltaje esta fuera de
                                                    rango"," ");
                break;
                case daqInProgErr:
```

```
Mensajes(strcat("Error Numero",numero),"La adquisición esta en
                                progreso","la llamada no se ejecutara");
break;
case noDAQErr:
Mensajes(strcat("Error Numero",numero),"No hay adquisición en
                            progreso la","llamada no ha tenido efecto");
break;

 case overFlowErr:
 Mensajes(strcat("Error Numero",numero),"Ha ocurrido un overflow en
                                la FIFO del A/D"," ");
 break;
 case overRunErr:
 Mensajes(strcat("Error Numero",numero),"Se ha excedido el mínimo
                                intervalo","de muestreo
");
 break;
 case brdTypeErr:
Mensajes(strcat("Error Numero",numero),"Tipo de placa no valida para
                                operar"," ");
 break;
 case wfInProgErr:
 Mensajes(strcat("Error Numero",numero),"Esto es una generación de
                            forma","de onda en progreso ");
 break;
 case noWfLoadErr:
Mensajes(strcat("Error Numero",numero),"No ha sido cargada forma de
                                onda"," ");
 break;
 case noWfInProgErr:
 Mensajes(strcat("Error Numero",numero)," No hay doble buffer","");
 break;
```

```
case badPreTrigCntErr:
Mensajes(strcat("Error Numero",numero),"Numero de puntos después
                                        de","prettriger no valido ");
break;
case badSigDirErr:
Mensajes(strcat("Error Numero",numero),"La señal al RTSI tiene","una
                                        dirección errónea ");
break;
case noDbDaqErr:

Mensajes(strcat("Error Numero",numero),"No hay adquisición de doble
                                        Buffer","en progreso ");
break;
case overWriteErr:
Mensajes(strcat("Error Numero",numero),"Los datos han sido
                                        sobrescritos"," ");
break;
case memErr:
Mensajes(strcat("Error Numero",numero),"No hay memoria o espacio
                                        en el","disco disponible ");
break;
case noConfigFile:
Mensajes(strcat("Error Numero",numero),"Fichero de configuración
                                        no","encontrado ");
break;
case intLevelInUse:
Mensajes(strcat("Error Numero",numero),"La interrupción asignada
es                                      usada","por otra placa");
break;

case DMAChanInUse:
```

```
                Mensajes(strcat("Error Numero",numero),"La DMA asignada es
usada                                                    por","otra placa ");
                break;
                case multSourceInputErr:
                Mensajes(strcat("Error Numero",numero),"Dos señales del RTSI no
                                        pueden","ser conducidas como entradas ");
                break;
                case noConnectionErr:
                Mensajes(strcat("Error Numero",numero),"El camino del RTSI no
                                                         est ","especificado ");
                break;
                case noPGInProg:
                Mensajes(strcat("Error Numero",numero),"No hay generación de
                                                      patron","en progreso ");
                break;
                case PGInProg:
                Mensajes(strcat("Error Numero",numero),"La generación del patrón
                                                         esta","en progreso ");
               break;
                case grpRateErr:
                Mensajes(strcat("Error Numero",numero),"La placa no soporta esa
                                                              frecuencia"," ");
                break;
                case openFileErr:
                Mensajes(strcat("Error Numero",numero),"No se pudo abrir el
                                                              fichero"," ");
                break;
                case writeFileErr:
                Mensajes(strcat("Error Numero",numero),"No se pudo escribir en el
                                                              fichero"," ");
                break;
                case noDbWvfmErr:
```

```
                Mensajes(strcat("Error Numero",numero),"No hay doble buffer en
                                                                marcha"," ");
                break;
                case oldDataErr:
                Mensajes(strcat("Error Numero",numero),"La generación se ha
parado
        ","");
                break;
                case dataNotAvailErr:
                Mensajes(strcat("Error Numero",numero),"El bloque requerido no
esta                                                            ","disponible ");
                break;
                case DMATransferCntNotAvail:
                Mensajes(strcat("Error Numero",numero),"Ha fallado la transferencia
                                                                DMA "," ");
                break;
               case noSetupErr:
                Mensajes(strcat("Error Numero",numero),"MDAQ_Setup debe ser
                                        llamado antes","de MDAQ_Start");
                break;
                 case triggerSourceErr:
                Mensajes(strcat("Error Numero",numero),"Debe ser modo pretrigger
                                para","recibir un disparo sobre el RTSI ");
                break;
               case noTrigEnabledErr:
                Mensajes(strcat("Error Numero",numero),"Se debe habilitar un
                                        disparo hardware","en modo
pretrigger");
                break;
               case preTrigScansErr:
                Mensajes(strcat("Error Numero",numero),"Numero de scans pre-
triger                                                          ","no valido ");
```

```
                break;
               case postTrigScansErr:
                Mensajes(strcat("Error Numero",numero),"Numero de scans pos-
triger                                                    ","no valido ");
                break;
                case scanRateErr:
               Mensajes(strcat("Error Numero",numero),"Velocidad demasiado
                                              elevada ","para el numero de canales ");
                break;
                   case invalidGetErr:
                Mensajes(strcat("Error Numero",numero),"Error en los par metros de
                                                          MDAQ_Get "," ");
                break;
                case calInputOutOfRange:
                Mensajes(strcat("Error Numero",numero),"Referencia externa ","fuera
                                                          de rango ");
                break;
                   case calResponseErr:
                Mensajes(strcat("Error Numero",numero),"Error durante la
calibración                                                              ","
");
                break;
                case calConvergeErr:
               Mensajes(strcat("Error Numero",numero),"Problemas durante la
                                                          calibración "," ");
                break;
                case calDACerr:
               Mensajes(strcat("Error Numero",numero),"Valor del DAC erroneo
                                              generado ","durante la calibración ");

                break;
                   case externalCalRefErr:
```

```
                Mensajes(strcat("Error Numero",numero),"La referencia externa es
                                                        ","erronea ");
                break;
                    case internalCalRefErr:
                Mensajes(strcat("Error Numero",numero),"La referencia interna es
                                                        erronea "," ");
                break;
            case badOutLineErr:
                Mensajes(strcat("Error Numero",numero),"Línea de salida
configurada                                             ","como entrada
");
                break;
            case dacUpdateErr:
                Mensajes(strcat("Error Numero",numero),"El DAC ha sobrescrito los
                                                        datos "," ");
                break;
            case interlvdDataAlignErr:
                Mensajes(strcat("Error Numero",numero),"Error en de DMA "," ");
                break;
                    case cannotAlignBufErr:
                Mensajes(strcat("Error Numero",numero),"El buffer no es suficiente
                                                grande ","para los datos  ");
                break;
                    case configFileErr:
                Mensajes(strcat("Error Numero",numero),"Los datos en fichero de
                                            configuración ","se han borrado ");
                break ;
                    case wvfmGrpAssignErr:
                Mensajes(strcat("Error Numero",numero),"Error al asignar los canales
                                                        de salida "," ");
                break ;
                    case chanNotAssignedGrpErr:
```

```
                Mensajes(strcat("Error Numero",numero),"El canal de salida no ha
sido                                                    ","asignado adecuadamente");
                 break ;
              case loadAfterStartErr:
                Mensajes(strcat("Error Numero",numero),"Error de DMA consulte el
                                                                manual "," ");
                 break ;
              case noUpdateRateErr:
                Mensajes(strcat("Error Numero",numero),"Debe especificar una
                                        frecuencia ","antes de llamar a START ");
                 break ;
                case invalidMemRegionErr:
                   Mensajes(strcat("Error Numero",numero),"El direccionamiento de
                                            la DMA ","esta limitado a 16 M ");
                 break;

                default:
                        Mensajes(strcat("Error Numero",numero),"Consulte el manual
                                        para ver  ","el significado de este error");
                }
}

void Mensajes(char *Cadena1,char *Cadena2, char *Cadena3)
{
 long TamaImagen,TamaCadena;
 int y1=200,y2=260,i,x1,x2;
 static char huge *BUFFER=NULL;
 FILE *PF;

  if((PF=tmpfile())==NULL)
  {
   printf("ERROR AL ABRIR EL FICHERO ");
```

```
  exit(1);
 }
if( (strlen(Cadena2)) > (strlen(Cadena3)) )
   TamaCadena=(strlen(Cadena2)*5L);
else
   TamaCadena=(strlen(Cadena3)*5L);
x1=((int)640-(int)TamaCadena)/(int)2;
if(x1<0)
        x1=2;
x2=(int)x1+(int)5+(int)TamaCadena;
if(x2>640)
        x2=638;
TamaImagen=_imagesize(x1,y1,x2,y1+10);
BUFFER=(char huge *)halloc(TamaImagen,sizeof(char));
for(i=200;i<261;i+=10)
{
 _getimage(x1,i,x2,i+10,BUFFER);
 fwrite(BUFFER,sizeof(char),(int)TamaImagen,PF);
}
titulos(x1+5,y1+5,x2-5,y2-5,ROJO);
_setfont("t'Tms Rmn' h8w8b");
PoneCadena(x1+8,y1+10,NEGRO,Cadena1);
PoneCadena(x1+8,y1+20,NEGRO,Cadena2);
PoneCadena(x1+8,y1+30,NEGRO,Cadena3);
PoneCadena(x1+8,y1+40,AZULCLARO,"PULSE UNA TECLA");
_setfont("t'Tms Rmn' h13w13b");
getch();
rewind(PF);
for(i=200;i<261;i+=10)
{
 fread(BUFFER,sizeof(char),(int)TamaImagen,PF);
 _putimage(x1,i,BUFFER,_GPSET);
```

```
/* GENERA LA PANTALLA DE PRESENTACION*/

#include <stdio.h>
#include <graph.h>
#include "colores.h"
void botoon(int,int,int,int,int);

main()
{
 int i;
_setvideomode(_MAXRESMODE);
for(i=150;i>5;i-=5)
botoon(40,40,600,400,i);

if(_registerfonts("TMSRB.FON")<0)
{
 printf("ERROR AL CARGAR EL TIPO DE LETRA");
 exit(1);
 }
 _setcolor(ROJO);
 _moveto(95,60);
 _setfont("t'Tms Rmn',h40w40b");
 _outgtext("TRABAJO FIN DE CARRERA");
 _setfont("t'Tms Rmn',h20w20b");
 _setcolor(GRIS);
 _moveto(70,140);
 _outgtext("ESTUDIO DE LA TARJETA PARA PROCESADO");
 _setcolor(7);
 _moveto(71,142);
 _outgtext("ESTUDIO DE LA TARJETA PARA PROCESADO");
 _setcolor(8);
 _moveto(80,170);
```

```
 _outgtext("  DE DE SEÑALES DE AUDIO AT-DSP2200");
 _setcolor(7);
 _moveto(82,171);
 _outgtext("  DE DE SEÑALES DE AUDIO AT-DSP2200");
 _setcolor(15);
 _moveto(80,200);
 _outgtext("     Y DESARROLLO DE UN SOFTWARE");
  _setcolor(7);
 _moveto(82,202);
 _outgtext("     Y DESARROLLO DE UN SOFTWARE");
 _setcolor(NEGRO);
  _setcolor(AZULCLARO);
 _setfont("t'Tms Rmn'h10w10bf");
 _moveto(70,365);
 _outgtext("José, Manuel Sánchez Martín");
  getch();
}

void botoon(int x1,int y1,int x2,int y2, int separacion)
{
 int i;
 _setcolor(15);
  for(i=0;i<separacion;i++) /*linea horizontal superior*/
     {
       _moveto(++x1,++y1);
       _lineto(--x2,y1);
      }
    x1=x1-separacion;
    x2=x2+separacion;
    y1=y1-separacion;
  for(i=0;i<separacion;i++)  /*linea vertical izda*/
     {
```

```
        _moveto(++x1,++y1);
        _lineto(x1,--y2);
        }
      x1=x1-separacion;
      y1=y1-separacion;
      y2=y2+separacion;
     _setcolor(8);
    for(i=0;i<separacion;i++)  /*linea horizontal inferior*/
      {
        _moveto(++x1,--y2);
        _lineto(--x2,y2);
        }
      x1=x1-separacion;
      x2=x2+separacion;
      y2=y2+separacion;
    for(i=0;i<separacion;i++) /*linea vertical dcha*/
      {
        _moveto(--x2,++y1);
        _lineto(x2,--y2);
        }
      x2=x2+separacion;
      y1=y1-separacion;
      y2=y2+separacion;
      _setcolor(7);
      _rectangle(3,x1+(separacion+1),y1+(separacion+1),x2-(separacion+1),y2-
(separacion+1));
      for(i=0;i<32000;i++) /* retardo */
      {
      }
}
```

```
/* CONVIERTE UN FICHERO DSP A FORMATO .WAV*/

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <string.h>
#include <graph.h>
#include "colores.h"
#include <conio.h>

int ExistFile(char *);

main(int argc, char *argv[])
{
 struct CABECERA /*estructura que contiene la cabecera del wav*/
 {
  char NombreFile[4];
  unsigned long TamaChuck;
  char Tipofile[4];
  char NomSeguChuck[4];
  unsigned long PrimeDigControl;
  int seguDigControl;
  int NuCanales;
  unsigned long FrecuMuestreo;
  unsigned long VeloTransfe;
  int AlinTrama;
  int Nbits;
  char datos[4];
  unsigned long LongDatos;
  }cabeza1;

struct
```

```
{
 int NumeroCanales;
 unsigned int Fm;
 }cabeza2;

 FILE *pf,*pf1;
 int valor,i,j;
 char car;
 unsigned long contador=0,total,resto;
 short oldsuperficie;
 long oldfondo;
 struct rccoord oldpostext;
 if(argc!=3)
 {
  printf("Este programa transforma un fichero de la AT-DSP2200 en un fichero
WAV\n");
  printf("Formato: Fichero AT-DSP2200  -> Fichero WAV");
  exit(1);
 }
/*salva la superficie el fondo y posicion original del texto*/
   oldsuperficie=_gettextcolor();
  oldfondo=_getbkcolor();
  oldpostext=_gettextposition();
  _clearscreen(_GCLEARSCREEN);
  _displaycursor(_GCURSOROFF);
  _setbkcolor(1L);   /*dibuja el fondo azul*/
  for(i=1;i<26;i++)
   for(j=1;j<80;j++)
   {
    _settextposition(i,j);
    _outtext(" ");
   }
```

```
  _setbkcolor(14L); /*cabecera*/
  for(i=1;i<3;i++)
   for(j=1;j<80;j++)
   {
     _settextposition(i,j);
     _outtext(" ");
   }
  _settextcolor(7);
  _settextposition(1,10);
  _outtext("UTILIDA QUE PERMITE TRANSFORMAR UN FICHERO DE LA
AT-DSP2200");
  _settextposition(2,17);
  _outtext("EN UN FICHERO CON FORMATO WAV DE WINDOWS");

  if(ExistFile(argv[2])==1)
  {
    _setbkcolor(4L);
    for(i=0;i<80;i++)
     {
      _settextposition(10,i);
      _outtext(" ");
     }
    _settextposition(10,1);
    _outtext(" EL FICHERO:");
    _settextposition(10,13);
    _outtext(argv[2]);
    _settextposition(10,25);
    _outtext("YA EXISTE, DESEA SOBRESCRIBIRLO S/N");
   _displaycursor(_GCURSORON);
    do
    {
     car=getch();
```

```
    }
    while ((toupper(car)!='S') && (toupper(car)!='N'));
    _displaycursor(_GCURSOROFF);
  }
    if(toupper(car)=='N')
     {
      _settextcolor(oldsuperficie);
      _setbkcolor(oldfondo);
      _clearscreen(_GCLEARSCREEN);
      _settextposition(oldpostext.row,oldpostext.col);
      _displaycursor(_GCURSORON);
      exit(1);
     }
_setbkcolor(14L);
  for(i=0;i<80;i++)
  {
   _settextposition(10,i);
   _outtext(" ");
  }
  _settextposition(10,1);
  _outtext(" FICHERO DE ENTRADA:");
  _settextposition(10,21);
  _outtext(argv[1]);
_settextposition(10,45);
  _outtext("FICHERO DE SALIDA:");
  _settextposition(10,63);
  _outtext(argv[2]);
_setbkcolor(14L);
  for(i=25;i<26;i++)
   for(j=1;j<80;j++)
   {
    _settextposition(i,j);
```

```
    _outtext(" ");
   }
  _settextcolor(7);
  _settextposition(25,20);
  _outtext("TRANSFORMACION EN CURSO, ESPERE POR FAVOR");
  _setbkcolor(4L);
 _settextposition(14,28);
 for(i=0;i<21;i++)
 _outtext("Í");
 _settextposition(16,28);
 for(i=0;i<21;i++)
 _outtext("Í");
 _settextposition(14,28);
 _outtext("É");

 _settextposition(15,28);
 _outtext("º");

 _settextposition(16,28);
 _outtext("È");

 _settextposition(14,49);
 _outtext("»");

 _settextposition(15,49);
 _outtext("º");

 _settextposition(16,49);
 _outtext("¼");
 _settextposition(15,29);
 for(i=0;i<20;i++)
 _outtext(" ");
```

```
_setbkcolor(4L);
_settextposition(15,29);


strcpy(cabeza1.NombreFile,"RIFF");
strcpy(cabeza1.Tipofile,"WAVE");
strcpy(cabeza1.NomSeguChuck,"fmt ");
cabeza1.PrimeDigControl=16;
cabeza1.seguDigControl=1;
cabeza1.Nbits=16;
strcpy(cabeza1.datos,"data");
  if((pf1=fopen(argv[1],"r+b"))==NULL)
 {
 _settextcolor(oldsuperficie);
 _setbkcolor(oldfondo);
 _clearscreen(_GCLEARSCREEN);
 _settextposition(oldpostext.row,oldpostext.col);
 _displaycursor(_GCURSORON);
 printf("Error 1 no se puede abrir el fichero:%s", argv[1]);
 exit(1);
 }

cabeza1.TamaChuck=((filelength(fileno(pf1))+40)-8);
cabeza1.LongDatos=((cabeza1.TamaChuck)-36);

fread(&cabeza2,sizeof(cabeza2),1,pf1);
cabeza1.FrecuMuestreo= (unsigned long)cabeza2.Fm;
cabeza1.NuCanales=cabeza2.NumeroCanales;

if(cabeza2.NumeroCanales==1)
{
   cabeza1.AlinTrama=2;
```

```
    cabeza1.VeloTransfe= (unsigned long)(cabeza2.Fm)*2;
 }
 if(cabeza2.NumeroCanales==2)
 {
    cabeza1.AlinTrama=4;
    cabeza1.VeloTransfe=(unsigned long)(cabeza2.Fm)*4;
 }
if((pf=fopen(argv[2],"w+b"))==NULL)
{
  _settextcolor(oldsuperficie);
  _setbkcolor(oldfondo);
  _clearscreen(_GCLEARSCREEN);
  _settextposition(oldpostext.row,oldpostext.col);
  _displaycursor(_GCURSORON);
  printf("Error 2 no se puede abrir el fichero:%s", argv[2]);
  exit(1);
 }
 fwrite(&cabeza1,sizeof(cabeza1),1,pf);
 total=filelength(fileno(pf1));
 resto=total/40;
while( fread(&valor,sizeof(int),1,pf1)!=0)
{
 fwrite(&valor,sizeof(int),1,pf);
 contador++;
 if((contador % resto)==0)
 _outtext("°");
}
 fclose(pf);
 fclose(pf1);

 _setbkcolor(4L);
 for(i=25;i<26;i++)
```

```
    for(j=1;j<80;j++)
    {
     _settextposition(i,j);
     _outtext(" ");
    }
   _settextcolor(7);
   _settextposition(25,20);
   _outtext("TRANSFORMACION FINALIZADA, PULSE UNA TECLA");
   printf("\x07");
   getch();
   /*restauramos el la pantalla original*/
   _settextcolor(oldsuperficie);
   _setbkcolor(oldfondo);
   _clearscreen(_GCLEARSCREEN);
   _settextposition(oldpostext.row,oldpostext.col);
   _displaycursor(_GCURSORON);
}

int ExistFile(char *NombreFichero)
{
 char NombreRuta[_MAX_PATH];
 int existe;
 char entorno[]="COMPILAR";

 _searchenv(NombreFichero,entorno,NombreRuta);
 if(*NombreRuta!='\0')
      existe=1; /*EL FICHERO NO EXISTE*/
 else
      existe=0; /*EL FICHERO EXISTE*/
 return(existe);
}
```

```
/*CONVIERTE UN FICHERO .WAV A FORMATO .DSP*/

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <string.h>
#include <graph.h>
#include "colores.h"
#include <conio.h>

int ExistFile(char *);

main(int argc, char *argv[])
{

 struct CABECERA /*estructura que contiene la cabecera del wav*/
 {
  char NombreFile[4];
  unsigned long TamaChuck;
  char Tipofile[4];
  char NomSeguChuck[4];
  unsigned long PrimeDigControl;
  int seguDigControl;
  int NuCanales;
  unsigned long FrecuMuestreo;
  unsigned long VeloTransfe;
  int AlinTrama;
  int Nbits;
  char datos[4];
  unsigned long LongDatos;
  }cabeza1;
  char wave[2];
  unsigned long frecuencias[12]={4000,5513,6400,8000,11025,12800,16000,22050,
                                 25600,32000,44100,51200};

 struct
 {
  int NumeroCanales;
  unsigned int Fm;
  }cabeza2;

  FILE *pf,*pf1;
  int valor,i,j,control=0;
  char car;
  unsigned long contador=0,total,resto;
  short oldsuperficie;
  long oldfondo;
  struct rccoord oldpostext;
```

```
 if(argc!=3)
 {
  printf("ESTE PROGRAMA TRANSFORMA UN FICHERO .WAV EN UN
FICHERO DSP\n");
  printf("Formato: Fichero WAV  -> Fichero AT-DSP2200");
  exit(1);
 }


 /*salva la superficie el fondo y posicion original del texto*/

  oldsuperficie=_gettextcolor();
  oldfondo=_getbkcolor();
  oldpostext=_gettextposition();
  _clearscreen(_GCLEARSCREEN);

  _displaycursor(_GCURSOROFF);

  if((pf1=fopen(argv[1],"r+b"))==NULL)
   {
   _settextcolor(oldsuperficie);
   _setbkcolor(oldfondo);
   _clearscreen(_GCLEARSCREEN);
   _settextposition(oldpostext.row,oldpostext.col);
   _displaycursor(_GCURSORON);
   printf("Error 1 no se puede abrir el fichero:%s", argv[1]);
   exit(1);

   }
  fread(&cabeza1,sizeof(cabeza1),1,pf1);
  /*comprobar que es un wav*/
  if(cabeza1.Nbits!=16)
  {
   _settextcolor(oldsuperficie);
   _setbkcolor(oldfondo);
   _clearscreen(_GCLEARSCREEN);
   _settextposition(oldpostext.row,oldpostext.col);
   _displaycursor(_GCURSORON);
   printf("Error: El fichero %s no es de 16 bits", argv[1]);
   exit(1);
  }
  cabeza2.NumeroCanales=cabeza1.NuCanales;
  cabeza2.Fm=(unsigned int)cabeza1.FrecuMuestreo;

  for(i=0;i<12;i++)
  {
   if(cabeza1.FrecuMuestreo==frecuencias[i])
   control=1;
   }
```

```
   if(control==0)
   {
   _settextcolor(oldsuperficie);
   _setbkcolor(oldfondo);
   _clearscreen(_GCLEARSCREEN);
   _settextposition(oldpostext.row,oldpostext.col);
   _displaycursor(_GCURSORON);
   printf("ERROR: LA FRECUENCIA %ld NO ES ACEPTADA POR LA AT-
DSP",cabeza1.FrecuMuestreo);
   exit(1);

   }

_setbkcolor(1L);  /*dibuja el fondo azul*/
  for(i=1;i<26;i++)
   for(j=1;j<80;j++)
   {
     _settextposition(i,j);
     _outtext(" ");
   }

  _setbkcolor(14L); /*cabecera*/
  for(i=1;i<3;i++)
   for(j=1;j<80;j++)
   {
     _settextposition(i,j);
     _outtext(" ");
   }
  _settextcolor(7);
  _settextposition(1,10);
  _outtext("UTILIDAD QUE PERMITE TRANSFORMAR UN FICHERO .WAV
DE WINDOWS");
  _settextposition(2,11);
  _outtext("EN UN FICHERO QUE PUEDA SER REPRODUCIDO POR LA AT-
DSP2200");

  if(ExistFile(argv[2])==1)
  {
    _setbkcolor(4L);
    for(i=0;i<80;i++)
     {
      _settextposition(10,i);
      _outtext(" ");
     }
    _settextposition(10,1);
    _outtext(" EL FICHERO:");
    _settextposition(10,13);
    _outtext(argv[2]);
    _settextposition(10,25);
```

```
  _outtext("YA EXISTE, DESEA SOBRESCRIBIRLO S/N");

 _displaycursor(_GCURSORON);
  do
  {
   car=getch();
  }
  while ((toupper(car)!='S') && (toupper(car)!='N'));
  _displaycursor(_GCURSOROFF);
}
  if(toupper(car)=='N')
   {
    _settextcolor(oldsuperficie);
    _setbkcolor(oldfondo);
    _clearscreen(_GCLEARSCREEN);
    _settextposition(oldpostext.row,oldpostext.col);
    _displaycursor(_GCURSORON);
    exit(1);
   }

 _setbkcolor(14L);
for(i=0;i<80;i++)
{
 _settextposition(10,i);
 _outtext(" ");
}
 _settextposition(10,1);
 _outtext(" FICHERO DE ENTRADA:");
 _settextposition(10,21);
 _outtext(argv[1]);

 _settextposition(10,45);
 _outtext("FICHERO DE SALIDA:");
 _settextposition(10,63);
 _outtext(argv[2]);

 _setbkcolor(14L);
for(i=25;i<26;i++)
 for(j=1;j<80;j++)
 {
   _settextposition(i,j);
   _outtext(" ");
 }
 _settextcolor(7);
 _settextposition(25,20);
 _outtext("TRANSFORMACION EN CURSO, ESPERE POR FAVOR");

 _setbkcolor(4L);
_settextposition(14,28);
```

```
for(i=0;i<21;i++)
_outtext("Í");

_settextposition(16,28);
for(i=0;i<21;i++)
_outtext("Í");

_settextposition(14,28);
_outtext("É");

_settextposition(15,28);
_outtext("º");

_settextposition(16,28);
_outtext("È");

_settextposition(14,49);
_outtext("»");

_settextposition(15,49);
_outtext("º");

_settextposition(16,49);
_outtext("¼");
_settextposition(15,29);
for(i=0;i<20;i++)
_outtext(" ");
_setbkcolor(4L);
_settextposition(15,29);



if((pf=fopen(argv[2],"w+b"))==NULL)
{

 _settextcolor(oldsuperficie);
 _setbkcolor(oldfondo);
 _clearscreen(_GCLEARSCREEN);
 _settextposition(oldpostext.row,oldpostext.col);
 _displaycursor(_GCURSORON);
 printf("Error 2 no se puede abrir el fichero:%s", argv[2]);
 exit(1);


 }

 fwrite(&cabeza2,sizeof(cabeza2),1,pf);
```

```
  total=filelength(fileno(pf1));
  resto=total/40;

 while( fread(&valor,sizeof(int),1,pf1)!=0)
 {
  fwrite(&valor,sizeof(int),1,pf);
  contador++;
  if((contador % resto)==0)
  _outtext("°");
 }
  fclose(pf);
  fclose(pf1);

  _setbkcolor(4L);
  for(i=25;i<26;i++)
   for(j=1;j<80;j++)
   {
    _settextposition(i,j);
    _outtext(" ");
   }

  _settextcolor(7);
  _settextposition(25,20);
  _outtext("TRANSFORMACION FINALIZADA, PULSE UNA TECLA");
  printf("\x07");
  getch();
  /*restauramos el la pantalla original*/

  _settextcolor(oldsuperficie);
  _setbkcolor(oldfondo);
  _clearscreen(_GCLEARSCREEN);
  _settextposition(oldpostext.row,oldpostext.col);
  _displaycursor(_GCURSORON);
}

int ExistFile(char *NombreFichero)
{

 char NombreRuta[_MAX_PATH];
 int existe;
 char entorno[]="COMPILAR";

 _searchenv(NombreFichero,entorno,NombreRuta);
 if(*NombreRuta!='\0')
      existe=1; /*EL FICHERO NO EXISTE*/
 else
      existe=0; /*EL FICHERO EXISTE*/
 return(existe);
}
```

```
/* FICHERO CABECERA CON LA DEFINICIÓN DEL MENÚ */

#define TRUE 1
#define FALSE 0
/*--------------------definicion de tipos-------------------------*/
typedef char cadena[15];
typedef unsigned char BYTE;
typedef unsigned int  WORD;
typedef unsigned char BOOLEAN;
void menu(struct REGSMENU);
 struct OPCIONES
  {
   cadena        Opcion;
   BYTE        CodOpcion;
   BOOLEAN   OpcionActiva;
  };
 struct REGSMENU
 {
   WORD   XInin,XFinal;
   cadena     Titulo;
   WORD    TamanoMenu;
   char        *ptr;
   BOOLEAN    Activo;
   BYTE       Elecs;
   struct OPCIONES ListaOpciones[5];
  };

struct REGSMENU DesMenu[4]={18,
                    80,
                    "Fichero",
                    11,
                    NULL,
```

```
{"Cargar",21,TRUE,
"Borrar",23,TRUE,
"Salir",24,TRUE,
"Ir al DOS",19,TRUE,
"Salvar",22,TRUE},
114,
170,
"Editar",
11,
NULL,
FALSE,
4,
{"Reproducir",31,TRUE,
"Borrar",32,TRUE,
"Copiar",33,TRUE,
"Zoom",34,TRUE},
206,
295,
"E/S",
11,
NULL,
TRUE,
3,
{"Adquirir",42,TRUE,
"Genera Rapido",41,TRUE,
"Generar",43,TRUE},
308,
381,
"Procesar",
11,
NULL,
TRUE,
```

```
3,
{"Butterworth",51,TRUE,
"Chebyshev",52,TRUE,
"Eliptico",53,TRUE}};
```

ANEXO III

1.1 INSTALACIÓN Y CONFIGURACIÓN

Para realizar la instalación del software debemos hacer lo siguiente; introducimos el disco en la disquetera a: y tecleamos instalar, el programa de instalación se encargará de realizar la instalación.

El programa se instala en el directorio DSP2200 y se copiaran los ficheros que se indican a continuación.

- **atdsp.exe**: este es el fichero ejecutable del programa que yo he creado.
- **tmsrb.fon**: fichero de tipos de letras.
- **courb.fon**: fichero de tipos de letras.
- **daqconf.exe** : fichero de configuración de la tarjeta.
- **dsp2200.out**: fichero que contiene las librerías para ser cargado en la tarjeta.
- **dsp-wav.exe**: fichero de conversión de formatos .DSP a .WAV.
- **wav-dsp.exe**: fichero de conversión de formatos .WAV a .DSP.
- **visor.exe**: visor de ficheros.

Cuando hayamos terminado la instalación lo que primero que debemos hacer es configurar la tarjeta e indicar donde esta el fichero dsp2200.out. Para ello ejecutamos el fichero atbrds.exe y configuramos las tarjeta según se indica en el capítulo 4, apartado 4.1.4. La configuración crea un fichero llamado atbrds.cfg, este fichero debe estar en el directorio raíz para que sea encontrado por el programa principal sin dificultad.

También es importante que antes de ejecutar el programa se compruebe que existe memoria extendida en una cantidad de al menos 4 Megas. Esto es muy importante, pues si no existe esta memoria la adquisición no funciona. Para configurar la memoria extendida debemos hacerlo con el controlador HIMEM.SYS del DOS. Cuando ejecutamos Windows se carga el controlador de Windows y el programa no funciona.

1.2 USO DEL PROGRAMA ATDSP

Cuando ejecutamos el fichero ATDSP.EXE podemos hacerlo de dos formas, tecleando el nombre de la forma atdsp sin ningún parámetro en cuyo caso el programa se ejecuta y antes pone la pantalla de presentación. Si no queremos ver la pantalla de presentación debemos teclear atdsp sp. Entonces la ejecución se produce sin cargar la pantalla.

Cuando el programa se haya cargado aparecerá una pantalla azul con una línea de menú en la parte superior que tendrá las siguientes opciones.

- Ficheros.
- Editar.
- E/S
- Filtrar.

Para activar uno de estos menús lo que tenemos que hacer es situarnos con el ratón sobre uno de estos nombres y pulsar la tecla izquierda del ratón. Cuando hacemos esto se despliega el menú que nos permite elegir una opción desplazando el puntero del ratón hacia abajo o hacia arriba. En las siguientes líneas describiremos cada uno de los apartados del menú.

1.2.1 Cargar

Esta opción del menú fichero nos permite cargar un fichero y verlo en pantalla. Cuando elegimos esta opción aparece un menú de carga con el listado de los ficheros del directorio activo. Para cargar un fichero debemos situar el puntero del ratón en el recuadro donde pone "NOMBRE DEL FICHERO", el recuadro se borrará se queda a la espera de que introduzcamos un nombre, después pulsamos ENTER. Si el fichero existe se cargará. Para ver más ficheros si no se pueden poner todos en la pantalla podemos pulsar las teclas RePág o AvPág. Para cambiar de directorio colocamos el ratón sobre el recuadro donde está este y escribimos el nombre del directorio al que queremos cambiarnos. Para ir hacia atrás escribimos "..". Sólo debemos escribir el nombre relativo del directorio.

1.2.2 Borrar

Esto nos permite borrar la pantalla.

1.2.3 Salir

Esto nos permite salir del programa y cuando seleccionamos esta opción el programa termina.

1.2.4 Ir al DOS

Esta opción nos permite salir al DOS pero sin abandonar el programa, cuando seleccionamos esta opción la pantalla se guarda en el disco y se sale al DOS. Para regresar al programa tecleamos EXIT desde el DOS.

1.2.5 Oír

Esta opción que pertenece al menú de Edición nos permite oír un trozo de fichero que hayamos seleccionado. Esta opción no esta activa si no se ha cargado un fichero. Para hacer uso de esta opción marcamos la porción de fichero que queramos oír situando el puntero de ratón en el primer punto y pulsando la tecla izquierda del ratón luego marcamos el segundo punto y pulsamos la tecla derecha. Después de esto aparecerá una pequeña pantalla que nos pide que introduzcamos el número de veces que queremos oír el trozo seleccionado.

1.2.5 Cortar

Esta opción que pertenece al menú de Edición nos permite cortar un trozo de fichero que hayamos seleccionado. Seleccionamos la porción a cortar de la misma forma que cuando seleccionamos oír. Antes de cortar se nos pregunta si queremos modificar el fichero original o si queremos crear uno nuevo. Si elegimos esta segunda opción se abre una pantalla para que introduzcamos el nombre. El programa comprueba si el fichero existe y si existe nos lo indica y se nos pide que

introduzcamos otro nombre. Después de todo esto se produce el corte de la porción seleccionada y luego se nos muestra el fichero cortado.

1.2.6 Copiar

Esta opción que pertenece al menú de Edición nos permite copiar un trozo de fichero que hayamos seleccionado. Esta opción nos permite copiar una porción de fichero en otra parte del mismo. El funcionamiento es igual que el apartado anterior.

1.2.7 Zoom

Esta opción que pertenece al menú de Edición nos permite hacer un Zoom de un trozo de fichero que hayamos seleccionado. Marcamos la porción que queramos ampliar y se nos muestra la porción ampliada. Después aparecen tres botones con las opciones OIR, ZOOM y SALIR. La primera opción nos permite oír lo que hemos ampliado, la segunda nos permite seguir ampliando y la tercera salir del zoom.

1.2.8 Adquirir

Esta función pertenece al grupo de E/S y es la encargada de permitirnos adquirir datos y guardarlos en el disco duro. Cuando elegimos esta función aparece una pantalla totalmente controlada por ratón que nos permite seleccionar si usaremos uno u dos canales. Por defecto aparecen marcados los dos canales, si solo vamos a

usar uno nos situamos con el ratón en la casilla del canal que vayamos a eliminar y pulsamos la tecla derecha del ratón, el canal quedará sin seleccionar.

También podemos elegir el acoplamiento AC/DC para ello simplemente marcamos la casilla correspondiente.

Luego seleccionamos la frecuencia de muestreo. La AT-DSP2200 tiene 16 frecuencias de muestreo. Si queremos elegir la frecuencia de 4000 Hz, que no está a la vista hacemos lo siguiente; seleccionamos la frecuencia base mas próxima, 32000 Hz en este caso, y luego seleccionamos división por ocho de forma que 32000 Hz dividido por ocho nos da 4000 Hz.

Hay que tener en cuenta que a mayor frecuencia de muestreo mayor tamaño de los ficheros, en concreto el espacio ocupado por un fichero viene dado por la fórmula siguiente.

$$X = \frac{t * 2 * N}{T}$$

X= Espacio ocupado en el disco.

t= Tiempo en segundos.

N= Número de canales.

T= Periodo de muestreo (1/f).

Otra cosa que debemos indicar es el nombre del fichero en el que se guardaran las muestras. cundo seleccionamos el nombre se comprueba si existe un fichero con

ese nombre en el directorio, si no existe lo crea y si existe nos pide que introduzcamos un nuevo nombre de fichero.

También podemos seleccionar las fuentes de disparo para comenzar la adquisición. Cuan seleccionamos disparo nos sale otra pantalla adicional que nos permite seleccionar el disparo analógico o digital así como el numero de muestras que queremos antes del disparo. Si seleccionamos el disparo analógico tenemos que indicar el canal y el flaco de la señal que producirá el disparo, subida o bajada y también el nivel. El nivel se indica en voltios entre ±2.828 y será la tensión de umbral a partir de la cual se produzca el disparo. Cuando seleccionamos el disparo digital tenemos que indicar solo el flanco de este subida o bajada. Para el disparo digital debemos aplicar la fuente al conector externo.

El número de pretriggers indica el número de muestras que se adquirirán antes del disparo. y no debe ser mayor de 1000. Si seleccionamos pretrigger y no seleccionamos una fuente de disparo se producirá un error.

Cuando todos estos parámetros se hayan completado pulsamos el botón GO! y la adquisición comienza. Durante la adquisición se va indicando el tiempo transcurrido, la capacidad ocupada y el número de tramas adquiridas. Para parar la adquisición pulsamos la tecla ESC o el botón PARAR. Puede ser que en determinados modos, sobre todo con frecuencias bajas, el ratón no responda, en ese caso debemos pulsar ESC para parar.

Si no queremos hacer uso de este procedimiento podemos pulsar el botón SALIR y se anula la opción.

1.2.9 Reproducir rápidamente.

Esta función pertenece al grupo de E/S. Cuando seleccionamos esta opción aparece una un pequeña pantalla que nos permite introducir el nombre del fichero. Después de teclear el nombre y pulsar ENTER el fichero es reproducido por la tarjeta.

1.2.10 Generar

Esta función pertenece al grupo de E/S. Esta función nos permite generar un fichero controlando sus parámetros. Cuando seleccionamos esta opción aparece una pantalla similar a la de adquisición y que debemos completar de la misma forma que esta.

1.2.11 Filtrar

Filtrar es un conjunto de herramientas que nos permite realizar un filtrado de un fichero aplicando uno de los siguientes tipos de filtros; Butterworth, Chebyshev y Elíptico. Cuando seleccionamos uno de estos filtros aparece una pantalla que nos permite introducir los datos del filtro. Los datos pueden ser introducidos mediante teclado o mediante el ratón pulsando sobre las flechas. Para introducir los datos por teclado seleccionamos con el ratón la casilla y luego tecleamos el número. Cuando

hayamos introducido todos los datos podemos pulsar el botón IR! que nos permite ver los coeficientes del filtro. Cuando los hayamos visto pulsamos una tecla y veremos la respuesta en frecuencia del filtro. En este momento podemos salir o filtrar, si elegimos filtrar se nos pide se nos pide que se introduzca el fichero origen y el fichero destino comprobando que existe el fichero origen. En el caso de que el fichero no se pueda filtrar se nos indica con un mensaje.

1.3 USO DE LOS CONVERSORES

Para usar los conversores hacemos lo siguiente.

- Para convertir un fichero DSP a WAV tecleamos:

  DSP-WAV *fichero origen fichero destino.*

  El fichero origen será el fichero con formato DSP y el fichero destino el fichero con formato WAV.

- Para convertir un fichero WAV a DSP tecleamos:

  WAV-DSP *fichero origen fichero destino.*

  El fichero origen será el fichero con formato WAV y el fichero destino el fichero con formato DSP.

1.4 USO DEL VISOR DE FICHEROS

Este programa nos permite ver dos ficheros simultáneamente para usarlo tecleamos desde la línea de comandos lo siguiente.

visor *primer fichero segundo fichero número de canales*

Es importante tener en cuenta que el número de canales debe ser el mismo para ambos canales.

BIBLIOGRAFÍA

BIBLIOGRAFÍA

Para la realización de este trajo he contado con la siguiente bibliografía.

- Manual de Usuario de la AT-DSP2200.
- Manual de Referencia del la NI-DAQ.
- Manual de Referencia de las NI-DSP.
- WE DSP32C Digital Signal Processor.
- Digital Filter Desingn. De la Editorial WILEY-INTERSCIENCE.
- Theory And Application Of Digital Signal Processing. De la Editorial PRENTICE-HALL.
- Microsoft C/C++ Manual de Referencia de la Biblioteca de Ejecución. De la Editorial MICROSOFT PRESS.
- Turbo C/C++ Iniciación y Programación Avanzada. De la editorial PARANINFO.
- Programación En Microsoft C Para el IBM PC y Compatibles. De la Editorial ANAYA MULTIMEDIA.