UNIVERSIDAD AUTONOMA METROPOLITANA
UNIDAD IZTAPALAPA

TESIS DE MAESTRÍA
EN INGENIERÍA BIOMÉDICA

DESARROLLO DE UN SISTEMA AUXILIAR PARA EL PROCESAMIENTO DIGITAL DE SEÑALES BASADO EN EL DSP 56000-1

Víctor Hugo Téllez Arrieta

1995

Area de Procesamiento Digital de Señales e Imagenes Biomédicas
Departamento de Ingeniería Eléctrica

Asesor:
Dr. Joaquin Azpiroz Leehan

REFLEXION:

Todos los seres humanos poseemos una filosofía, unas más ricas que otras; así como hay vidas que trascienden su tiempo, estan las de la gran mayoría que pasan desapercibidas en el transcurrir interminable del tiempo, en el que sus huellas fueron una tenue marca en la arena que con las mas leve brisa se borra para siempre. No se puede aspirar a ser una persona de excelencia a menos que se posea una filosofía lo suficientemente trascendente que lo lleve a permanecer mas alla de su tiempo; de alguien que cuando desaparezca su obra continue, de los cuales cuando dejen de existir se pueda escribir: "nunca fueron vencidos, simplemente murieron".

Víctor Hugo.

Dedicado a Adriano.

AGRADECIMIENTO:

A los que indirectamente en el pasado influyeron para involucrarme dentro de la Maestria en Ingenieria Biomedica:

M. en C. Tomas Aljama Corrales
M. en C. Carlos Aviles Cruz
M. en C. Miguel Cadena Méndez
M. en C. Sonia Charleston Villalobos
Dr. Adriano De Lucca Penachia
Ing.Dpl. Enrique Hernández Matos
M. en C. Caupolican Muñoz Gamboa
M. en C. Marco Antonio Reyna Carranza
M. en C. Oscar Yanez Suárez

Muchas e Infinitas Gracias!

Y a mi esposa Silvia Cristina Iuga

INDICE

1.- Introducción a los Procesadores Digitales de Señales (DSP). 1

2.- Introducción al DSP 56000-1. 9

2.1 Arquitectura del DSP 56000-1. 9

2.1.1 Bus de Datos. 10

2.1.2 Bus de direcciones. 10

2.1.3 ALU de Datos. 11

2.1.4 Unidad Generadora de Direcciones (AGU). 11

2.1.5 Memoria de datos X e Y independientes. 11

2.1.6 Memoria de Programa. 11

2.1.7 Periféricos Internos. 12

2.2 Organización del Modo Host. 13

3.- Desarrollo de una Arquitectura para PC. 17

3.1 Mapeo de Memoria. 17

3.2 Arquitectura del sistema. 21

3.3 Organización de la Memoria X,Y y P. 25

4.- Desarrollo del Software para PC y DSP56000-1.
28
4.1 Inicialización.
28
4.2 Verificación de Interfase.
30
4.3 Transferencia HOST-DSP Paralela de 24 bits.
31
4.4 Transferencia DSP-HOST Paralela de 24 bits.
33
4.5 Carga de Datos a las Memorias X e Y.
35
4.6 Carga de Instrucciones a la Memoria RAM de Programa.
36
4.7 Transferencia Bidireccional de Modo Host.
37

5.- Algoritmos de Prueba, Resultados y Conclusiones.

5.1 Algoritmos de Prueba.
39
5.1.1 Generación de tablas Senos y Cosenos.
39
5.1.2 FFT e IFFT.
40
5.1.3 Transformada Discreta Cosenoidal.
43
5.2 Resultados.
5.3 Conclusiones.

Bibliografía.

Apéndice A.

Apéndice B.

Apéndice C.

1.- Introducción a los Procesadores Digitales de Señales (DSP).

En este tipo de sistemas de procesamiento, la característica fundamental es la capacidad de ejecutar funciones de procesamiento de señales que resultan lentas en computadoras personales de manera eficiente. La tecnología actual permite tomar caminos diferentes para resolver los problemas de velocidad de ejecución en microcomputadoras[1], y una de las opciones que se utilizan constantemente para aumentar la velocidad de procesamiento de señales, es el utilizar procesadores tipo DSP (Digital Signal Processor). Este es el punto a tratar en el presente trabajo.

Algunos ejemplos de procesamiento por parte del DSP son los siguientes:

- Filtrado de señales.
- Convolución: mezcla de dos señales.
- Correlación: comparación de dos señales.
- Rectificación.
- Amplificación.
- Transformación de señales.

Como se mencionó, todas estas funciones tradicionalmente realizadas por sistemas convencionales, resultaban ser lentas, y por otro lado, en algunos casos, eran efectuadas por sistemas analógicos. En el ejemplo de la figura 1.1a se muestra una descripción del procesamiento de una señal analógica.

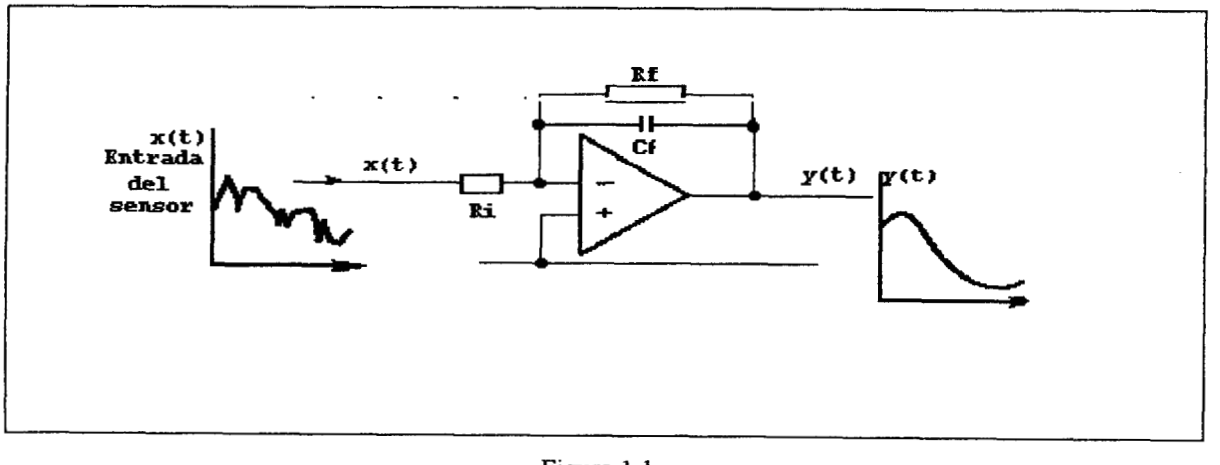


Figura 1.1a

Adquisición analógica.

Aquí la señal x(t) que va al amplificador operacional por la entrada no inversora pasando por una resistencia Ri, es retroalimentada por el paralelo de una resistencia Rf y un Capacitor c_f obteniendo así una impedancia que está en la ecuación 1.1, y la ganancia esta en la ecuación 1.2. para así obtener su salida en y(ω).

$$Rf \prod Cf = \frac{Rf}{(1 + j\omega CfRf)} \qquad (1,1)$$

$$\frac{x(\omega)}{y(\omega)} = \frac{Rf}{(1 + j\omega CfRf)} \qquad (1,2)$$

Haciendo un análisis a la segunda ecuación, la frecuencia de corte del filtro es (Fc):

$$Fc(\omega) = 1/j\omega RfCf \qquad (1,3)$$

donde en condiciones ideales se requiere un característica en frecuencia que se muestra en la gráfica de la figura 1.1b, pero como no es posible el desarrollo en forma ideal de este tipo de filtro, tal como se muestra en la característica de frecuencia real de la figura 1.1c, el diseñador tendrá que hacer lo necesario para hacer un filtro con una respuesta aceptable, y considerando lo siguiente:

1. variaciones en la temperatura
2. variaciones en la fuente de alimentación
3. precisión de los componentes pasivos

Dando como resultando un circuito que tiene poca inmunidad al ruido, que requiere ajustes, y tiene dificultades para modificaciones a futuro.

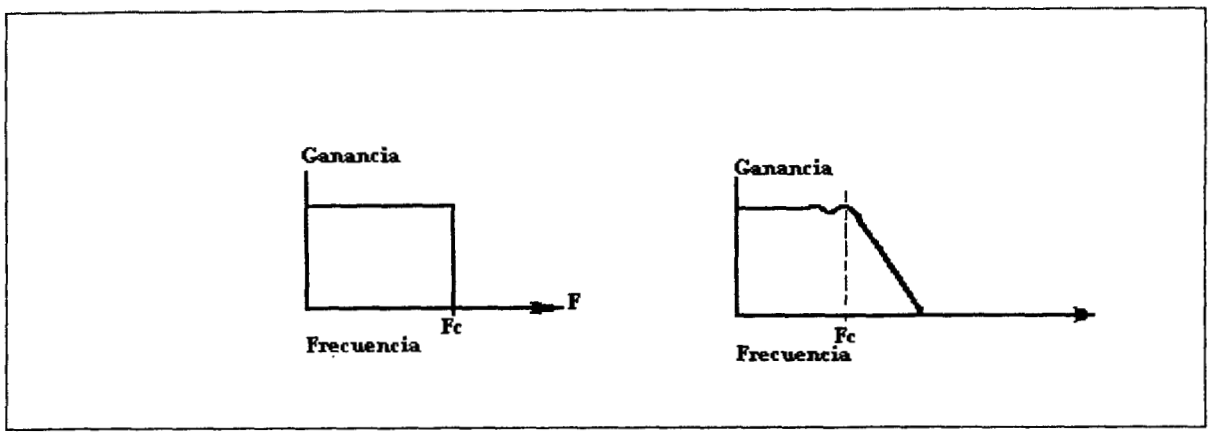


Figura 1.1b Figura 1.1c

Características en frecuencia. Salida Ideal (1.1b)

Salida real del filtro analógico (1.1c)

Pero si se realizase ésta y otras funciones en forma digital usando al DSP, se contaría con la herramienta necesaria para realizarlo en forma más rápida.

En el ejemplo de la figura 1.2 se muestra el funcionamiento utilizando al DSP.

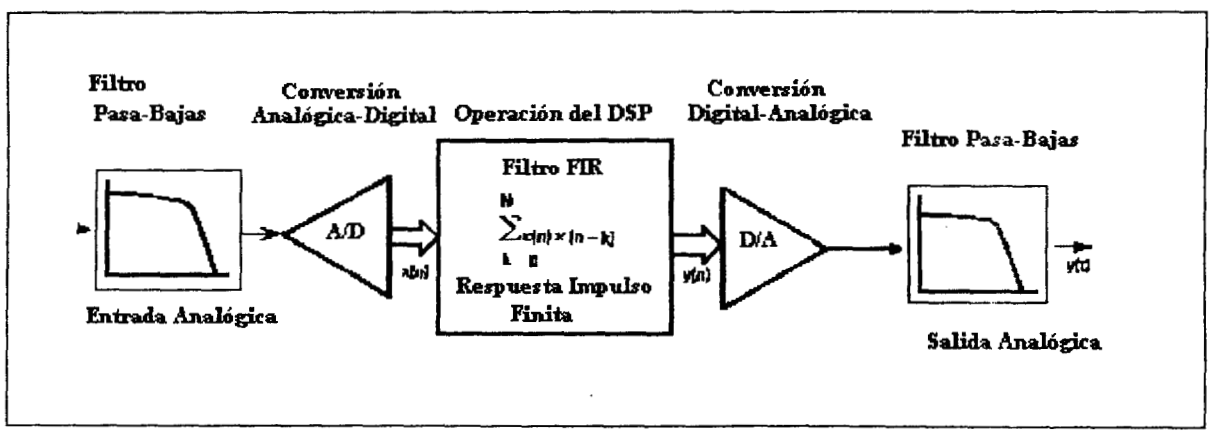


Figura 1.2a

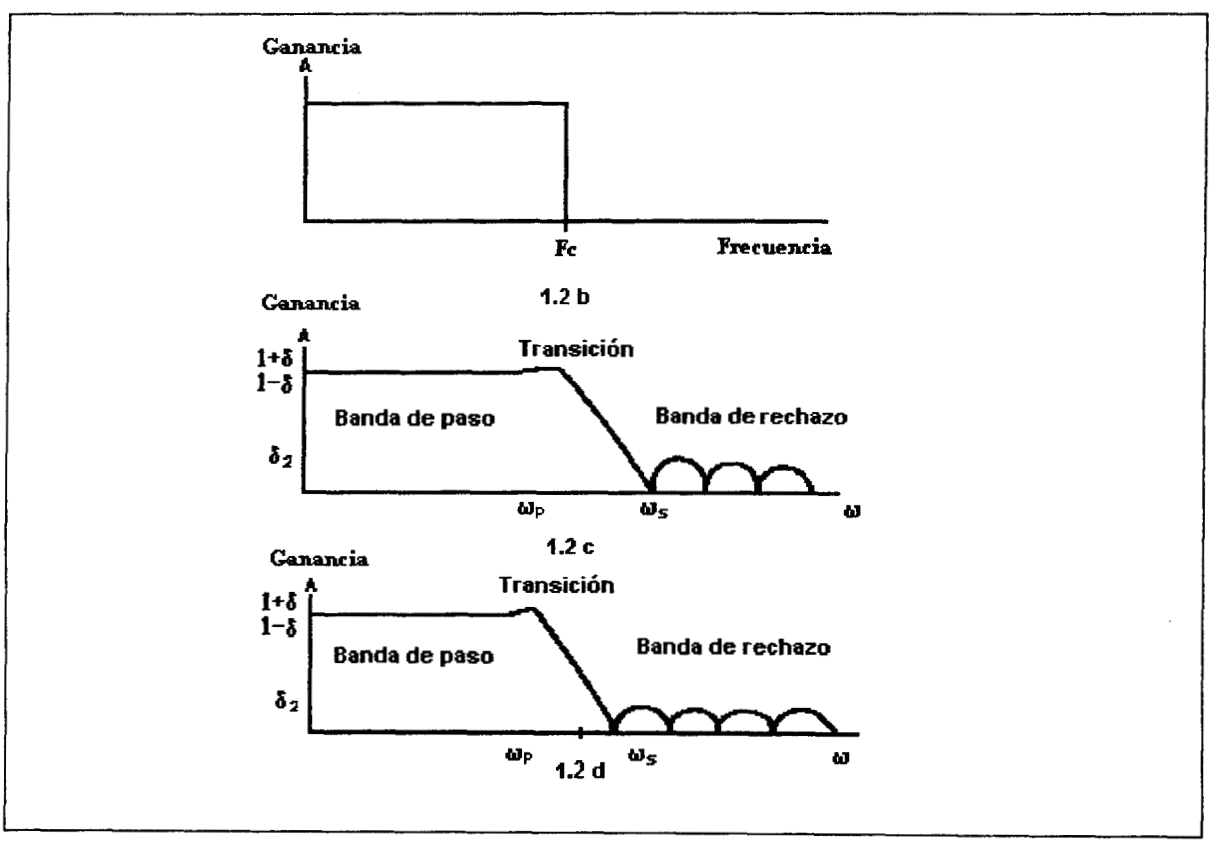
Desarrolllo de un filtro digital por medio de un DSP.

El procesamiento en este circuito empieza limitando en banda la entrada con un filtro pasa bajos, eliminando señales no deseadas. Entonces las señal es muestrada y digitalizada por medio de un convertidor analógico-digital y es enviada al DSP. Como en el DSP se puede programar directamente cualquier

filtro utilizando técnicas digitales, para este ejemplo consiste en diseñarlo con fase lineal, donde la función a implementar esta en la ecuación 1.4 [9].

$$\sum_{k}^{N} C(n)X(n-k) \qquad (1.4)$$

Este filtro es implementado por el producto de la función de transferencia del sistema y los coeficientes, dando como resultado total la suma de todos los productos (de 0 hasta n), siendo aquí el sistema referido, uno de Respuesta Impulso Finita (RIF). Para las especificaciones del filtro hay que referirse a la figura 1.2c y 1.2d, comparando con la figura 1.2b que representa la característica ideal. Se ven unas oscilaciones que en δ1 y δ2 representan la tolerancia de paso y rechazo en banda respectivamente, ωp y ωs son los límites de frecuencia de paso y rechazo respectivamente de una red característica real.



Características en Frecuencia.

También los filtros adaptivos pueden ser fácilmente implementados (éstos son de difícil desarrollo usando técnicas analógicas). La salida del DSP se pasa por un convertidor digital-analógico y un filtro pasa bajos para remover los efectos de digitalización.

En resumen, la ventaja de utilizar DSP es, dentro del dominio del procesamiento en tiempo real, el de transformar funciones analógicas en circuitería digital, y estas ventajas (densidad del circuito, precisión y estabilidad) son ahora disponibles en áreas que tradicionalmente eran usadas por componentes analógicos. Otra ventaja más en los DSP, es el traducir algoritmos de procesamiento del dominio de circuitería al dominio de la programación. Tradicionalmente las funciones analógicas como los filtros, eran reemplazadas por una forma de software, debido a que es muy flexible, fácil de cambiar y más adecuado para sistemas complejos. Finalmente, el procesamiento digital puede usarse en aplicaciones que en la forma analógica no son prácticas.

El DSP abarca un amplio campo de aplicación; por ejemplo, para una sección bicuadrática de una respuesta a impulso infinita de segundo orden en cascada y con cuatro coeficientes, tiene cuatro multiplicandos por cada sección. Para ese algoritmo, el mínimo de operaciones teóricas para una arquitectura de una simple multiplicación es de cuatro por sección.

Como se ve en la figura 1.3, el DSP emplea las siguientes características especiales:

- La operación de Multiplicación/Acumulador (MAC).
- Fetch de operandos por medio del MAC.
- El control del programa permite una operación versátil.
- Entradas/Salidas permite el movimiento de datos hacia y desde el DSP.

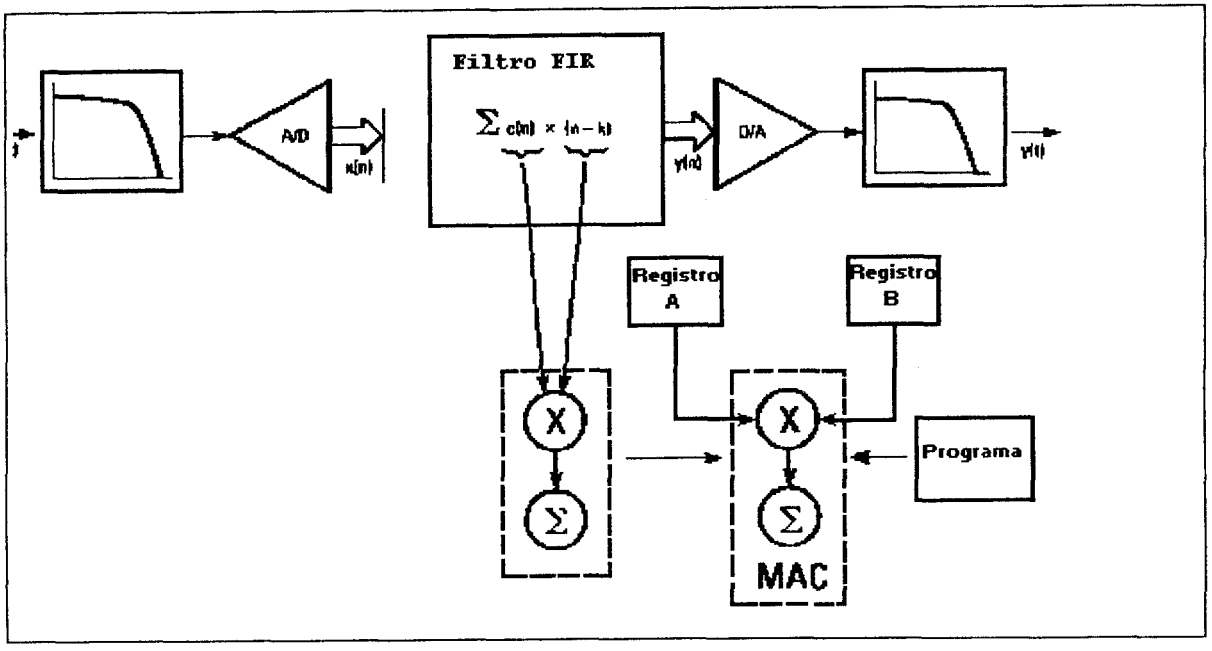


Figura 1.3

Características especiales en el DSP para procesamiento de señales

El MAC es la operación básica utilizada en el DSP, a continuación se describe la forma en que se realiza: Los dos operandos C() y X() son directamente multiplicados en la operación, mientras que el resultado se suma. Este proceso es construído dentro del DSP utilizando dos registros separados para proveer un ciclo MAC sencillo. Los registros y el MAC son independientes, por lo que se pueden ejecutar dos transferencias de datos para multiplicar y acumular en una sola operación.

En la figura 1.4 se muestra como el MAC, los registros y el control de programa se configuran en el DSP. Adicionalmente existen memorias independientes con sus respectivos buses que se utilizan para trasladar operandos al MAC, mientras que al mismo tiempo se realiza el FETCH a un programa (ver apéndice A). La unidad generadora de direcciones (AGU) frecuentemente es dividida en distintas unidades aritméticas que son usadas independientemente para el control de memorias y para suministrar operandos al MAC. Se necesita un bloque adicional para integrar circuitos de entrada/salida, como por ejemplo un convertidor A/D D/A. Con lo anteriormente expuesto el DSP es considerado como una máquina con arquitectura Harvard optimizada para operaciones MAC.

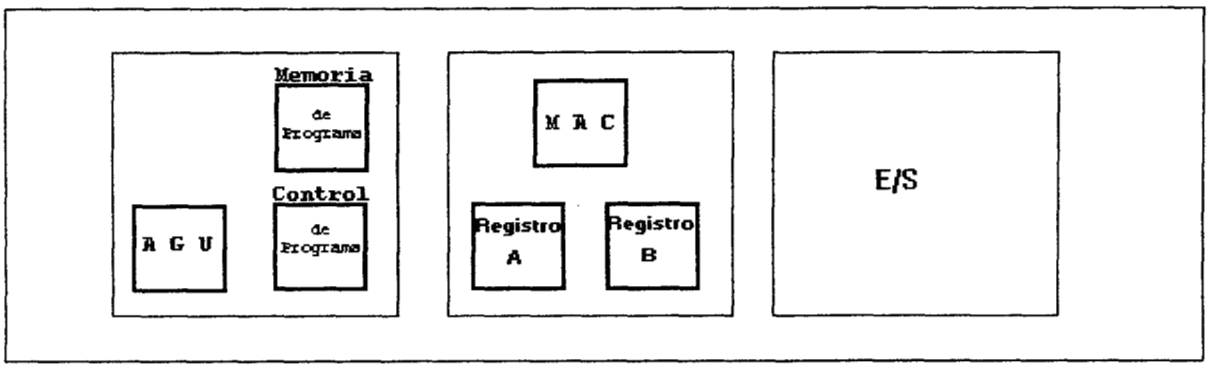


Figura 1.4
Arquitectura básica de un DSP.

Es pertinente mencionar que para los usuarios del DSP, la velocidad y el tamaño de la memoria son factores limitantes. Por otra parte el uso de convertidores analógico a digital y digital a analógico, es frecuentemente causa de introducción de ruido al sistema. Los Procesadores Digitales de Señales de propósito general, han alcanzado los límites de velocidad del diseño de sistemas VLSI, distinguiéndose estos procesadores por el conjunto de instrucciones simples con un tiempo de ejecución de un ciclo y una arquitectura especial que acelera la aplicación de algoritmos dedicados. Lo mismo se puede decir acerca de otros tipos de procesadores, como los del tipo RISC (Computadoras con Conjunto de Instrucciones Reducidas), y tal como estos dispositivos un DSP es efectivo para soluciones reales que requieren procesamiento en tiempo real [14].

Como resultado de las posibilidades que ofrecen estos circuitos, los DSP's pueden encontrarse con una amplia variedad en sus características. En la tabla 1.1 se muestran las comparaciones en los tiempos de cálculo para un procesamiento sencillo de distintos fabricantes de DSP.

Procesador	Fabricante	Año	Tiempos de ciclo	Tiempo de Cálculo FFT 256 Puntos	Tiempo de Cálculo FFT 1024 Puntos
TMS32010	TI	83	200 ns	4.1 ms	31.0 ms
TMS32020	TI	85		2.4 ms	14.2 ms
TMS320C25	TI	87	125 ns	1.5 ms	7.1 ms
ADSP2100	AD	82	62-100 ns		4.5 ms
NEC7720	NEC	82	250 ns	15.0 ms	83.0 ms
NEC77C25	NEC	89	120 ns	7.5 ms	41.0 ms
ST18930	SGS-Thomson		80 ns		
DSP16A	Western El.	86	25-75 ns	0.59 ms	10.0 ms
ZR34161	Zoran		80 ns	0.63 ms	3.3 ms
DSP56001	Motorola	87	74-97 ns	0.49 ms	1.5-1.9 ms
NEC77220	NEC	88	100-122 ns		

Tabla 1.1

Comparación de los distintos fabricantes de procesadores tipo DSP en tiempo de procesamiento con la FFT en distintos números de puntos.

2.- Introducción al DSP 56000-1.

La familia del DSP 56000-1 de Motorola incluye varios tipos de procesadores digitales de señales con una gran precisión y memoria integrada. El tiempo de ciclo de instrucciones se reduce a 97.5 ns [1], incluyendo todas las operaciones de datos de la ALU, aún cuando se requieran tres busquedas a memoria por medio de una sola instrucción, las cuales son ejecutadas en un ciclo de reloj. Toda la memoria de datos es de 24 bits de longitud, para dar 144 dB de rango dinámico para almacenamiento de señales. La Unidad Aritmética Lógica (ALU) interna, contiene dos acumuladores de 56 bits cada uno, permite un rango dinámico extendido en 33 dB para cálculo iterativo. Los espacios de memoria son divididos entre tres, para poder tener más de 9500 bytes de almacenamiento y en el 56001, 512 palabras de RAM interna de programa.

2.1 Arquitectura del DSP 56000-1.

La arquitectura del DSP56000-1 ha sido concebida para maximizar las aplicaciones de procesamiento de datos, y este objetivo ha sido el resultado de una arquitectura dual expandible con periféricos internos sofisticados y dispositivos generales de entrada-salida (ver figura 2.1).

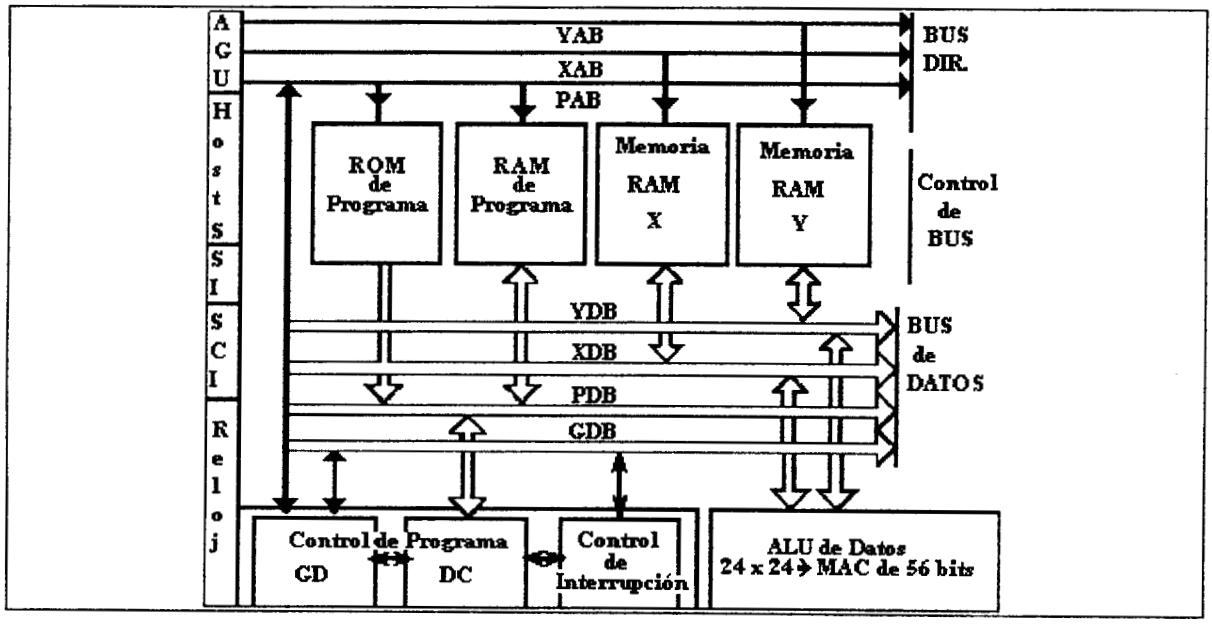


Figura 2.1
Arquitectura del DSP56000-1

Esta dualidad consiste en dos espacios de memoria expandibles e independientes, dos unidades de generación de direcciones (AGU), una unidad aritmética lógica (ALU) de datos con dos acumuladores. Los componentes más comunes dentro del DSP son los siguientes:

- Bus de datos
- Bus de direcciones
- ALU de datos
- AGU
- Memorias de datos X e Y independientes
- Memoria de programa
- Recursos Internos:
  - Expansión de memoria (Puerto A)
  - E/S de propósito general (Puertos B y C)
  - Interfase de comunicación serial (SCI)
  - Interfase de comunicación síncrona (SSI)
  - Interfase Host (ver sección 2.2)

2.1.1 Bus de Datos.

El DSP56000-1 se organiza alrededor de registros de un procesador central compuesto de tres unidades independientes de ejecución. Los buses mueven los datos e instrucciones mientras se ejecutan programas en las unidades de ejecución. El movimiento de datos en el sistema ocurre bajo cuatro buses bidireccionales de 24 bits: el bus X, el Y, el de programa y el bus de datos global.

2.1.2 Bus de Direcciones.

El direccionamiento de 16 bits unidireccional es dado para las dos memorias (X e Y) internas y externas. Para la memoria de programa existe un bus de direcciones específico. Solo una memoria de las tres externas puede ser accesada en un ciclo de instrucción, y no existe pérdida en la velocidad si una de ellas es accesada, pero si dos ó tres espacios de memoria son accesados en una instrucción sencilla, ésta podrá tener uno ó dos ciclos de retraso respectivamente.

2.1.3 ALU de Datos.

Esta ALU ejecuta todas las operaciones aritméticas y lógicas. Consta de cuatro registros de entrada de 24 bits, dos acumuladores de 48 bits, dos registros de extensión del acumulador de 8 bits, un registro de corrimiento, y una unidad de acumulador multiplicador (MAC) sin pipeline. Las operaciones de la ALU pueden ser leídas ó escritas como operandos de 24 ó 48 bits a través de los datos de las localidades en X e Y, siendo capaz de ejecutar cualquier operación en un solo ciclo de instrucción.

2.1.4 Unidad Generadora de Direcciones (AGU).

Todos los cálculos necesarios para efectuar direccionamiento en la memoria se ejecutan por la AGU. Esta unidad opera en paralelo con otros dispositivos internos para obtener una dirección en el espacio de memoria, conteniendo 8 registros de direccionamiento (R0-R7), 8 registros de desplazamiento (N0-N7), y 8 registros modificadores (M0-M7). Rn es un registro de 16 bits que puede contener una dirección o un dato, cada contenido puede ser una salida a cada uno de los 65535 espacios de direcciones de X, Y ó P. Mn y Nn son registros de 16 bits regularmente son usados para actualizar o modificar a Rn, también pueden ser utilizados para almacenar datos de 16 bits.

2.1.5 Memoria de Datos X e Y independientes.

Las memorias RAM X ó Y , internamente son memorias del tipo estático de 24 bits ocupando 256 localidades del espacio de memoria. Los periféricos ocupan 64 localidades en el espacio de memoria, pudiendo hacerse una expansión de cada una de ellas en su propio espacio hasta un total de 65535 bits de direcciones.

2.1.6 Memoria de Programa.

El programa de memoria interno consta de 3750 palabras por 24 bits en ROM para el DSP 56000, o palabras de 512 por 24 bits en RAM para el DSP 56001, pudiendo ser expandida la memoria de programa hasta un total de 65535 localidades.

2.1.7 Periféricos Internos.

Algunos sistemas VLSI (incluyendo a los DSP), para hacer más simple su aplicación, han incluido dentro de su diseño de arquitectura a los periféricos. En el caso del DSP se cuenta con un puerto Host de 8 bits, dos interfases seriales y líneas de entrada/salida de propósito general. Una de las dos interfases seriales, es la Interfase Serial Síncrona (SSI) para conexiones con otro DSP u otro microprocesador. La otra interfase serial es la Interfase de Comunicación Serial (SCI), para sofisticadas interfases seriales en red, microprocesadores, MODEM y otros dispositivos seriales.

Cada dispositivo interno puede ser encendido o apagado, y si éste es apagado, entonces sus líneas externas pueden ser usadas como líneas de entrada de propósito general (figura 2.2).

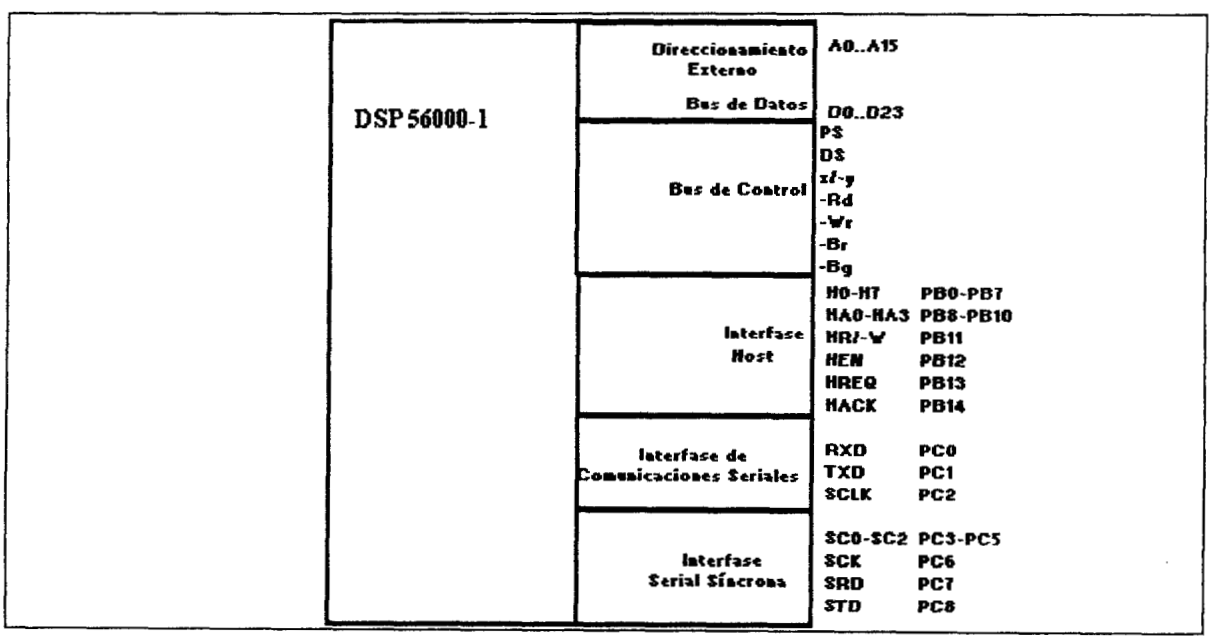


Figura 2.2.

Líneas de E/S de propósito general del DSP56000-1.

2.2 Organización del Modo Host.

La interfase Host es una conexión directa al bus de datos de un procesador Huésped y éste puede ser cualquier microcomputadora estándar, otro DSP o circuitería DMA, ya que esta interfase puede ser vista como parte de una memoria. A su vez esta interfase es asíncrona y consiste de dos registros, uno accesible para el procesador Host y otro accesible al DSP (Ver figura 2.3).

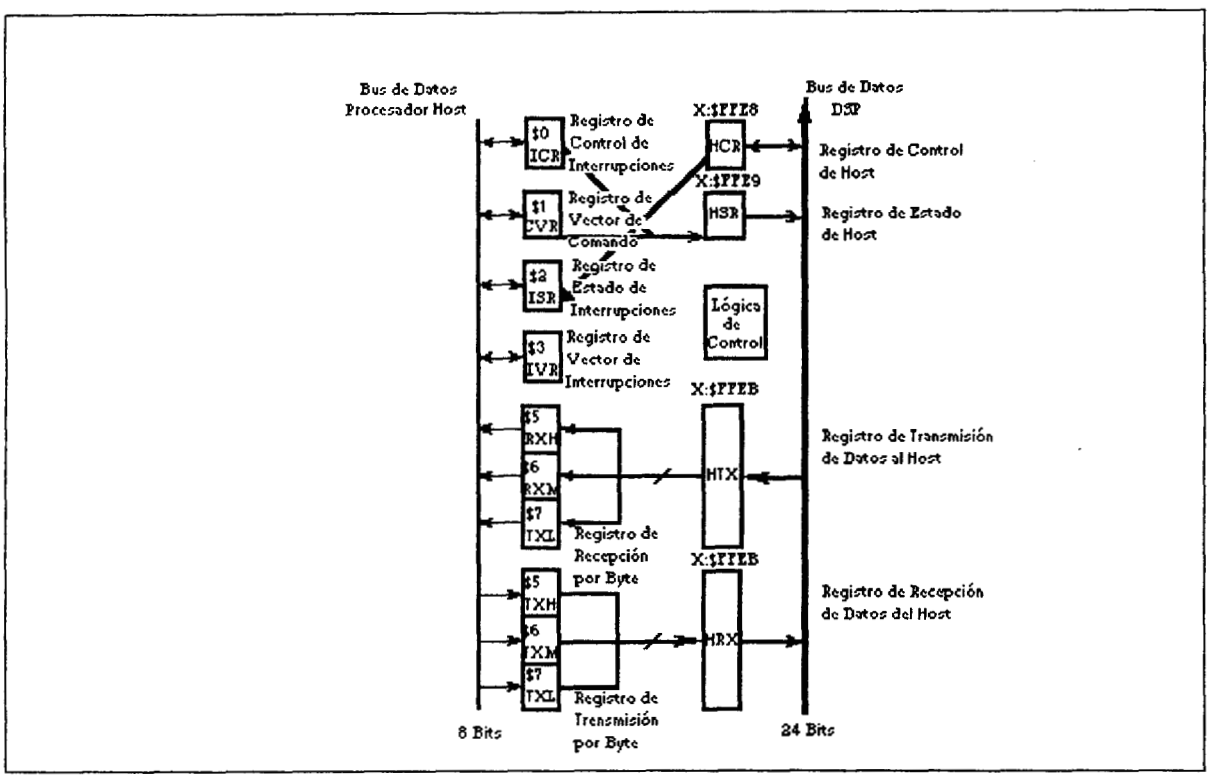


Figura 2.3
Comunicación del puerto Host

EL DSP visualiza a la interfase Host como un periférico mapeado a memoria ocupando tres palabras de 24 bits en un espacio de memoria de datos. Este puede usar a la interfase como un periférico usando técnicas de programación por interrupciones o muestreo y en forma separada transmiten y reciben datos a alta velocidad entre el DSP y la interfase Host eficientemente.

Por otro lado la interfase Host es mapeada en sus registros internos al procesador Host como un periférico a memoria ocupando 8 bytes en su espacio de direcciones (Ver figura 2.4), donde estos registros pueden ser vistos como

registros de control (ICR), registro de estados (ISR) y registros de vectores (IVR,CVR) siendo CVR un registro de comando especial que es usado por el procesador Host para enviarle instrucciones al DSP, y solo accesado por el mismo procesador y no por el DSP. Cada uno de estos registros tiene una función en especial, que el programador puede disponer de ellos a voluntad, para el manejo de control bidireccional de transferencias "HandShaking", entre el DSP-Host, y el Host-DSP. Utilizando los correspondientes bits (capítulo 4) del lado del Host (figura 2.4a) denominados HF0 y HF1, el primero tiene la característica de informar al DSP cuando se concluye la transferencia de un programa después de un Reset en el modo de special Boostrap.

$00

INIT	HM1	HM0	HF1	HF0	0	TREQ	RREQ

⇓	⇓	
0	0	Modo de interrupción (DMA apagado)
0	1	Modo DMA 24 bits
1	0	Modo DMA 16 bits
1	1	Modo DMA 8 bits

Figura 2.4a
Registro de Control de Interrupciones (ICR)

$01

HC	0	0	Vector Host

Figura 2.4b
Registro de Vector de Comando (CVR)

Por otra parte en la figura 2.4c, los denominados bit HF3 y HF2, se pueden utilizar por el DSP (figura 2.3) para informarle al Host acerca de un proceso, en especial este punto se detalla en el capítulo 4. El acceso a estos registros por parte del Host es a través de instrucciones estándar (lenguajes de alto nivel, ensamblador, etc) y cualquier modo de direccionamiento para comunicarse con estos registros[1,22,27].

$02

HREQ	DMA	0	HF3	HF2	TRDY	TXDE	RXDF

Figura 2.4c
Registro de Estados de Interrupciones (ISR)

$03

Número de vector de Interrupción

Figura 2.4d
Registro de Vector de Interrupciones (IVR)

31 $4	24,23 $5	16,15 $6	8,7 $7	0
00000000	RXH	RXM	RXL	
sin uso	TXH	TXM	TXL	
7	0,7	0,7	0,7	0

Figura 2.4e
Registros de Transmisión / Recepción de 24 bits

Los DSP56000-1 son optimizados para ejecutar algoritmos en la menor cantidad de operaciones que sean posibles mientras se mantiene un alto grado de precisión. Estos diseños dan como resultado una naturaleza dual, y una arquitectura expandible con sofisticados periféricos internos en el circuito integrado y en las entradas/salidas de propósito general.

3.-Desarrollo de una Arquitectura para PC.

El desarrollo del sistema DSP-56000-1 esta basado en la técnica de wire-wrap sobre una tarjeta de expansión de propósito general para conectarse dentro de una ranura de expansión de una computadora de tipo personal. Los componentes utilizados son listados a continuación:

- Lógica de control del procesador Host y DSP (Basado en compuertas TTL).
- DSP56001RC20, con encapsulado cerámico de 88 pins tipo PGA..
- Oscilador a 19 Mhz.
- Memoria tipo S-RAM de 32k x 24 para el DSP (a una velocidad de 150 ns).
- Buffer de tres estados, para interfase entre DSP y PC.

3.1 Mapeo de Memoria

En una computadora personal, los puertos son direccionables para uso de entrada/salida en el espacio de direcciones de la PC, donde los datos pueden ser enviados a ésta por medio de instrucciones de "salida", o ser leídos por medio de instrucciones de "entrada" [32]. Dichas instrucciones pueden escribirse en cualquier tipo de lenguaje de programación para la PC. El mapa de puertos de la PC esta dividido en dos partes: la primera es el espacio de direcciones de $0000 a $01FF, en la que reside la base del sistema. La segunda parte de este espacio es de la dirección $0200 a la $03FF, el cual es usado como puerto de direcciones en el sistema de bus y que es disponible en la ranura de expansión de una computadora del tipo personal [32].

Un esquema para la activación de distintas selecciones de circuitos, esta basado en el diagrama mostrado en la figura 3.1, sabiendo que para que la PC pueda considerarse como puerto de entrada/salida [3], se establece que ésta este a partir de la dirección 300 Hex, definida por la tabla de verdad 3.1.

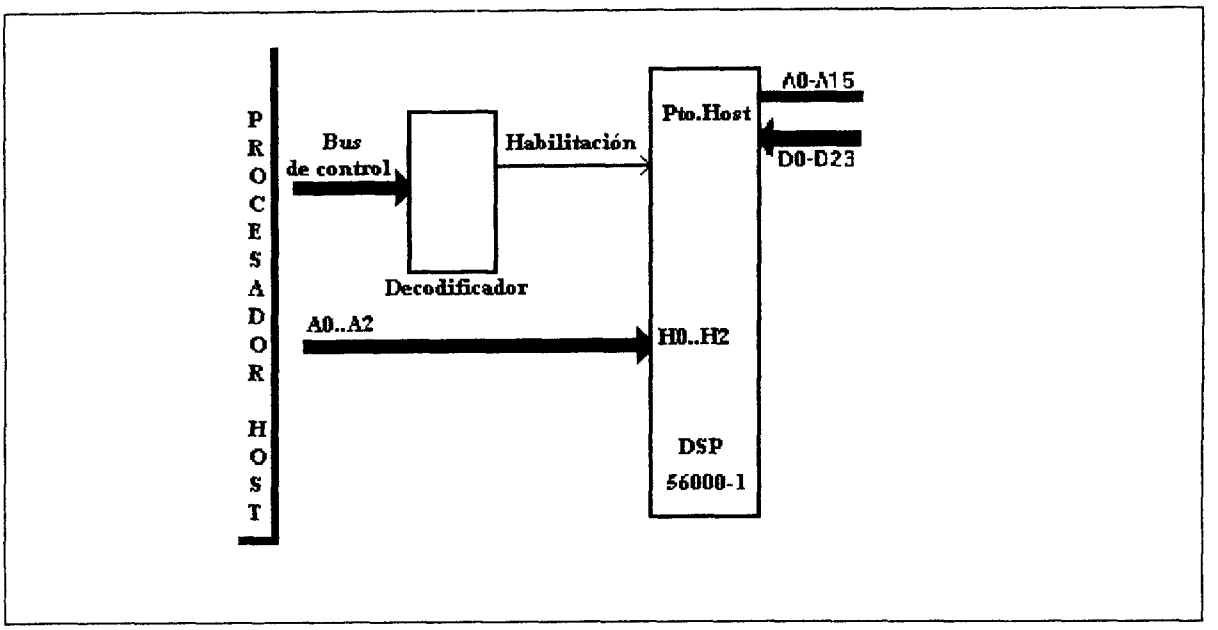


Figura 3.1
Esquema de mapeo del DSP en el puerto Host

Bus de Direcciones de la P.C.

A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
0	1	1	0	0	0	0	0	X	X	X

valores hexadecimales

300
301
302
303
304
305
306
307

tabla 1.1

Los últimos bits (A00 - A02) representan el direccionamiento para los distintos canales utilizados en el manejo de un sistema periférico externo. En este caso a las señales de direccionamiento H0-H2 de la interfase host (capítulo 2), y la

señal A3 en conjunto con la señal proveniente de los distintos "chip enables" y por medio de unas compuertas tipo "OR", se genera la señal de "Hen" que es la habilitación hacia el DSP, tal como se observa en la figura 3.2.

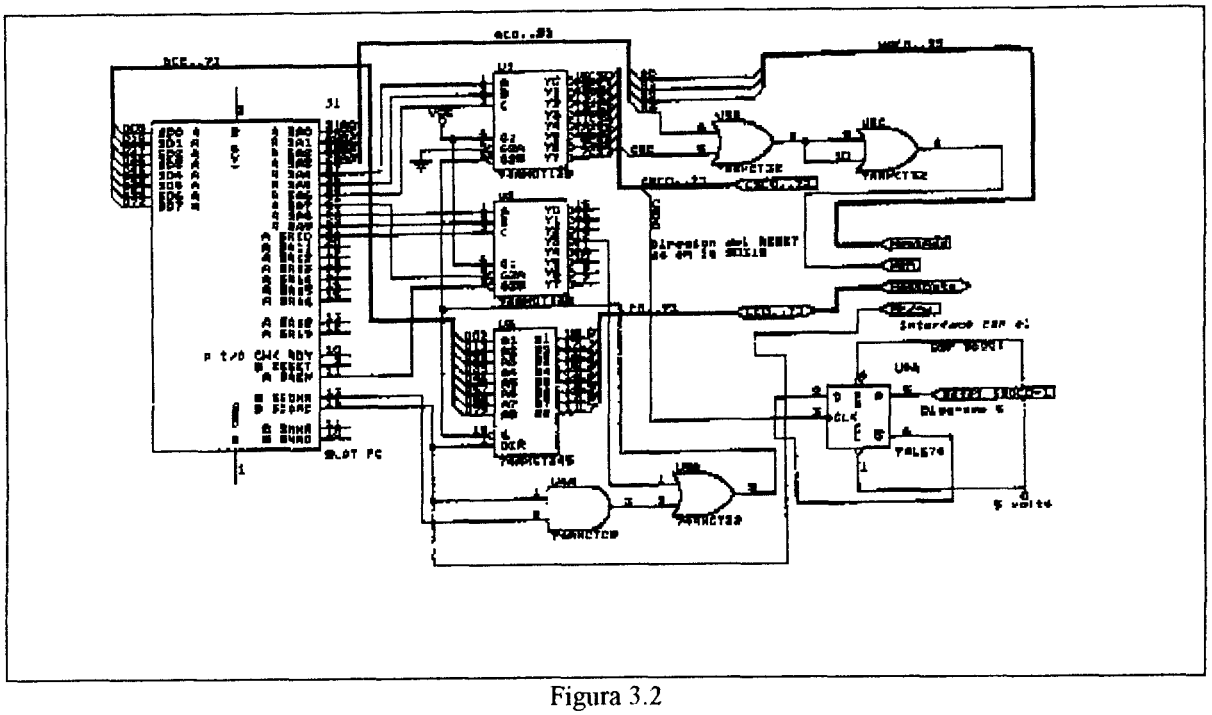


Figura 3.2
Inteface DSP-Host, Host-DSP

A su vez, y por medio de estas compuertas se genera un retraso (doble compuerta "OR") , con el objetivo de permitir la sincronización en el ciclo de escritura de 24 bits al DSP, que es permitido dentro de los límites que especifica el fabricante, como se muestra en la figura 3.3. El círculo indica el tiempo máximo para poder escribir correctamente hacia el DSP con una variación de retraso aproximadamente de 1 nseg [1,21] con respecto a la señal del "Hen". El no respetar este tiempo máximo ocasiona error al escribir a la interface. Un ejemplo de escritura es cargar el programa que proviene de la PC.

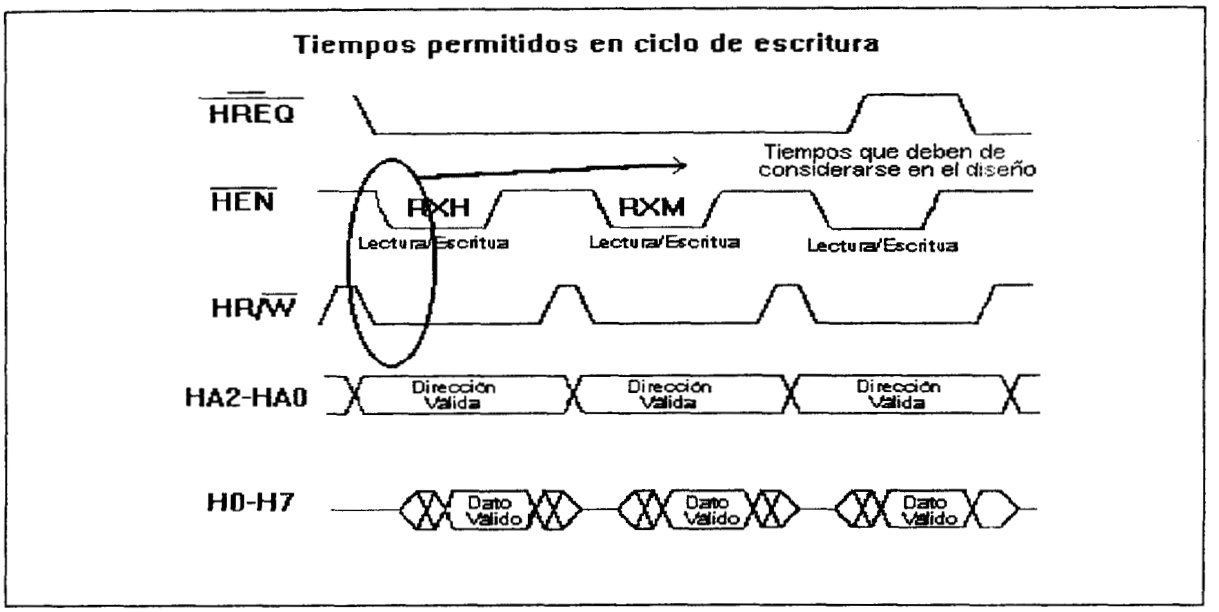


figura 3.3
Ciclo de escritura de 24 bits.

También, tal como se observa en los ciclos de tiempo de las figuras 3.4 y 3.5, es necesario desarrollar la lógica para hacer compatibles las secuencias. Las señales de control de habilitación (Saen PC), lectura (Rd PC) y escritura (Wr PC), son modificadas para el control de un bus bidireccional de datos de 8 bits en la que la señal de Rd (PC) que es utilizada para la dirección del mismo. A su vez esta misma línea junto con la línea de Wr (P) llega a una compuerta del tipo AND, y su salida junto con la línea que es la salida de un decodificador del tipo 3 a 8 (pin 12 de un 74138-U1) y que es habilitado por la línea de Saen y un bit del bus de direcciones de la PC (A7), entra a una compuerta del tipo "OR", y la salida de este genera la activación al bus bidireccional. Al mismo tiempo activa a un decodificador del mismo tipo (74138-U2), para la generación de los distintas líneas de selección, en las que solo se utilizan dos: una para "Hen" que corresponde al mapeo descrito en la tabla 3.1, y la segunda para la activación de un Flip-Flop tipo D que constituye el RESET del DSP, que corresponde a la dirección de mapeo $318, donde su habilitación esta basada en el control descrito anteriormente.

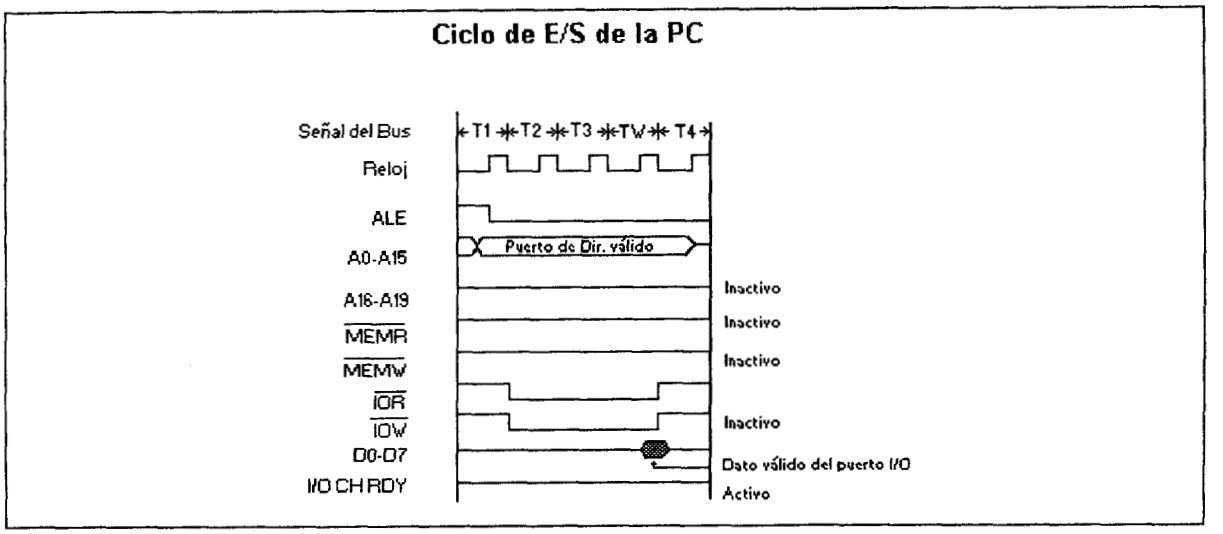


figura 3.4

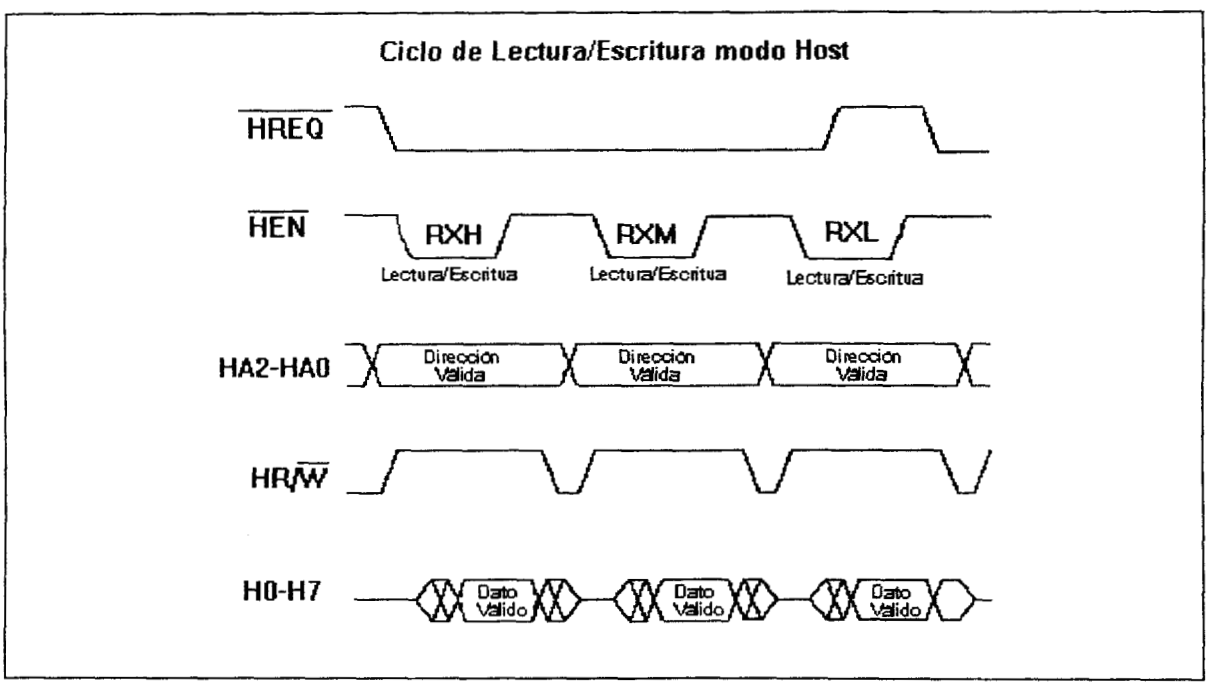


figura 3.5

3.2 Arquitectura del sistema.

Como el sistema DSP56000-1 tiene distintas formas de operación hacia el CPU (tabla 3.2). El modo de operación del sistema seleccionado es del tipo 'Special Boostrap', este modo permite la carga de programas a la memoria RAM interna

de programa. Estos pueden provenir de una EPROM ó de la interfase HOST. De esta última se realiza la transferencia de programas al DSP (figura 3.6).

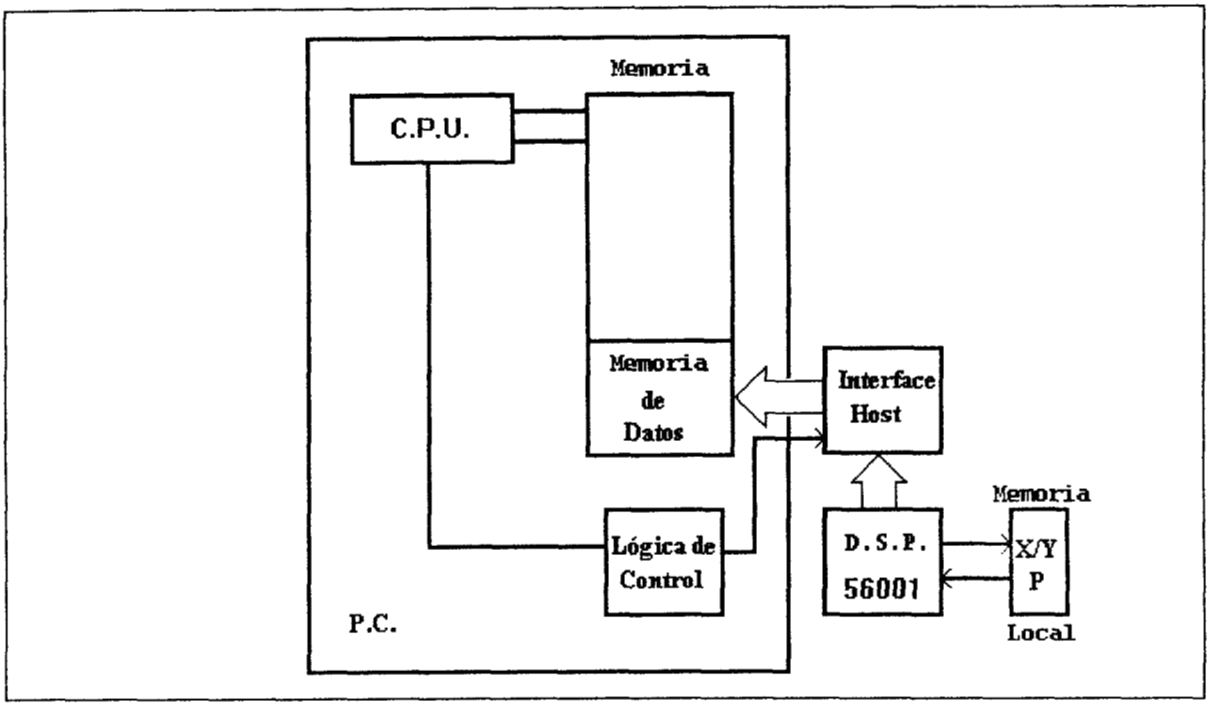


Figura 3.6
Diagrama a bloques del sistema

En este modo, después de cargar la memoria interna, el DSP cambia su operación al modo sencillo (modo 0) y empieza a ejecutar el programa. La forma de habilitar al modo 1 es cuando el DSP recibe un estado de RESET, anteriormente descrito, además de que por software y controlado por el procesador HOST (con instrucciones sencillas de entrada/salida al Puerto[$318]) se proporciona el tiempo suficiente para que el DSP realice el muestreo de las entradas de MODB y MODA (estos deben de ser cero y uno respectivamente), y el DSP identifique el modo de operación (tabla 3.2).

MB	MA	Modo de operación del circuito
0	0	Circuito sencillo, no expandido.
0	1	Boostrap
1	0	Expandido normal
1	1	Expandido para desarrollo

Tabla 3.2

Una vez identificado el modo (Special Boostrap) ocurren las siguientes acciones antes de inhabilitar al RESET:

1. La lógica de control mapea a la ROM de Boostrap en el espacio de memoria interna de programa del DSP en la localidad $0000.
2. La lógica de control causa que el programa de la ROM de Boostrap sea leído (solo las direcciones de los bits menos significativos) y toda la escritura va hacia la RAM de programa (todos los bits son utilizados). Esta condición permite al programa de Boostrap cargar al programa del usuario en las direcciones de $0000-$01FF.
3. La ejecución del programa empieza en la localidad $0000 de la ROM de Boostrap, y permitirá cargar a la RAM de programa a través de la interfase HOST, siendo el único requerimiento que el bit 23 del bus de datos externo este en cero.
4. Para finalizar la operación de Boostrap y empezar el programa de usuario, la ROM de Boostrap primero pone el modo 0 escribiendo al registro OMR (esto es para remover a la ROM de Boostrap de la memoria de programa y rehabilitar al acceso de lectura/escritura a la RAM de programa) y por último, el cambio de modo permite que el programa de Boostrap limpie el registro de estados, y entonces ejecute una instrucción de brinco a la dirección $00 para ejecutar el programa del usuario.

En la figura 3.7 se puede observar que el diseño para el sistema es como fue descrito en la figura 3.6. Por una parte se observa que la conexión para IRQA e IRQB que son las respectivas señales de entrada de MODA y MODB, y como fue descrito en el modo de operación, la primera fue conectada a una resistencia de 15 kohm y un punto a tierra. La segunda hace lo mismo, pero a través de un diodo se conecta a la señal que viene del RESET, debido a que la activación de esta señal es por medio de un nivel bajo, dirigido hacia la entrada

de MODB, manteniendo la duración mientras este activo el RESET, para que se pueda permitir el proceso de rastreo de las líneas MODA y MODB descrito anteriormente, para que este último regrese a un nivel alto y así evitar que se genere su otro modo de operación, es decir actuar como interrupción, que en este caso no se utilizan.

Por otra parte se observan otras conexiones del DSP. Una de ellas, es la del reloj de 20 Mhz, conectada a las líneas de EXTAL y XTAL, debido a que el sistema desarrollado es un DSP56001 con terminación RC20, (20 significa el tiempo máximo de soporte).
Las líneas de escritura, lectura, DS y PS, serán caracterizadas en la siguiente sección.

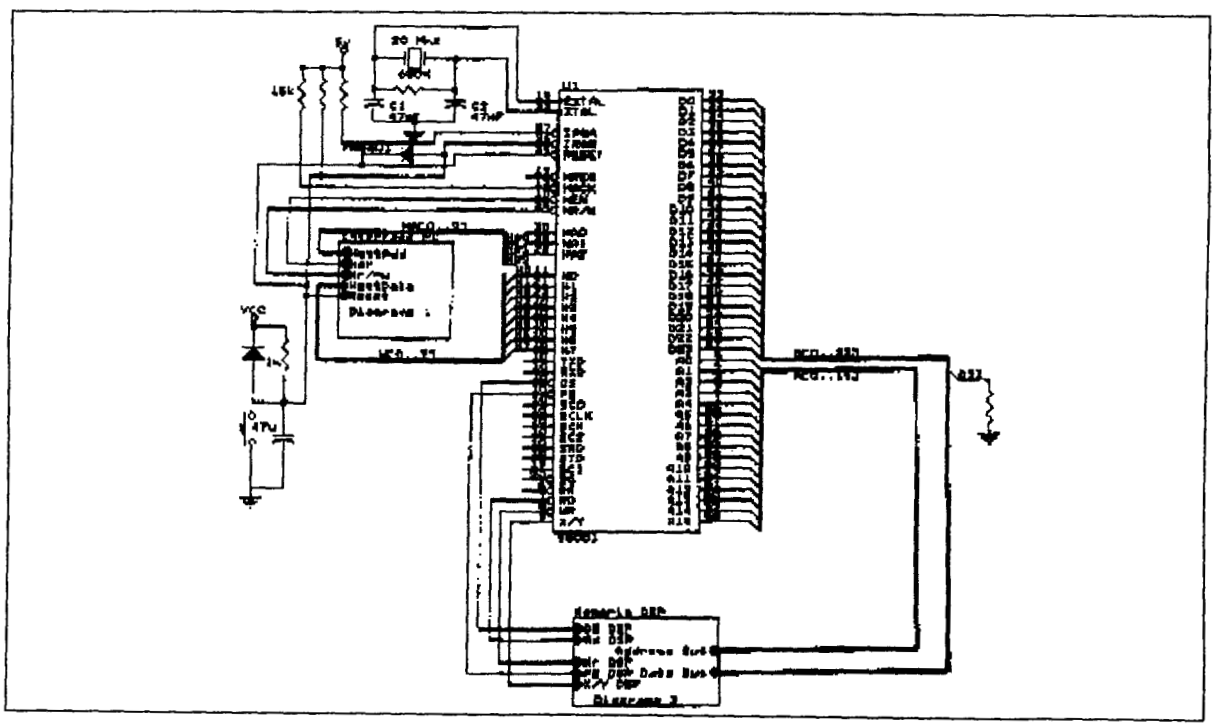


Figura 3.7
DSP 56000-1 modo spccial Boostrap.

3.3 Organización de la Memoria X,Y y P.

La organización de la memoria local, se basa en memorias que responden a 150 nseg en el proceso de escritura/lectura. Se considera que es un tiempo suficiente para que un sistema de 20 Mhz pueda tener acceso y así permitir el proceso pipeline interno (ver apéndice B). Esta organización esta basada en el control para que el DSP pueda tener una partición de sus tres bancos de memorias con las que cuenta (ver capítulo 2). Como se muestra en la figura 3.8 se cuenta adicionalmente con la memoria RAM de datos, que es del tipo de X e Y, y todas cuentan con una capacidad de 32 kbytes.

Esta organización de memoria es hecha a partir de la dirección $0000 (figura 3.10). En los tres casos la partición tiene la restricción de no ocupar las direcciones altas debido a que en ella se encuentra mapeado el control del sistema, por ejemplo: para manejo de periféricos internos (SCI, SSI, Control de Host, etc) se utiliza la parte alta de la memoria en X ($FFC0-$FFFF), para el control de periféricos externos (mapeo de memoria) se usan las mismas direcciones pero en la región de memoria Y, y para el manejo de interrupciones, estados de excepción, comandos de Host etc, se utilizan las regiones de memoria de programa parte baja ($0000-$003F).

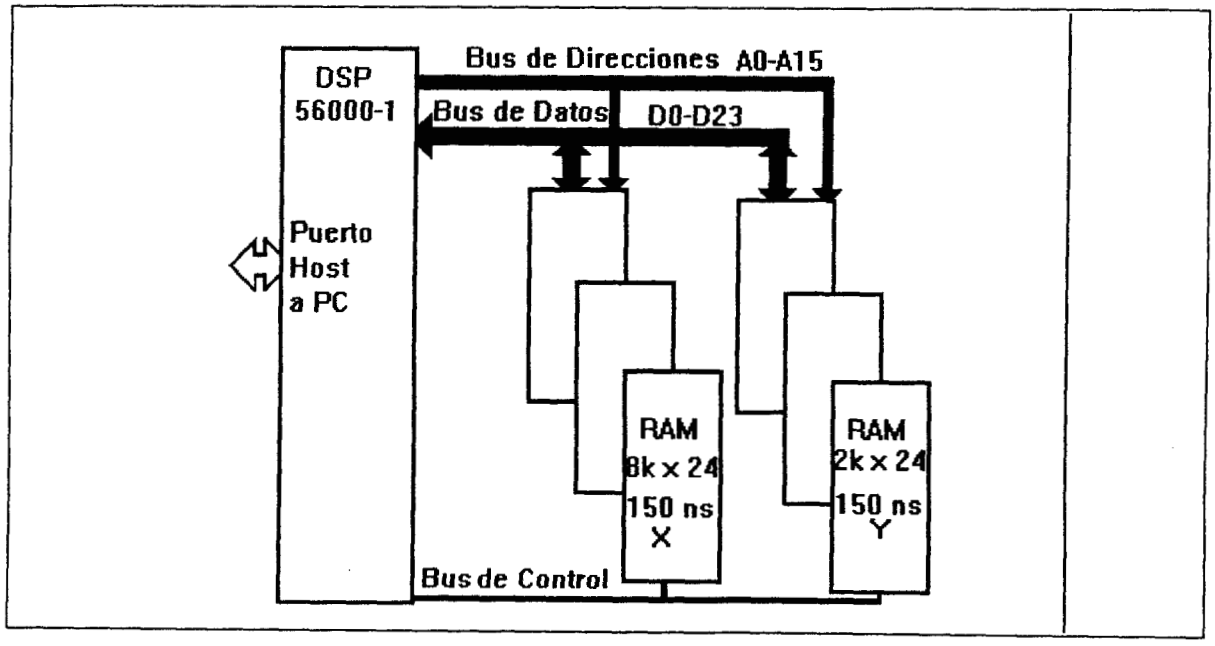


Figura 3.8
Organización de la Memoria del DSP

La memoria del sistema en X e Y interna se encuentra a partir de la dirección $0000-$00FF, mientras que este entre la dirección $0100-$80FF, que son las dimensiones de la memoria tipo 62256, por lo que fue necesario emplear un decodificador sencillo utilizando las líneas de direccionamiento alto, tal como se ve en los diagramas de tiempo de acceso a memoria externa (figura 3.9). También se utilizaron las señales de control DS y X/Y que son las habilitaciones para las líneas CE de la memorias RAM; muestran que las señales del OE y WR son manejadas por las líneas de control Rd y Wr respectivamente.

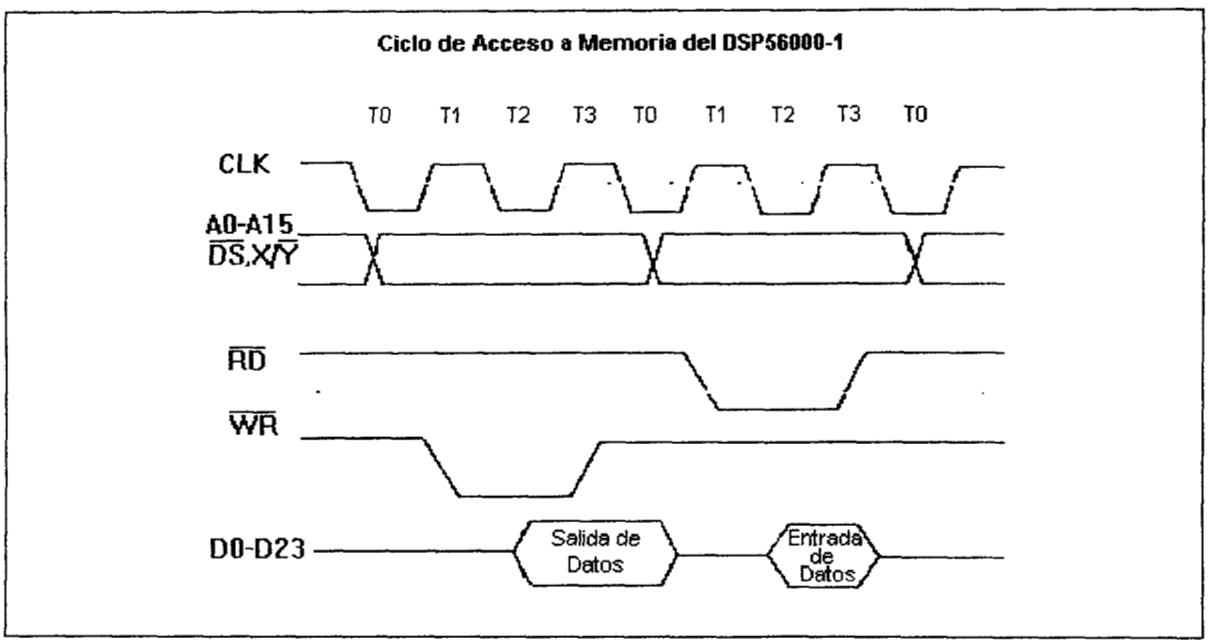


Figura 3.9
Ciclo dc acceso a memoria dc datos X/Y

En la memoria de programa, se tiene de la dirección $000 a la $1FF, por lo que no fue necesario emplear una expansión ni el uso de la señal de control de PS. Esto representa una restricción para el programador, ya que debe de realizar solo programas que no excedan ese tamaño de capacidad.

Con este desarrollo de la circuitería, el siguiente paso es el de desarrollar un programa (lenguaje de alto nivel) para valoración. El programador tiene las siguientes restricciones :

- Memoria interna en X e Y de $0000-$01FF.
- Memoria externa en X e Y de $0200-$80FF.
- Memoria de programa de $0000-$01FF (interna).
- Mapeo de memoria en la PC de los registros...
    - ICR $300
    - CVR $301
    - ISR $302
    - IVR $303
    - sin uso $304
    - RXH y TXH $305
    - RXM y TXM $306
    - RXL y TXL $307
    - RESET DSP $318

Los últimos tres registros en el DSP pertenecen al espacio de direcciones del DSP en $FFEB. El registro ICR es el HSR del DSP en la dirección $FFE8, y el registro ISR es el HCR del mismo en la dirección $FFE9.

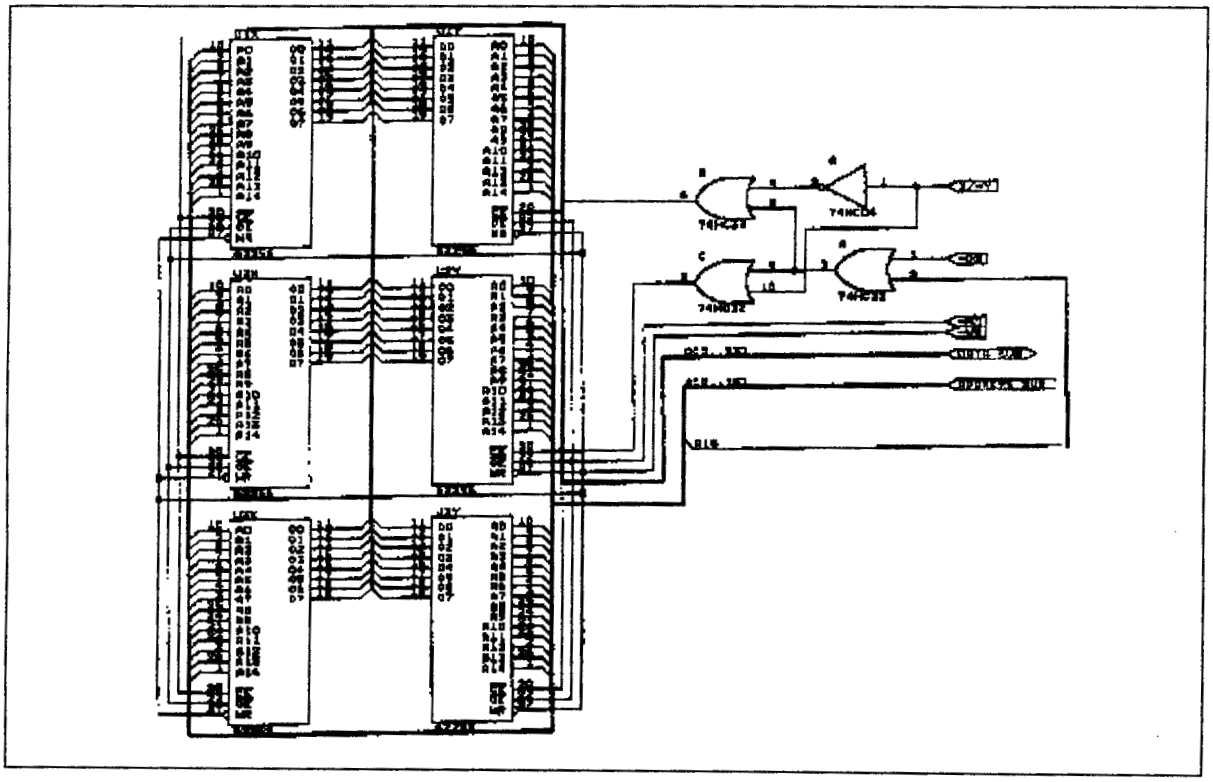


figura 3.10
Expansión de memoria de datos X/Y del DSP

4.- Desarrollo del Software para PC y DSP56000-1.

Una parte fundamental para el desarrollo del sistema es el software, dividido en dos programas, uno para la PC (procesador Host) y el otro para programas de aplicación. El primero, desarrollado en el lenguaje "Pascal", da la inicialización a la interfase Host del DSP. En un inicio se pide el nombre del archivo del programa a transferir en código del DSP (Nombre.lod), junto con los datos necesarios para su transferencia, ya que este programa del DSP permite además de recibir el código ensamblador, identificar el número de bloques y el tamaño de cada una de las memorias, tanto de programa como de datos X e Y. A su vez este programa contiene tres procedimientos principales:

1. Verificación de la Interfase.
2. Carga a las Memorias de datos X e Y.
3. Carga de instrucciones a la Memoria RAM de Programa.

En el segundo y tercer punto se lleva a cabo la transferencia de un programa en ensamblador DSP56000-1, y siendo el punto dos dedicado solamente a cargar cada una de la memorias de datos X e Y, mientras que en el tres se carga a una aplicación en especial. En todos los puntos anteriores el programa ejecuta las siguientes funciones:

- Inicialización.
- Transferencia HOST-DSP Paralela de 24 bits.
- Transferencia DSP-HOST Paralela de 24 bits.

4.1 Inicialización.

La inicialización del sistema DSP es por medio de un RESET, el cual consiste en escribir cualquier vector al puerto $318 (Port[Reset_On]) para activar la línea, y volver a escribir para desactivarla (Port[Reset_Off]), de tal manera que se mantenga activa durante un tiempo, que para este caso es de 1 segundo.

La inicialización del modo Host debe realizarse en dos etapas:

Primero: Hay que inicializar la sección correspondiente del lado del DSP. Es necesario seleccionar las interrupciones y los bits de señalización (banderas de estados). Para nuestro propósito, donde no se maneja el tipo DMA, solo basta aplicar un RESET al DSP56000-1 (Procedimiento 1: Reset al DSP), ya que de

esta forma se pone a los registros del lado del HOST en el estado inicial que se muestra en la tabla 4.1. Una vez hecho esto, se espera un tiempo de retraso calculado de 1 segundo, antes de dar inicio a que el DSP ejecute el protocolo descrito en la sección 3.2., para luego escribir en esa misma dirección y desactivar al RESET. Una vez fuera del estado de RESET, el DSP está en espera de la transferencia de instrucciones por parte del procesador HOST, que se realiza mediante la transferencia de datos, descrita en "Transferencia DSP-HOST, HOST-DSP paralela de 24 bits". Una vez realizada la transferencia del programa inicial, el sistema espera a que en el registro ICR (dirección $300), sea puesto en 1 el bit correspondiente al HF0 (procedimiento 2: Termina), el cual indica al DSP que la transferencia de programas ha terminado, para así realizar un reinicio a la dirección $00, y empezar a ejecutar el programa transferido.

procedimiento 1
Reset al DSP

```
Procedure HP_RESET(splitmem:boolean);

begin
   Port[RESET_ON]:=0; {reset on}
   delay(50); {Pon un retraso para confirmación de modo de operación}
   Port[RESET_OFF:=0; {reset off}
  delay(50);
End
```

Valores de Vectores después del RESET

ICR $300	=	$00
CVR $301	=	$12
ISR $302	=	$0F
IVR $303	=	$00
sin uso $304	=	$xx
RXH y TXH $305	=	$xx
RXM y TXM $306	=	$xx
RXL y TXL $307	=	$xx

TABLA 4.1

Una vez que lo anterior se ha llevado a cabo, el programa organiza el manejo de los puertos (registros ICR, CVR, ISR, e IVR) para su propio uso, es decir transfiriendo la rutina principal (programa de desarrollo de procesamiento) a la memoria RAM de programa.

Procedimiento 2
Finaliza transferencia de datos al DSP (HF0=1)

```
{Interrumpe la recepcion de datos en el DSP}
Procedure Termine;
Var
  ICR : byte;
Begin
    ICR:=Port[PC56_ICR];
    ICR:=ICR or $8; {bit 3 (HF0) miss}
    Port[PC56_ICR]:=ICR;
end
```

4.2 Verificación de Interfase

Para la verificación del sistema en la parte del software, se hace la "Inicialización" del sistema (Procedimiento 3, Verificación de Interface). Así los registros del lado del HOST se ponen en estado de inicio (ver tabla 4.1), y se encuentran listos para recibir los parámetros de programación (capítulo 2). Aquí se utiliza esta característica para saber si el sistema trabaja correctamente.

Procedimiento 3
Verificación de interface.

```
procedure HP_TEST;
Begin
    Port[PC56_IVR] := $AA;  {envía un byte al puerto de 16 bits y lo recupera}
    if (Port[PC56_IVR] and $FF) <> $AA then {son iguales ?}
    begin
        writeln(' No existe enlace con el DSP 56000 en dirección ',hexval(PC56_ICR));
        halt;
        end
    else
    writeln('La dirección del DSP 56000 es: ',hexval(PC56_ICR));
    end;
```

Utilizando el registro CVR ($301) se manda a escribir un vector conocido de 8 bits, y entonces se espera que el dato leído en este registro sea el mismo al que se mandó, y de ser así, se continúa con el proceso de carga de datos a las memorias X e Y.

4.2 Transferencia HOST-DSP Paralela de 24 bits

El programa de la PC verifica los siguientes bits antes de enviar los datos de programa (figura 4.1, Procedimiento 4: Envio de 24 bits al DSP por parte del HOST):

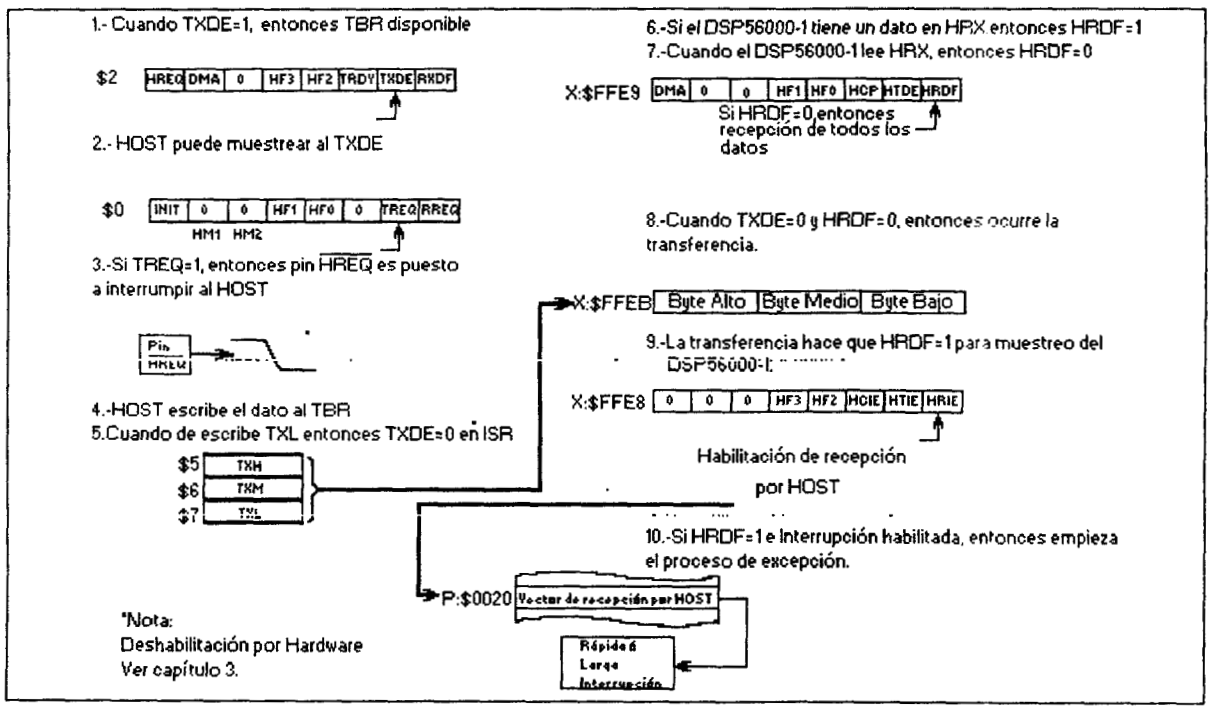


figura 4.1
Transferencia Host-DSP de 24 bits

1. En el puerto del registro ISR del DSP se verifica el bit denominado TXDE que indica que el puerto se encuentra disponible para recibir los datos (bit = 1). Si no lo está, el programa tiene una rutina de espera con un contador que hace que se tenga un límite máximo de tiempo.
2. Una vez hecho esto, el procesador Host envía datos de 24 bits divididos en segmentos de 8 bits al escribir en los correspondientes puertos de TXH, TXM y TXL. Esto se hace enviando el dato de 24 bits al primer registro,

pero antes, dividiendo al dato por el valor decimal 65536 para que tome una longitud de palabra de 8 bits. Al siguiente dato hace el mismo proceso pero se envía al segundo registro (TXM) y se divide a éste entre 256 para obtener una longitud de 8 bits. El último dato pasa sin dividir ya que este termina al finalizar en 8 bits (ver siguiente ejemplo).

Dato = dato a transmitir al DSP de 24 bits de longitud de palabra.
Dato = Dato / 65536
TXH = [Dato] (8 bits)
Dato = Dato / 256
TXM= [Dato] (8 bits)
Dato = Dato
TXL=[Dato] (8 bits)

Procedimiento 4
Envio de 24 bits al DSP por parte del HOST

```
Procedure HP_Send(data:longint;var error:boolean);

Var
  XMIT_RDY,b : byte;
  count      : longint;

begin
   XMIT_RDY := $2;
   error    := FALSE;
   count    := 0;

While ((port[PC56_ISR] and XMIT_RDY)<> $2) and not error do
    begin
       if (count > timeout) then error:= true;
       inc(count);
    end;
    if not error then
      begin
         b:=data div 65536; port[PC56_HI]:=b;
```

```
        {write(' ',hexval(b));}
        b:=data div 256; port[PC56_MID]:=b;
        {write(' ',hexval(b));}
        b:=data ; port[PC56_LOW]:=b;
        {write(' ',hexval(b));}
    end
  else writeln(' >> Send-Time Out ');
end;
```

4.3 Transferencia DSP-HOST paralela de 24 bits.

Para la recepción por parte de la PC de los 24 bits el DSP utiliza el siguiente procedimiento (Figura 4.2, Procedimiento 5: Recepción de datos del DSP):

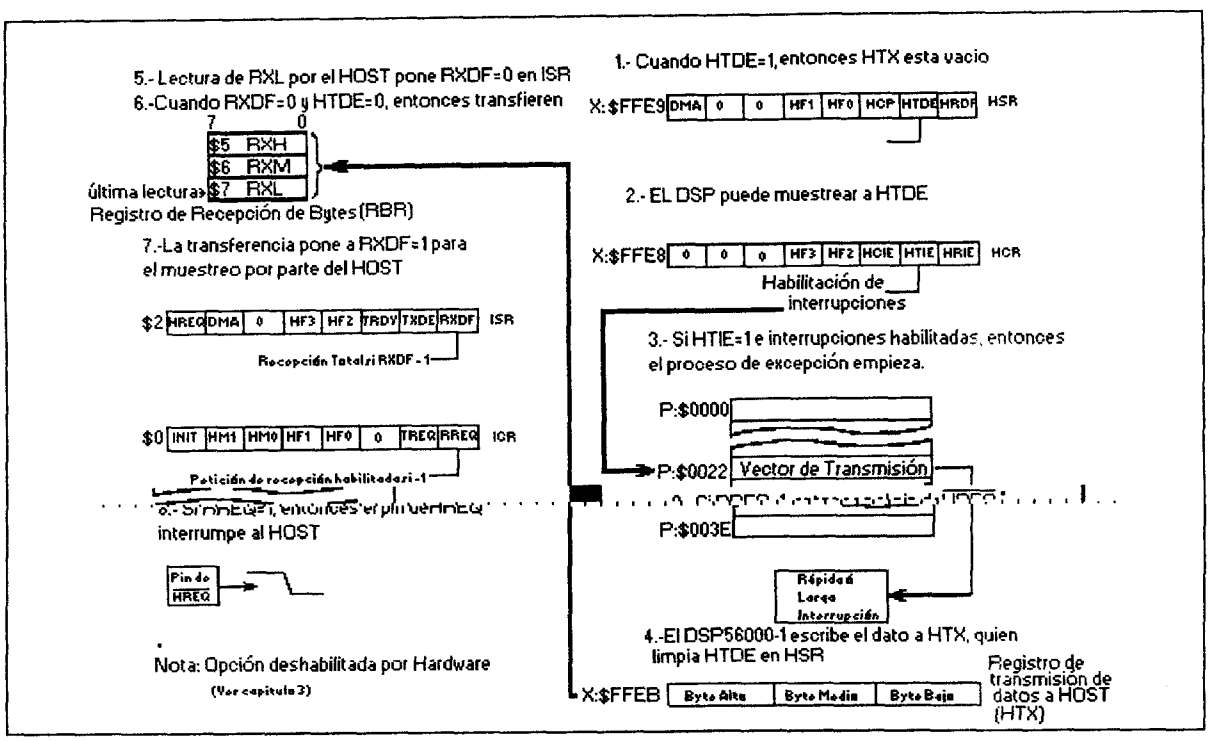


Figura 4.2
Transferencia DSP-Host

1. En el puerto del registro ISR, el DSP verifica al bit denominado RXDF que indica la disponibilidad de recibir los datos (bit = 1). Si no lo está, el programa tiene un contador que hace que se tenga un límite máximo de tiempo de espera.
2. El procesador Host recibe los datos de 24 bits divididos en intervalos de 8 bits leyéndolos uno por uno de los correspondientes puertos de RXH, RXM y RXL recibiendo y almacenando el primer dato de 8 bits en un arreglo. Al siguiente dato hace el mismo proceso pero lo envía a un segundo arreglo, y el último dato pasa a un tercer arreglo, formando la palabra de 24 bits multiplicando el producto del primer arreglo por 65536 y sumandolo al segundo arreglo multiplicado por 256, sumandolo al tercer arreglo (ver siguiente ejemplo, Procedimiento 4: Recepción de Datos del DSP).

Arreglo1 = [RXH] (8 bits)
Arreglo2 = [RXM] (8 bits)
Arreglo3 = [RXL] (8 bits)

dato:=Arreglo1*65536 + Arreglo2*256 +Arreglo3 (24 bits)

Procedimiento 5
Recepción de datos del DSP.

```
{Recibe un longint del puerto B del DSP}
Procedure HP_Receive(var data:longint;var error: boolean);

var
  RCV_RDY : Byte;
  count   : longint;

const
   f : LONGINT=256; {si no de define asi, 256 = 0}

begin
   RCV_RDY := 1; error := false; count := 0;
   {no funciona la bandera de error y lee $FFFFFF como dato}
   while ((Port[PC56_ISR] and RCV_RDY)<>$1) and not error do
      begin
```

```
        if (count > timeout) then error := true;
        inc(count);
    end;
    if not error then
      Begin
        data:=Port[PC56_HI]*65536 + Port[PC56_MID]*f + Port[PC56_LOW];
        Writeln(hexval(data));
      end
    else
      writeln(' >> Receive-Time Out');
end;
```

4.4 Carga de Datos a las Memorias X e Y.

Para este procedimiento, se realiza una "Inicialización" al DSP. Al momento de salir de ese estado, el DSP esta en espera de una "Transferencia Paralela de 24 bits" de un programa en ensamblador del DSP56000-1, que realiza la carga de datos a memoria X e Y. Esta se realiza de la siguiente manera (Procedimiento 6: Transferencia de Datos a las memorias X e Y):

1. El DSP pide el número bloques a la memoria X.
2. Pide el número de datos por bloque en la misma memoria.
3. Se transfieren los primeros datos en la memoria.
4. El software pregunta si se transfirieron todos los datos. De ser así se continúa con el siguiente bloque.
5. Si se transfirieron todos los bloques de la memoria, entonces se repiten los puntos del 1 al 5, ahora para la memoria Y.
6. El programa finaliza la carga, y entonces se continúa con el procedimiento de transferencia del programa principal al DSP.

Procedimiento 6
Transferencia de Datos a las memorias X e Y.

```
Procedure cargapxy(var err:boolean);

const
    ndb = 58;    { DSP_H3.asm}

     px: array[1..ndb] of longint =
(movep  #$0001,x:m_pcddr      ;solo lsb de Port-C puede ser una salida
 movep  #$0000,x:m_pcc        ;todos los pins de Port-C son I/O
 _loop
 bchg   #0,x:m_pcd            ;el lsb de Port-C
 jmp    <_loop *);

Var
  k : integer;

Begin
    HP_Reset(splt);
    for k:= 1 to ndb do
       HP_Send(px[k],err);
       termine;
end;
```

4.5 Carga de instrucciones a la Memoria RAM de Programa.

Como se explicó en los capítulos 2 y 3, la interfase Host es una conexión directa al bus de datos de un procesador Host (siendo en este caso el procesador Host la PC, lo que permite que esta interfase pueda ser vista como parte de la memoria de la misma). En nuestro sistema, se emplea el modo "special boostrap" (capítulo 3). Para poder hacer que el DSP56000-1 identifique el modo de trabajo, es necesario conectar una resistencia entre la salida de la línea 23 del Bus de datos de este mismo y tierra, para que así por medio de esta conexión el hardware pueda reconocer el modo de operación, y después de un RESET generar una búsqueda de instrucciones hacia el puerto Host.

4.6 Transferencia Bidireccional del Modo Host

Para el trabajo del DSP, en el modo Host Command, el programa principal del sistema DSP contiene varios estados de excepción Host (Host Commands) en el que cada uno significa un tipo de procesamiento. En este caso se envía un vector por medio del programa, escribiendo al puerto correspondiente (registro CVR) el número de comando.

Aquí se utiliza la comunicación por "Handshaking". El programa del procesador Host hace uso de los bits que tienen como banderas de señalización en los registros ISR e ICR (Figura 4.3), por lo que el programa del DSP deberá tener acceso a estas mismas banderas por medio de los registros HSR y HCR. Los bits que se traducen como banderas utilizados por el procesador Host como indicadores al DSP son HF0 y HF1 del ICR (bits 3 y 4 respectivamente). Por el DSP los bits empleados para indicaciones al procesador Host son: HF2 y HF3 del HCR (bits 3 y 4 respectivamente).

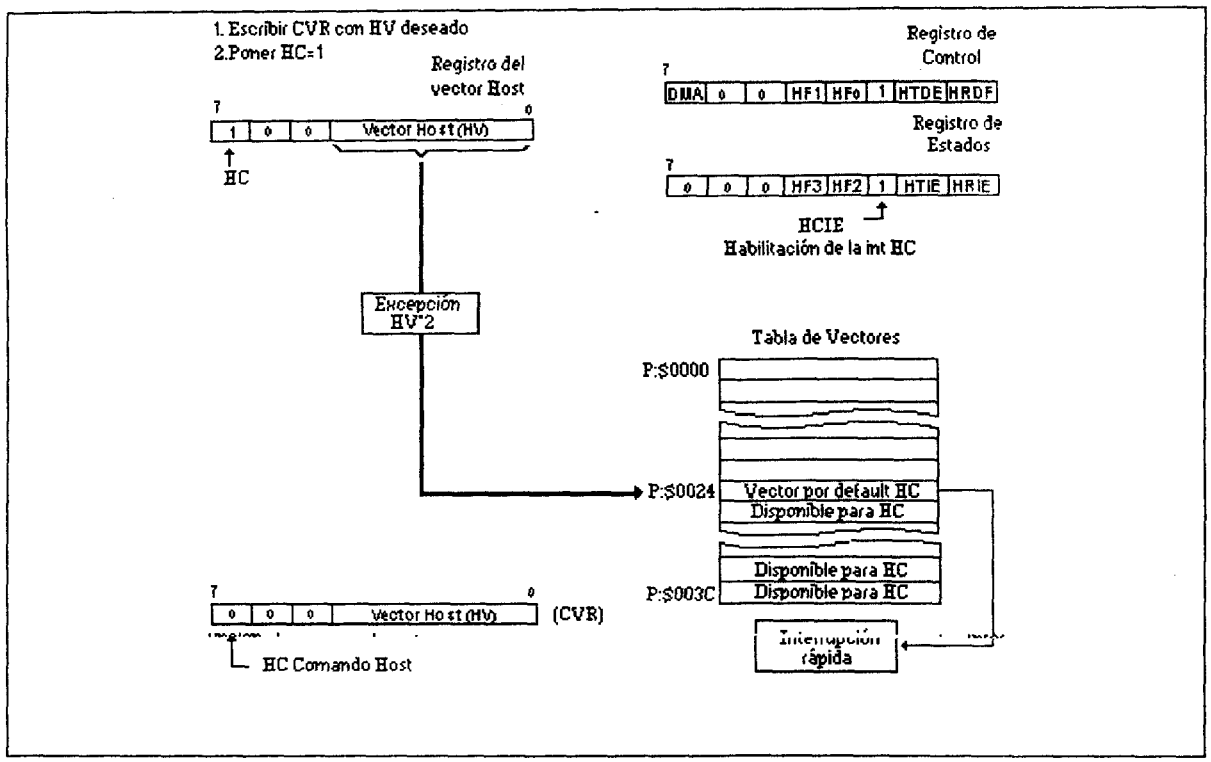


Figura 4.3
Ejecución de un Host Command

Pocedimiento 7
Ejecución de un Host Command.

```
Port[$341]:=$97;  {pon HC = IFFT, para ejecución de una FFT previamente programado}
 while (port[$342] and $10) = $00 do {HF3 = 1, entonces finaliza proceso}
 begin
    outtextxy(-xmin,textos*24,' Procesamiento/ DSP56000-1');
 end;
```

Una vez que ya se cargó el programa en el DSP, e independientemente de la aplicación a realizar (ver las secciones posteriores de este capítulo), el programa del procesador Host y el DSP están habilitados para saber en que momento finaliza un procedimiento, ya que este último tiene la facilidad de modificar las banderas del registro HCR, el procesador Host utiliza el registro ISR (solo lectura). El caso contrario ocurre cuando el DSP, desea saber si el procesador Host esta listo, modificando a través del registro ICR; para el DSP solo se verifica el HSR (solo lectura).

5.- ALGORITIMOS DE PRUEBA, RESULTADOS Y CONCLUSIONES.

5.1 ALGORITMOS DE PRUEBA.

5.1.1 Generación de tablas Senos y Cosenos.

El programa SINCOS.ASM genera tablas de coeficientes para la FFT (Twiddle Factors). Estos coeficientes son de naturaleza compleja, con la parte real: Cosenos, en la memoria de datos X y la parte imaginaria: Senos, en la memoria de datos Y. Cada tabla tiene la mitad de un ciclo de la onda, π radianes ó 180 grados; para una FFT de N puntos son requeridos N/2 datos tanto para las localidades de X como las de Y. Cabe mencionar que las funciones trascendentales del ensamblador DSP56000-1 para el cálculo de éstas, toman valores en rangos que van de -1 a 1. Un redondeo particular es utilizado para minimizar el total de la distorsión armónica de la onda (THD).

Programa1:
generación de coeficientes de tablas de FFT

```
sincos  macro  points,coef
sincos  ident  1,2
;       sincos -      este macro genera los coeficientes de senos y cosenos
;                     para tablas de decimación en tiempo de la FFT
;       points -      número de puntos (2 - 32768, potencia de 2)
;       coef   -      apuntador base de dirección de tabla de senos/cosenos
;                     valor negaivo de cosenos en memoria X
;                     valor negativo de senos en memoria Y
pi      equ    3.141592654
freq    equ    2.0*pi/@cvf(points)

        org    x:coef
count   set    0
        dup    points/2
        dc     -@cos(@cvf(count)*freq)
count   set    count+1
        endm
        org    y:coef
count   set    0
        dup    points/2
        dc     -@sin(@cvf(count)*freq)
count   set    count+1
        endm
        endm   ;fin de macro de senos/cosenos
```

5.1.2 FFT e IFFT.

Para la utilización del algoritmo de Cooley-Tukey de la transformada rápida de Fourier (FFT) de decimación en tiempo (DIT), se empleó la versión radix 2 usando 24 bits. Aquí se utilizan varias técnicas de programación y estrategias de mapeo de memoria para efectuar una FFT, usando los buses de datos internos duales, memorias y la habilidad del control del bus externo para efectuar un acceso sin añadir ningún estado de espera. Esta implementación usa 24 bits en datos almacenados y 24 bits de coeficientes. La ALU (unidad aritmética lógica) mantiene una precisión de 56 bits en sus operaciones. Los buffer externos requieren un número N de localidades en la RAM.

El algoritmo se ejecuta en forma de una macroinstrucción (Macro); busca la tabla de senos y cosenos como coeficientes (twiddle factors) y ejecuta para este caso una FFT de 255 puntos. Utilizando registros de propósito general para el manejo de las variables empleadas en esta rutina. Sin embargo el programa asume que los registros no deben ser alterados por la rutina de la FFT y deben ser salvados por el programa principal.

Todos los datos y coeficientes son complejos, con la parte real almacenada en la memoria de datos X y la parte imaginaria en la memoria de datos Y. Para el caso de N puntos el buffer de datos requerido es de N datos tanto para la memoria X e Y, y por lo tanto también para la ejecución del algoritmo. La entrada de datos es en forma secuencial, y la salida es en forma de direccionamiento inverso, para obtener los datos de salida en el orden correcto.

Como ya se mencionó, este programa usa la tabla de senos y cosenos almacenados en las memorias de datos, para una ejecución a máxima velocidad. La FFT busca la tabla de operación para tener un nuevo valor de senos y cosenos por cada grupo de mariposa (ver generación de tablas de senos y cosenos sección 5.1.1). Para una FFT de N puntos, se requieren de N/2 datos para la memoria X y el mismo número de datos para la memoria Y, para poder llevar a cabo la ejecución de la FFT.

Programa 2
Cálculo de la FFT e IFFT

```
;  ejecuta complex FFT or IFFT
        movem p:N,r0            ; Pone Tamaño N de FFT via r0
        movem p:log2N,r3        ; log2N via r3
        movem p:dataadr,r2      ; Apuntador de arreglo de datos
        movem p:coefadr,m2      ; Apuntador de coeficientes
        move  n3,b              ;Verifica la bandera de IFFT
        tst   b                 ;Bandera de IFFT es cero
        jne   ift               ;Brinca a rutina de IFFT
        jsr     fft             ;otro haz FFT
ffta    equ     *
        move    r0,m3           ; salva N en m3 para uso posterior
        move    r0,A            ; N en A y r0 para cálculo posterior
        asr     A
        lua     (r0)-,r0  ; r0 = N-1
        move    A1,n0           ; A = N/2 -> n0
        move    #1,n2           ; inicializa contador de grupos por paso
        move    r0,m0           ; inicializa direccionamiento de a y b
        move    m0,m1           ;
        move    m0,m4
        move    m0,m5
        move    n2,n6           ; Direccionamiento lineal a través de tabla de inversión de bit
        move    #$FFFF,m6       ;
        do      r3,end_pass     ; Haz pasos de log2(N)
        move    r2,r0           ; inicializa arreglo apuntador a entrada
        move    r0,r4           ; inicializa apuntador de salida
        lua     (r0)+n0,r1      ; inicializa entrada b
        move    m2,r6           ; inicializa entrada c
        lua     (r1)-,r5        ; inicializa salida b
        move    n0,n1           ; inicializa apuntador offsets
        move    n0,n4
        move    n0,n5
        do      n2,end_grp
        move    x:(r1),X1       y:(r6),Y0       ; vc por valores de -sin y -cos
        move    n3,B                            ; precarga datos
        tst     B       x:(r5),A                ; si bandera de IFFT distinto de 1
        jne     _nift                           ; pone escala por .5
        lsl     A
        ori     #$04,mr
_nift
        move    x:(r6)+n6,X0    y:(r0),B        ; precarga datos
                                                ; actualiza apuntador c
;  Nota: Instrucciones etiquetadas con fft_1 y fft_2 pueden ser reemplazadas
;          para hacer el valor del seno en negativo de IFFT
        do      n0,end_bfy                      ; Radix 2 DIT, mariposa
fftg_1  mac     X1,Y0,B                         y:(r1)+,Y1
        macr    -X0,Y1,B        A,x:(r5)+       y:(r0),A
°       subl    B,A             x:(r0),B B,y:(r4)
        mac     -X1,X0,B        x:(r0)+,A       A,y:(r5)
fftg_2  macr    -Y1,Y0,B        x:(r1),X1
        subl    B,A             B,x:(r4)+       y:(r0),B
```

```
end_bfy
        move    A,x:(r5)+n5     y:(r1)+n1,Y1    ; actualiza apuntadores a y b
        move    x:(r0)+n0,X1    y:(r4)+n4,Y1
end_grp
        move    n0,B1
        andi    #$f3,mr ;
        lsr     B       n2,A1   ; divide mariposa por grupo (n0)
        lsl     A       B1,n0   ; multiplica grupos por paso (n2)
        move    A1,n2
end_pass
        andi    #$f3,mr         ;
        move    #0,n3           ; limpia bandera ifft
;  Bit reverse para reordenamiento post-FFT
        move    m3,A
        asr     A       A1,X0   ; carga X0 con N, divide entre 2
        clr     B       A1,n4   ; copia  offset de apuntador de salidaN/2 a n4
        clr     A       r2,r0   ; inicializa apuntador de entrada
        move    #$FFFF,m0       ; direccionamiento lineal
        move    r0,r4           ; inicializa apuntador de salida
        move    #0,m4           ; modo de carry inverso
        move    r0,B0           ;
; Haz reordenamiento inversión de bit
do    X0,_order
        move    r4,A0
        cmpm    B,A             ;es r0<r4?
        jlt     _do_mov         ;si es entonces Haz "move"
        lua     (r0)+,r0
        lua     (r4)+n4,r4
        jmp     _skip           ;siguiente lazo
_do_mov
        move    x:(r0),X0       y:(r4),Y1       ;swap datos x & y
        move    x:(r4),X1       y:(r0),Y0
        move    X1,x:(r0)       Y0,y:(r4)
        move    X0,x:(r4)+n4    Y1,y:(r0)+
_skip   move    r0,B0
_order
        move    m0,m1
        move    m0,m2
        move    m0,m3
        move    m0,m4
        move    m0,m5
        move    m0,m6
        rts
```

5.1.3 Transformada Discreta Cosenoidal.

Este programa calcula la transformada discreta cosenoidal (DCT) usando la FFT. [38]

Este algoritmo es como sigue:
1. La secuencia original en la memoria Y es ordenada [17].
2. Un FFT es efectuada sobre la secuencia modificada.
3. La salida de la FFT es multiplicada por una exponencial compleja y se toma la parte real.

4. El resultado es reordenado para obtener la secuencia adecuada.

$g(0) = g(0)*1.414/N$
$g(k) = g(k)*2.0/N \qquad k=1,2,...,N-1$

Programa 3
Transformada Discreta Cosenoidal

```
      include 'sincos'
      include 'fftr2aa'

start  equ    $100                ;Inicio del programa en memoria P
points equ    16                  ;número de puntos para dct
data   equ    0                   ;datos en memoria Y
coef   equ    16                  ;coefficients
      sincos  points,coef         ;genera twiddle factors para FFT.

rot    equ    pi/@cvf(2*points)
       org    x:
exptbl
count  set    0
       dup    points
       dc     -@sin(@cvf(count)*rot)
count  set    count+1
       endm
       org    y:
count  set    0
       dup    points
       dc     -@cos(@cvf(count)*rot)
count  set    count+1
       endm
       org    p:start
```

```
;           x(k)=y(2k)
;           x(N-1-K)=x(2k+1)      k=0,1,...,N/2-1

      move    #$ffff,m0
      move    m0,m4
      move    #data,r0                ;apuntador de datos
      move    #2,n0
      move    r0,r4
      move    y:(r0)+n0,a
      rep     #points/2
      move    y:(r0)+n0,a  a,x:(r4)+
      move    #data+1,r0
      move    #data+points-1,r4
      move    y:(r0)+n0,a
      rep     #points/2
      move    y:(r0)+n0,a  a,x:(r4)-
      clr     a       #data,r0
      rep     #points
      move    a,y:(r0)+

;     Parte real  dct=re[ (a+jb)*(c+jd)] = a*c-b*d donde a+jb es la
;     salida de la FFT y c+jd es el factor de rotación.

      move    #cxptbl,r0
      move    #$ffff,m0
      move    #data,r4
      move    r4,r5
      move    #0,m4
      move    m4,m5
      move    #points/2,n4
      move    n4,n5
      move    x:(r4),x0  y:(r0),y0
      do      #points,_gf
      mpy     -x0,y0,a  x:(r0)+,x1   y:(r4)+n4,y1
      macr    -x1,y1,a  x:(r4),x0    y:(r0),y0
      move            a,x:(r5)+n5
_gf
```

5.2 RESULTADOS.

En la figura 5.1 se observa la tarjeta prototipo, ocupando un espacio de 11x16 cms. En la parte superior izquierda se encuentra el DSP-56001, y en la parte derecha se encuentran los bancos de memoria externos.

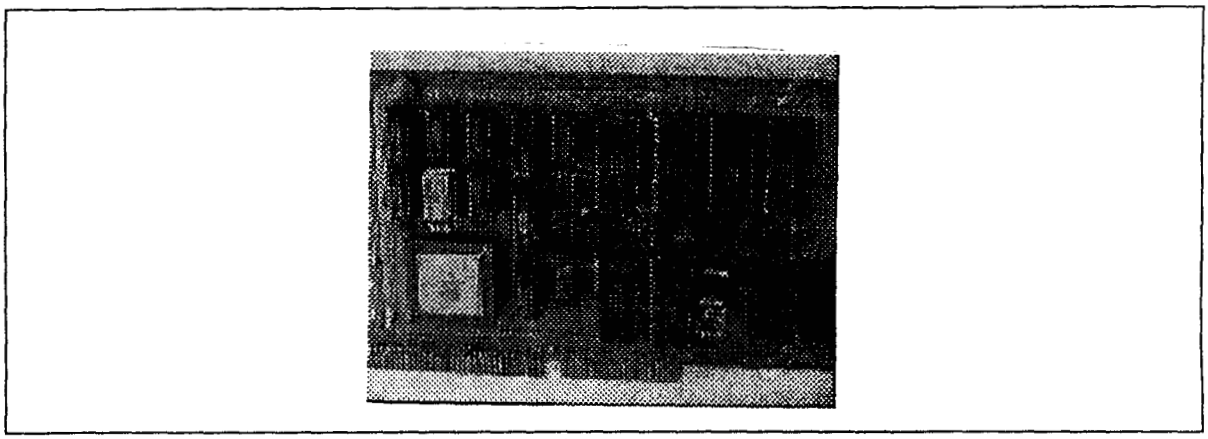


Figura 5.1
Tarjeta Prototipo con Interfase DSP56001

Las pruebas que se realizaron al sistema son las siguientes:

Se probó el sistema con el programa de la FFT (descrito anteriormente), que requiere aproximadamente de 40 palabras de programa en la memoria para una transformada entre 2 y 512 puntos. La rutina de FFT requirió de 3.39 milisegundos a una velocidad de aproximadamente 19.9 Mhz del reloj principal del DSP para 512 puntos (calculados por un programa interno de la PC). En las gráficas de las figuras 5.2a y 5.3a se observa la FFT a 512 y 256 puntos respectivamente, en estos ejemplos la señal de entrada es aleatoria (figura 5.2c y 5.3c) generada por la PC. Las figuras 5.2b y 5.3b son las mismas respectivamente solo que a través del banco de memorias externas del DSP X e Y, aquí se observa que la señales son la mismas respectivamente, comprobándose el funcionamiento correcto de las memorias.

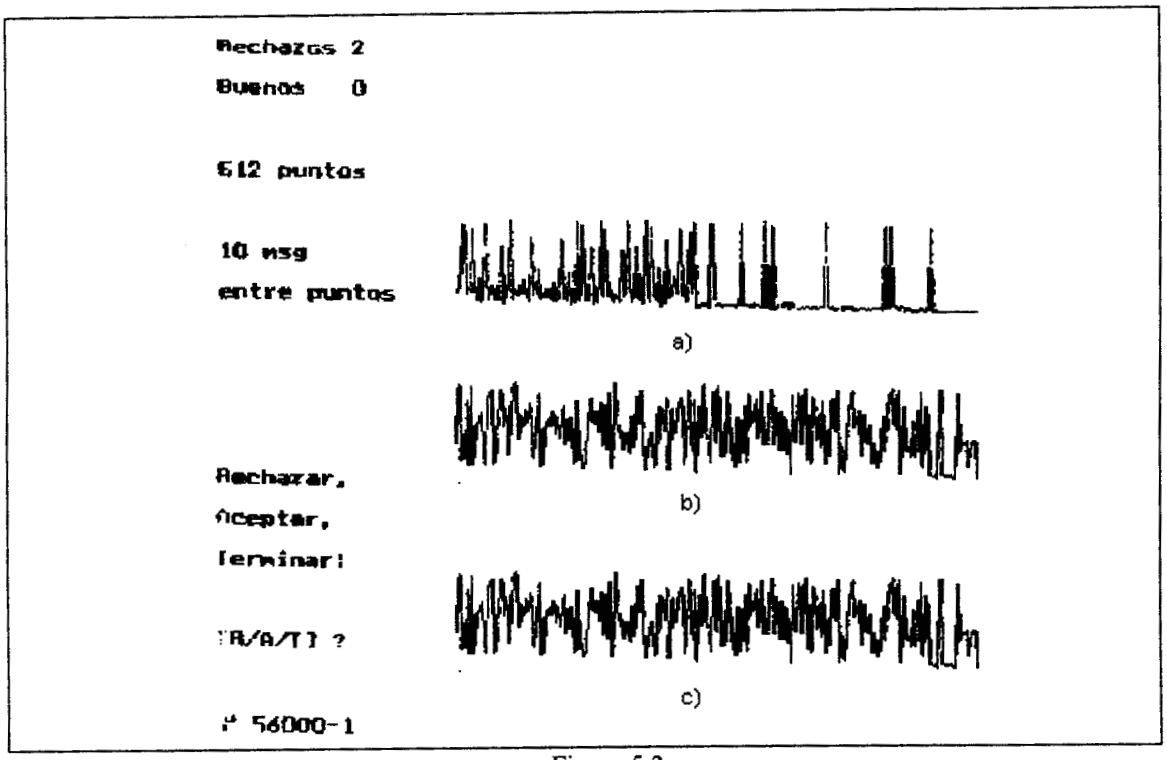


Figura 5.2
FFT de una señal aleatoria de 512 puntos.

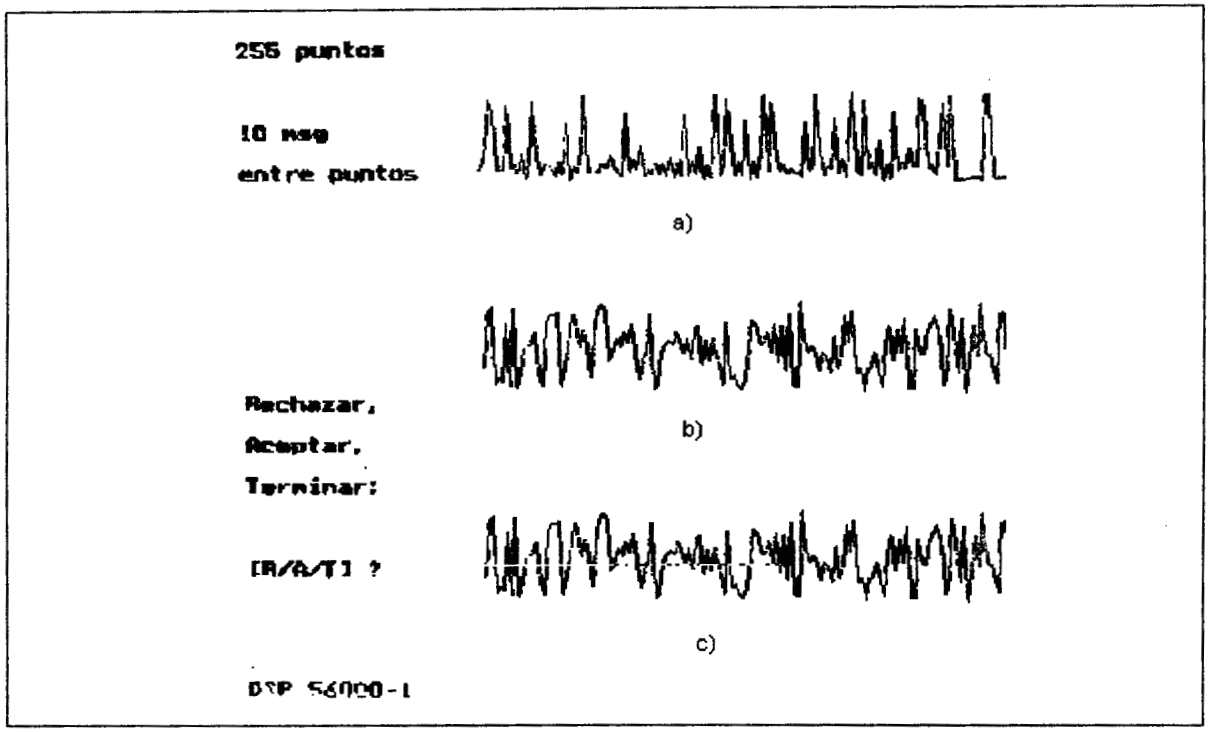


Figura 5.3
FFT de una señal aleatoria de 255 puntos.

Por otra parte, en la figura 5.4 se observa la gráfica de FFT a 128 puntos procesada por el paquete comercial MATLAB en una computadora de tipo personal 486 a 33 Mhz, se utilizó la misma señal aleatoria que para el procesamiento por parte del DSP 5600-1, con la diferencia de procesamiento a 128 puntos.

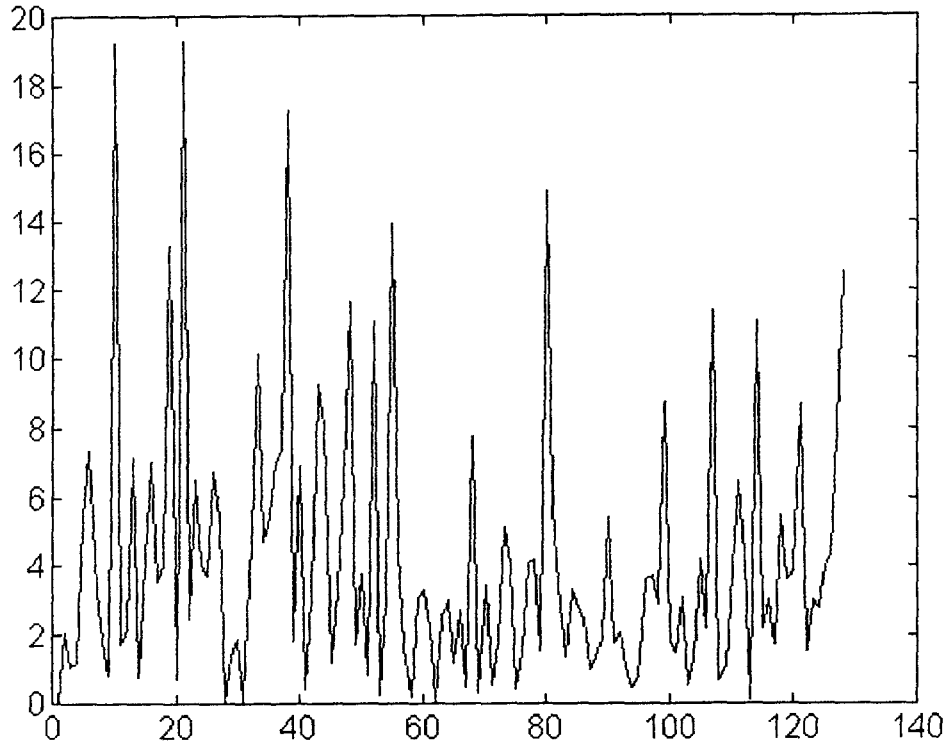


Figura 5.4
FFT procesada por MATLAB a 128 puntos

En la tabla 5.1 se observa los tiempos requeridos para diferente número de puntos, usando un reloj de 19 Mhz.

Puntos	Int.P,X,Y	Int.P,Ext.X,Y
16	0.032 msec	0.04 msec
64	0.148	0.23
256	0.712	1.17
512	( )	5.66

Los ( ) indican que no es posible con la memoria interna de datos del DSP560001.

Tabla 5.1
Tiempos de ejecución de la FFT por el DSP-56001

Como en el programa de la transformada discreta cosenoidal se utiliza la FFT (programa anteriormente probado), se obtuvieron los resultados mostrados en la tabla 5.2

Puntos	Int.P,X,Y	Int.P,Ext.X,Y
256	0.71 msec	1.75 msec
512	( )	5.81

Tabla 5.2
Tiempos de ejecución de la transformada discreta cosenoidal

Como se observa en las tablas 5.1 y 5.2, el tiempo de procesamiento es menor cuando no se hacen referencias a memoria externa, en el caso para la FFT los coeficientes son almacenados en la memoria externa de datos.

5.3 CONCLUSIONES.

Podría considerarse el diseño de una arquitectura basada en el modelo Harvard como complicado (capítulo 3) ya que el sistema desarrollado se ajustó a los siguientes puntos:

- El cumplimiento de los tiempos de acceso por parte del Host hacia el DSP y viceversa deben estar sincronizados tanto para el ciclo de escritura como de lectura, no descuidando que se logre el tiempo mínimo establecido por el fabricante.
- El desempeño adecuado de las memorias externas (X e Y) tipo S-RAM, se obtuvo sin la necesidad del uso de instrucciones "NOP" (no operación), recurso que usualmente se ocupa para la sincronización de dispositivos lentos con rápidos.

Esto implica que cualquier periférico mapeado a la memoria del DSP debe tener adaptaciones a los requerimientos de diseño anteriormente mencionados.

Por otro lado el uso de instrucciones de doble acceso, como

move x:(r6)+n6,X0 y:(r0),B

no afecta a la velocidad del sistema, (como se mencionó en el capítulo 2 el fabricante predice un tiempo de acceso de 3 ó 4 ciclos) además es un auxiliar para acceso a memoria que permite escribir ó leer a dos bancos de memoria simultáneamente haciendo eficiente el manejo de los recursos del DSP.

Se cuenta con dos propuestas para el mejor desempeño de la memoria:

- La primera es un diseño especial de memoria de doble acceso, recomendada por Motorola, sus principales ventajas consisten en el rápido acceso externo y compatibilidad con el DSP.
- La segunda propuesta sugerida por Stephen Jobbs [53] consiste en la ampliación de la memoria de datos utilizando el manejador de memoria MMU88200 [14, 53,54,55] de doble bus de direcciones/datos y memoria caché interna. Esta configuración es empleada por la "Next Station", y se basa en el microprocesador Motorola 68040 que se encuentra interconectado con el DSP 56001.

Para la decodificación del mapeo de memoria por parte de la PC, se puede implementar Arreglos de Lógica Programable (PLA), los principales beneficios de los PLA permiten decodificaciones rápidas y disminución de espacio físico. También se propone un desarrollo a futuro manejando DMA haciendo referencia a memoria por medio del HOST, logrando sistemas independientes.

Por último, a pesar de los problemas que surgan al momento de accesar a la memoria externa (principalmente cuestiones de tiempo), se tiene un sistema capaz de hacer procesamiento digital de señales en tiempo real, con características similares a las tarjetas comerciales de desarrollo (ARIEL Co.). La tarjeta prototipo cuenta finalmente con las siguientes características:

- Memoria S-RAM de datos externa X e Y de 32 Kbytes.
- Memoria interna de 512 bytes.
- Memoria de Programa de 512 bytes.
- Interfaz para computadora tipo personal.

Apéndice A.

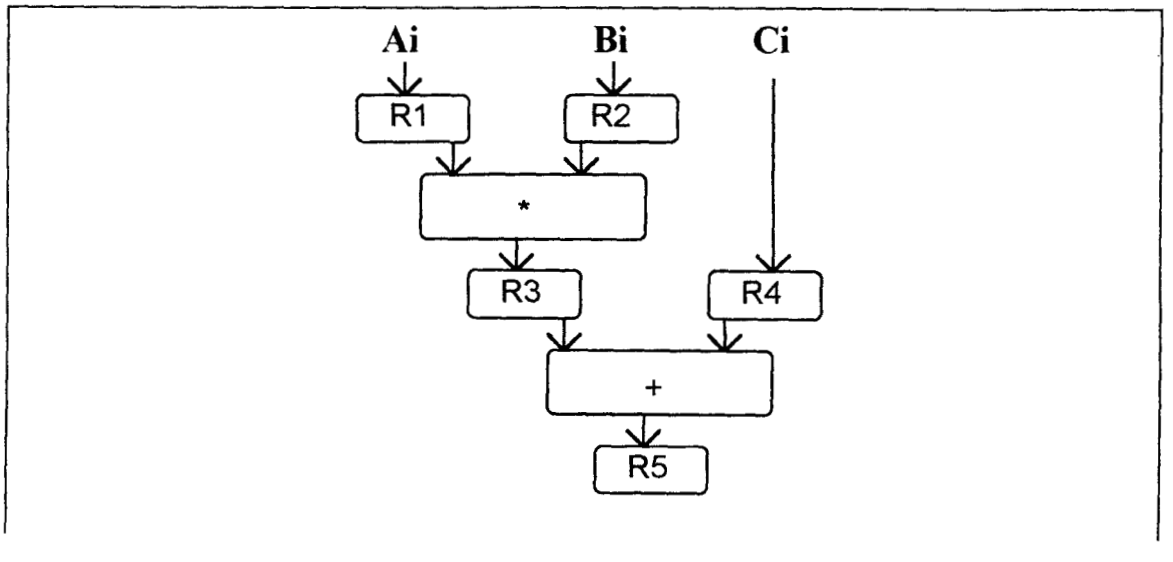
Procesamiento Pipeline.

La técnica de PIPELINE consiste en descomponer un proceso secuencial en sub-operaciones, en cada subproceso en ejecución existe un segmento especial dedicado que opera con otros segmentos. Un pipeline puede ser visualizado como una colección de segmentos procesados donde fluye información binaria. Cada segmento ejecuta procesamiento parcial dedicado por medio de partición de tareas. El resultado obtenido del proceso es transferido al siguiente segmento del pipeline. El resultado final es obtenido después de que el dato ha pasado a través de todos los segmentos. La característica del pipeline es que varias operaciones pueden ser ejecutadas en distintos segmentos al mismo tiempo. El traslape de las operaciones es posible asociando un registro con cada segmento en el pipeline, el registro nos da la separación entre los segmentos tal que puedan operar simultáneamente sobre datos separados.

Ejemplo:

Suponga que deseamos operar la combinación de sumar y ejecutar operaciones con varios números:

Ai * Bi + Ci para i = 0..7
donde R1..R5 son registros que reciben un
nuevo dato cada pulso de reloj



Los cinco registros son cargados con un nuevo dato en cada pulso de reloj.

pulso	segmento 1		segmento 2		segmento 3
	R1	R2	R3	R4	R5
1	A1	B1	--	--	-- --
2	A2	B2	A1*B1	C1	-- --
3	A3	B3	A2*B2	C2	A1*B1+C1
4	A4	B4	A3*B3	C3	A2*B2+C2
5	A5	B5	A4*B4	C4	A3*B3+C3
6	A6	B6	A5*B5	C5	A4*B4+C4
7	A7	B7	A6*B6	C6	A5*B5+C5
8	--	--	A7*B7	C7	A6*B6+C6
9	--	--	--	--	A7*B7+C7

En este ejemplo, se requiere de tres pulsos de reloj para llenar al Pipeline, y recuperar el primer dato en R5, cada pulso produce una nueva salida moviendo el dato un paso abajo al pipeline. Cuando no hay más datos disponibles, los pulsos de reloj continúan hasta que emerge la última salida del pipeline.

Note que el procesador ejecuta operaciones simultáneas en cada segmento. No importa cuantos segmentos haya en el sistema, una vez que el pipeline se llena, toma un pulso de reloj para obtener la salida, sin importar cuantos pasos son requeridos para ejecutar un proceso entero. Si el tiempo que toma para procesar una operación en cada segmento es un intervalo t, y si hay k segmentos, entonces cada operación total se completa en k*t tiempos.

Cualquier operación que pueda ser descompuesta en secuencia de suboperaciones de la misma complejidad, puede ser implementada por un procesador pipeline. El procesamiento de datos por técnica pipeline ha sido aplicado regularmente en operaciones aritméticas de punto flotante.

Apéndice B.

```
{PROGRAMA FINAL DE TESIS DE MAESTRIA DE INGENIERIA BIOMEDICA,
"DESARROLLO DE UN PROCESADOR AUXILIAR PARA PROCESAMIENTO DIGITAL DE
SEÑALES,BASADO EN EL
MOTOROLA DSP 56001"}

Program sdp_h3;

{ Programa para inicializar la memoria XY, y P.
        (permite cargar varios bloques)
1.- Carga en el DSP el programa que permitira  la inicializacion
2.- Se ejecuta un programa en el DSP el que:
     pide la zona de memoria X y/o Y,
     pide el numero de datos a enviar
     y pide la direccion inicial
3.- envia datos al DSP
4.- carga el programa de aplicacion (unicamente el origen)
5.- ejecuta el programa de aplicacion

}

Uses crt,dos;

const
   PC56_ICR = $340;
   PC56_CVR = $341;
   PC56_ISR = $342;
   PC56_IVR = $343;
   PC56_NUL = $344;
   PC56_HI  = $345;
   PC56_MID = $346;
   PC56_LOW = $347;
   timeout  = 20000; { contador de espera }
   splt     = FALSE;
   maxd     = 256;
   maxm     = 1024;{ empleadas en lectura de PXY}
   maxb     = 10;

type  {empleadas en lecturas PXY}

   tbase = array[1..maxb] of longint;
   tmem  = array[1..maxm] of longint;
   tn    = array[1..maxb] of integer;

var
   error  :  boolean;
   ndd    :  longint;
   i,ik,j :  integer;
   d      :  array[0..maxd] of longint;
   tiempo :  real;
   {empleadas en lectura PXY}
```

```
    np,nx,ny  : tn;
    memp,memx,memy  : tmem;
    basex,basey,basep  : tbase;
    ip,ix,iy  : integer;
    gp,gx,gy  : integer;
    nombre    : string;

Function ftiempo(ti:real):real;

Var
  h,m,s,c : word;
  Begin
      GetTime(h,m,s,c);
      ftiempo:= ( 60.0*h+m)*60.0+s+c/100.0-ti;
  end;

Function Hexval(n:longint):string;

Const
    Hex : array [0..15] of char = '0123456789ABCDEF';
Var
  i,j  : Integer;
  ss   : String;

  Begin
      i:=0;
      ss:='';
      Repeat
          j:= n mod 16;
          ss:= hex[n mod 16] +ss;
          n:=(n-j) div 16;
          i:= i+1;
      Until (n<16) or (i>16);
      hexval := '$'+hex[n] + ss;
  End;


{Lectura de PXY realiza la lectura de los bloques de datos que serviran
para inicializar las memorias X,Y, y P.
--memx,memy,memp     contienen los datos;
--basex,basey,basep  contienen las direcciones de base de los bloques
                     respectivos;
--nx,ny,np             contienen los tamanios de cada bloque;
--ix,iy,ip           son el numero de bloques a cargar
                     para X,Y, y P respectivamente.
}

Procedure lecturapxy;

type
   tren = string[70];
   tdato = string[7];

Var
```

```
        cuando se emplea write(dato) en guarda}
    until (dato='') or (dato[1]='_');
    readln(f,ren); ren:=dato+ren;
    {writeln(ren)}
end;

Begin;    {LECTURA PXY}
    i:=0;
    repeat
        write('nombre del archivo (sin extensión): ');
        readln(nombre);
        nombre:=nombre + '.LOD';
        assign(f,nombre);
        {$I-}
        reset(f);
        {$I+}
        errorr:=false;
        if ioresult<>0 then
        begin
            errorr:=true;
            Writeln('Nombre no existe');
        End;
        i:=i+1;
        if i=2 then writeln('Es tu ultima oportunidad');
    Until (not errorr) or (i=3);
    if errorr then halt;

    ip:=0;{contadores de bloques}
    ix:=0;
    iy:=0;

    gp:=1;{contadores generales}
    gx:=1;
    gy:=1;

    readln(f,ren);
    repeat
        if pos('_DATA',ren) > 0 then
        Begin
            case ren[7] of
            'P':begin;
                    lectura(gp,ip,np,basep,memp);
                    writeln;
                end;
            'X':begin;
                    lectura(gx,ix,nx,basex,memx);
                    writeln;
                end;
            'Y':begin;
                    lectura(gy,iy,ny,basey,memy);
                    writeln;
                end;
            end;
        end
```

```
        else
          begin

            readln(f,ren);{prepara otro renglon}
            writeln(ren);{renglon poco interesante}
          end;
    until pos('_END',ren) > 0;
    close(f);

    for i:= 1 to ip do
    writeln('p':10,i:2,' origen ',hexval(basep[i]):5,' -> ',np[i]:4,' datos ');
    for i:= 1 to ix do
    writeln('x':10,i:2,' origen ',hexval(basex[i]):5,' -> ',nx[i]:4,' datos ');
    for i:= 1 to iy do
    writeln('y':10,i:2,' origen ',hexval(basey[i]):5,' -> ',ny[i]:4,' datos ');
    end;


procedure HP_TEST;

Begin
   Port[PC56_IVR] := $AA;
   {envia un byte al puerto de 16 bits y lo recupera}
   if (Port[PC56_IVR] and $FF) <> $AA then {son iguales ?}
   begin
      writeln(' No existe enlace con el DSP 56000 en direccion ',hexval(PC56_ICR));
      halt;
      end
   else
   writeln('La direccion del DSP 56000 es: ',hexval(PC56_ICR));
   end;

Procedure HP_RESET(splitmem:boolean);
begin
   Port[PC56_ICR+$C000]:=0; {reset on}
   {delay(50);}
   Port[PC56_ICR+$8000]:=0; {reset off}
   if splitmem then
     Port[PC56_ICR+$A000]:=0; {split memory map}
     {delay(50);}
End;

{envia un longint al puerto B de DSP }

Procedure HP_Send(data:longint;var error:boolean);

Var
  XMIT_RDY,b : byte;
  count      : longint;

begin
   XMIT_RDY := $2;
   error    := FALSE;
```

```
    count    := 0;

While ((port[PC56_ISR] and XMIT_RDY)<> $2) and not error do
    begin
        if (count > timeout) then error:= true;
        inc(count);
    end;
    if not error then
      begin
          b:=data div 65536; port[PC56_HI]:=b;
          {write(' ',hexval(b));}
          b:=data div 256; port[PC56_MID]:=b;
          {write(' ',hexval(b));}
          b:=data ; port[PC56_LOW]:=b;
          {write(' ',hexval(b));}
      end
    else writeln(' >> Send-Time Out ');
end;

{Recibe un longint del puerto B del DSP}
Procedure HP_Receive(var data:longint;var error: boolean);

var
  RCV_RDY : Byte;
  count   : longint;

const
    f : LONGINT=256; {si no de define asi, 256 = 0}

begin
    RCV_RDY := 1; error := false; count := 0;
    {no funciona la bandera de error y lee $FFFFFF como dato}
    while ((Port[PC56_ISR] and RCV_RDY)<>$1) and not error do
        begin
            if (count > timeout) then error := true;
            inc(count);
        end;
        if not error then
          Begin
              data:=Port[PC56_HI]*65536 + Port[PC56_MID]*f + Port[PC56_LOW];
              Writeln(hexval(data));
          end
        else
           writeln(' >> Receive-Time Out');
    end;

{Interrumpe la recepcion de datos en el DSP}

Procedure Termine;
Var
  ICR : byte;

Begin
    ICR:=Port[PC56_ICR];
```

```
    ICR:=ICR or $8; {bit 3 (HF0) miss}
    Port[PC56_ICR]:=ICR;
end;


Procedure cargapxy(var err:boolean);

const
    ndb = 58;    { DSP_H3.asm}

     px: array[1..ndb] of longint = (
     $08F4A8,$000000,$08F4A0,$000001,$0AA980,$000004,$08582B,$06D800,
     $000014,$0AA980,$000009,$08502B,$0AA980,$00000C,$08592B,$06D900,
     $000013,$0AA980,$000011,$0858AB,$000000,$000000,$0AA980,$000016,
     $08582B,$06D800,$000026,$0AA980,$00001B,$08502B,$0AA980,$00001E,
     $08592B,$06D900,$000025,$0AA980,$000023,$0858EB,$000000,$000000,
     $0AA980,$000028,$08582B,$06D800,$000038,$0AA980,$00002D,$08502B,
     $0AA980,$000030,$08592B,$06D900,$000037,$0AA980,$000035,$08586B,
     $000000,$000000);


Var
  k : integer;

Begin
    HP_Reset(splt);
    for k:= 1 to ndb do
       HP_Send(px[k],err);
       termine;
end;

Procedure envia(var g,ii:integer; var n:tn; var base:tbase; var mem:tmem);

begin
    hp_send(ii,error); {envia el numero de bloques}
    if error then writeln(' error en la transmision');
      g := 1;
      for i:= 1 to ii do
         Begin
             hp_send(base[i],error); {envia la base}
             if error then writeln('error en la transmision');
               for j:= 1 to n[i] do
                  begin
                      hp_send(mem[g],error); {envia los datos}
                      g:=g+1;
                      if error then writeln('error en la transmision');
                  end;
         end;
         if error then writeln('error en la transmision')
         else
            write('carga completa de la memoria')
end;
```

```
Begin { Programa Principal}
   clrscr;
   hp_reset(splt);
   writeln('salida del reset al DSP-56000');
   HP_Test;
   lecturapxy;

   writeln('Cargado al DSP');
   cargapxy(error);
   if not error then
     begin
        gotoxy(0,15);
        envia(gx,ix,nx,basex,memx);
        writeln(' X');
        gotoxy(0,16);
        envia(gy,iy,ny,basey,memy);
        writeln(' Y');

        writeln('carga completa de la memoria de Programa');
        hp_send(ip,error);{envia el numero de bloques}
        if error then writeln('error en la transmision');
        gp := 1;              {se emplea no por ahora}
        case ip of
           0 : ;

           2..maxb :
              begin
                 writeln('no cargo p: se tiene mas de un bloque de datos');
              end;

           1 :  if basep[1] = 0 then
           begin
              writeln('origen $0000 ');
              {carga con la rom y ejecuta}
              HP_Reset(splt);
              {for i:= 1 to np[1] do
              write(memp[i]);}
              for i:= 1 to np[1] do
              HP_Send(memp[i],error);
              writeln('El programa ',nombre);
              termine;
              writeln('se encuentra corriendo a toda velocidad');
           end
           else {carga con el programa}
           begin
              hp_send(basep[1],error); {envia la base}
              if error then writeln('error en la transmision');
              hp_send(np[1],error);{envia el numero de datos}
              if error then writeln('error en la transmision');
              for i:=1 to np[1] do
              begin
                 hp_send(memp[i],error); {envia los datos}
                 if error then writeln('error en la transmision');
              end;
```

```
                {para comensar la ejecucion : }
                {escribe en $0000 un brinca a base[1] y hace un reset}
                HP_reset(splt);
                HP_Send(($0C0000 or basep[1]),error);
                write(' el programa ', nombre);
                termine;
                writeln(' se encuentra corriendo a toda velocidad');
            end;

        end;{case}

              {envia(gp,ip,np,basep,memp);}
    {end;no se como pero con este end funciono el programa}
    end
    else
      writeln(' error de carga del programa de lectura');
readln;
{HP_Receive(memp[1],error);}
end.
```

DESARROLLO DE UN SISTEMA AUXILIAR PARA EL PROCESAMIENTO DIGITAL DE SEÑALES BASADO EN EL DSP56000-1

Apéndice C.

PROGRAMA PRINCIPAL DSP.

```
        page    135
        opt     cex,nomd,mex,mu


pixdB   equ     4               ;
nshifts equ     (7-@cvi(@log(pixdB)/@log(2.0)))
pixels  equ     280             ;
maxy    equ     30              ;
offset  equ     (maxy*256)      ;
correct equ     $05B000         ;
                                ;
                                ;
maxdB   equ     0               ;
mindB   equ     (-(pixels/pixdB)*32768)      ;

points  equ     255
coef    equ     1024
data    equ     0000
order   equ     3               ;
npts    equ     20              ;


sci1    ident   1,0
        include 'ioequ'
        include 'sincos'
        include 'wbh4m'

        wbh4m   points
        sincos  points,coef

        org     x:1000
state   ds      10              ;

        org     y:1000
kadd    dc      .5,-.5,.2       ;

        ORG     P:$0000

;  interrupt vector table

    jmp     Inicio      ; hardware vector de reset
```

```
        nop             ;
        nop             ;
        nop             ;
        nop
        nop
        nop

;

;       ORG   P:$0008           IRQA
        NOP
        NOP

;       ORG   P:$000A           ; IRQB
        NOP
        NOP

;       ORG   P:$000C           ; SSI
        NOP
        NOP

;       ORG   P:$000E           ; SSI rcv
        NOP
        NOP

;       ORG   P:$0010           ; SSI
        NOP
        NOP

;       ORG   P:$0012           ; SSI
        NOP
        NOP

;       ORG   P:$0014           ; SCI
        NOP
        NOP

;       ORG   P:$0016           ; SCI
        NOP
        NOP

;       ORG   P:$0018           ; SCI
        NOP
        NOP
```

```
;       ORG   P:$001A           ; SCI
        NOP
        NOP

;       ORG   P:$001C           ; SCI
        NOP
        NOP

;       ORG   P:$001E           ;
        nop                     ;
        nop                     ;

;       ORG   P:$0020           ; Host receive data
        nop
      nop


;       ORG   P:$0022           ; Host transmit data
        nop
      nop


;       ORG   P:$0024           ;host command
        nop
      nop

; Victor`s Host Commands

;       ORG   P:$0026           ;Host Command $13
                                ;Lectura de datos de la PC
      jsr   >read

;       ORG   P:$0028           ;Host Command $14
                                ;Escitura de datos a la PC
      jsr   >write

; Host Commands restantes, para definidas aplicaciones

;       ORG   P:$002A           ;Host Command $15
      nop
      nop                       ;

;       ORG   P:$002C           ;Host Command $16
        jsr   usfft             ;Brinca a rutina FFT
```

```
;       ORG   P:$002E           ;Host Command $17
        jsr   usifft            ;realiza una IFFT


;       ORG   P:$0030           ;Host Command $18
        NOP                     ;
        NOP                     ;

;       ORG   P:$0032           ;Host Command $18
        NOP                     ;
        NOP                     ;

;       ORG   P:$0034           ;
        NOP                     ;
        NOP                     ;

;       ORG   P:$0036           ;Host Command $18
        NOP                     ;
        NOP                     ;

;       ORG   P:$0038           ;
        NOP                     ;
        NOP                     ;

;       ORG   P:$003A           ;Host Command $18
        NOP                     ;
        NOP                     ;

;       ORG   P:$003C           ;Host Command $18
        NOP                     ;
        NOP                     ;

;       ORG   P:$003E           ;
        NOP                     ;
        NOP                     ;

;       ORG   P:$0040

;Programa principal para inicio de variables y configuracion del sistema
;DSP 56000/1.
;solo se activa a traves del reset, y por cada vector del Host Command
;para salir del mismo, hay que realizar otro reset.
```

```
Inicio movep  #1,x:M_PBC          ;Enciende el puerto b(HOST)
     movep  #$0c00,x:M_IPR        ;Pon prioridad de interrupciones
     movep  #0,x:M_HCR            ;Deshabilita interrupciones
     move   #0,sr                 ;Desenmascara interrupciones
     move   #>$8,x0               ;Pon bandera par verificar HF0 de HI1
Loop  movep  x:M_HSR,a            ;Espera que HF0=0 por parte del programa
     and    x0,a                  ;PADA.PAS sea inicializada
     jeq    Loop
     ;movep  #$12,x:M_HCR         ;Avisa a HI a traves de TESIS.PAS que se HF3
  ;rutina para inicializar a la recepcion del DSP por interrupciones
     ;movep  #$11,x:M_HCR         ;Prende el bit de rx DSP
     movep  #$14,x:M_HCR          ;enciende flag de Host Command
     ;movep   #$17,x:M_HCR        ;enciende la Excepciones de Rx,Tx y HC
     ;movep    #$13,x:M_HCR       ;Interrupciones Rx y Tx
     jmp    *                     ;recibio por medio de HF3


read   equ   *
     move  #data,r1
     do    #points,sread          ;
vread  jclr  #M_HRDF,x:<<M_HSR,vread ;
     movep  x:M_HRX,x:(r1)+        ;
sread
     rti


write  equ    *
     move   #0,r0                  ;
     do     #points,swrite         ;
vwrite jclr     #M_HTDE,x:M_HSR,vwrite  ; espera a Host para trasmitir dato
                                   ; a que este vacio para envio
     movep  x:(r0)+,x:M_HTX        ;Transfiere el dato a HI
     ;Rutina para solo sacar #points datos hacia HI
swrite
     rti

usifft equ    *
     bclr   #M_HF3,x:M_HCR  ;
     move     #1,n3                ; clr bandera IFFT
     jmp    usrfft                 ; llama rutina FFT

usfft   equ    *
     bclr   #M_HF3,x:M_HCR  ;
     move     #0,n3                ;
     jmp    usrfft                 ;
```

```
usrfft  equ     *

;  Set up parameters

;  Clear imaginary array

        move  #>1,A             ; ordena memoria  y:flag (1)
        movemA,p:memflag
        jsr     zerom           ; limpia arreglo de memoria y

;  Window data

        clr     A               ; pass x memory flag (0)
        movemA,p:memflag
      jsr       window
      clr       A               ; pass x memory flag (0)
        movemA,p:memflag
        jsr     window
      ;jsr    fir
;  Perform complex FFT or IFFT
        movemp:N,r0
        movemp:log2N,r3
        movemp:dataadr,r2
        movemp:coefadr,m2
        move    n3,b
        tst     b
        jne     ift
        jsr     fft
        jmp     lmag
ift     jsr     ifft            ;
        ;jmp    kaw

;   magnitud
lmag    ;jsr    logmag
kaw     bset    #M_HF3,x:M_HCR  ;

;*************************************************************************

zerom  equ      *

        movemp:memflag,A
        movemp:dataadr,r0
        movemp:N,n0
        jsr     <clrm
```

```
        rts

;*************************************************************************

clrm    equ     *

        clr     B

        tst     A
        jne     yspace

        rep     n0              ;
clr_x   move    B,x:(r0)+
        rts

yspace  rep     n0
clr_y   move    B,y:(r0)+
        rts

;*************************************************************************
window          equ     *

        movemp:N,A
        move    A,n0
        asr     A
        move    A,n1
        movemp:dataadr,r0
        movemp:windadr,r4

        move    #>1,A
        movemp:log2N,X0     ; log2(N)
        movemp:log2max,B    ; log2(maxN)
        sub     X0,B
        jeq     no_shft
        do      B,e_exp
        lsl     A
e_exp
no_shft         move    A,n4

        movemp:memflag,A

        jsr     <wnd

        rts
```

```
wnd0  move  B,x:(r0)+
wnd1  move  B,y:(r0)+

wnd   equ   *

      tst   A
      jeq   x_wind
      move  p:wnd1,A
      jmp   wnd_0
x_windmove  p:wnd0,A
wnd_0 move  A,p:wnd2
      move  A,p:wnd3


      move  r4,A1

;  Do in-place window.

      do    n1,e_wnd1
      move  x:(r0),X0
      move  x:(r4)+n4,Y0
      mpy   X0,Y0,B
wnd2  move  B,x:(r0)+
e_wnd1

      move  A1,r4

      do    n1,e_wnd2
      move  x:(r0),X0
      move  y:(r4)+n4,Y0
      mpy   X0,Y0,B
wnd3  move  B,x:(r0)+
e_wnd2

      rts


;*************************************************************************

fft   equ   *
    jsr     <ffta
    rts
```

```
ifft    equ     *               ; FFT inversa
        jsr     <iffta
      rts

ffta    equ     *

        move    r0,m3
        move    r0,A
        asr     A
        lua     (r0)-,r0
        move    A1,n0

        move    #1,n2

        move    r0,m0
        move    m0,m1
        move    m0,m4
        move    m0,m5

        move    n2,n6
        move    #$FFFF,m6

        do      r3,end_pass
        move    r2,r0
        move    r0,r4
        lua     (r0)+n0,r1
        move    m2,r6
        lua     (r1)-,r5
        move    n0,n1
        move    n0,n4
        move    n0,n5

        do      n2,end_grp
        move    x:(r1),X1       y:(r6),Y0
        move    n3,B
        tst     B       x:(r5),A
        jne     _nift
        lsl     A
        ori     #$04,mr
_nift
        move    x:(r6)+n6,X0  y:(r0),B

        do      n0,end_bfy
fftg_1  mac     X1,Y0,B                         y:(r1)+,Y1
        macr    -X0,Y1,B        A,x:(r5)+       y:(r0),A
```

```
        subl    B,A             x:(r0),B        B,y:(r4)
        mac     -X1,X0,B        x:(r0)+,A       A,y:(r5)
fftg_2  macr    -Y1,Y0,B        x:(r1),X1
        subl    B,A             B,x:(r4)+       y:(r0),B
end_bfy
        move    A,x:(r5)+n5     y:(r1)+n1,Y1
        move    x:(r0)+n0,X1    y:(r4)+n4,Y1
end_grp
        move    n0,B1

        andi    #$f3,mr

        lsr     B       n2,A1
        lsl     A       B1,n0
        move    A1,n2
end_pass

        andi    #$f3,mr
        move    #0,n3
        move    m3,A
        asr     A       A1,X0
        clr     B       A1,n4

        clr     A       r2,r0
        move    #$FFFF,m0
        move    r0,r4
        move    #0,m4

        move    r0,B0

        do      X0,_order
        move    r4,A0
        cmpm    B,A             ;is r0<r4?
        jlt     _do_mov
        lua     (r0)+,r0
        lua     (r4)+n4,r4
        jmp     _skip

_do_mov         move            x:(r0),X0       y:(r4),Y1
        move    x:(r4),X1       y:(r0),Y0
        move    X1,x:(r0)       Y0,y:(r4)
        move    X0,x:(r4)+n4    Y1,y:(r0)+
_skip   move    r0,B0

_order
```

```
        move  m0,m1
        move  m0,m2
        move  m0,m3
        move  m0,m4
        move  m0,m5
        move  m0,m6

        rts
;*************************************************************************
;ifft    equ     *

        ;move  #N,r0
        ;move  #log2N,r3
        ;move  #data,r2
        ;move  #coef,m2
      ;move   #0,n3
        ;jsr   <iffta
      ;rts

iffta   equ     *

        move  r0,m3
        move  r0,A
        asr   A
        lua   (r0)-,r0
        move  A1,n0           ; A = N/2 -> n0

        move  #1,n2

        move  r0,m0
        move  m0,m1
        move  m0,m4
        move  m0,m5

        move  n2,n6
        move  #$FFFF,m6

        do    r3,iend_pass    ; Do log2(N) passes
        move  r2,r0
        move  r0,r4
        lua   (r0)+n0,r1
        move  m2,r6
        lua   (r1)-,r5
        move  n0,n1
```

```
        move  n0,n4
        move  n0,n5

        do    n2,iend_grp
        move  x:(r1),X1      y:(r6),Y0
        move  n3,B
        tst   B      x:(r5),A
        jne   _inift
        lsl   A
        ori   #$04,mr
_inift
        move  x:(r6)+n6,X0  y:(r0),B

        do    n0,iend_bfy           ; Radix 2 DIT butterfly kernel
ifft_1  mac   -X1,Y0,B                    y:(r1)+,Y1
        macr  -X0,Y1,B      A,x:(r5)+     y:(r0),A
        subl  B,A           x:(r0),B      B,y:(r4)
        mac   -X1,X0,B      x:(r0)+,A     A,y:(r5)
ifft_2  macr  Y1,Y0,B           x:(r1),X1
        subl  B,A           B,x:(r4)+     y:(r0),B
iend_bfy
        move  A,x:(r5)+n5   y:(r1)+n1,Y1
        move  x:(r0)+n0,X1  y:(r4)+n4,Y1
iend_grp
        move  n0,B1

        andi  #$f3,mr

        lsr   B     n2,A1
        lsl   A     B1,n0
        move  A1,n2
iend_pass

        andi  #$f3,mr
        move  #0,n3

        move  m3,A
        asr   A     A1,X0
        clr   B     A1,n4
        clr   A     r2,r0
        move  #$FFFF,m0
        move  r0,r4
        move  #0,m4

        move  r0,B0
```

```
        do      X0,_iorder
        move    r4,A0           ;
        cmpm    B,A             ;is r0<r4?
        jlt     _io_mov
        lua     (r0)+,r0
        lua     (r4)+n4,r4
        jmp     _iskip

_io_mov move            x:(r0),X0       y:(r4),Y1
        move    x:(r4),X1       y:(r0),Y0
        move    X1,x:(r0)       Y0,y:(r4)
        move    X0,x:(r4)+n4    Y1,y:(r0)+
_iskip  move    r0,B0
_iorder

        move    m0,m1
        move    m0,m2
        move    m0,m3
        move    m0,m4
        move    m0,m5
        move    m0,m6

        rts

;*************************************************************************
logmag          equ     *

        movemp:dataadr,r0
        movemp:L,r6

        do      r6,e_logm

        move    x:(r0),X0
        mpy     X0,X0,A                 y:(r0),Y0
        mac     Y0,Y0,A


        jsr     >db48
        move    A,x:(r0)+
e_logm

        rts

        opt     cc
```

```
db48    equ     *
        tst     A               ; if A <= 0 then
        jgt     log2            ; (-144.0 dB)
        move    #$B80000,A      ; else
        rts                     ;
                                ;

log2    equ     *               ;
        move    #0,r7
        rep     #47
        norm    r7,A
        move    A,X0

        move    #0.7689,B       ;  0.7689 => B
        cmp     B,A
        jge     third           ;  if(A < 0.7689)then

poly    mpyr    X0,X0,A                 ;     x**2
        move    #>0.9981958,Y0          ;     get a1
        mpy     X0,Y0,A     A,X1        ;     a1*x, mv x**2
        move    #>-0.3372223,Y0         ;     get a2
        mac     X1,Y0,A                 ;     a2* x**2
        move    #>-0.6626105,Y0         ;     get a0
        add     Y0,A            ;
        rep     #4              ;
        asr     A               ;
        jmp     add_ex          ;

                                ;
third   move    #>$800000,A     ;     move 1.0 to A
        neg     A
        sub     X0,A            ;     compute 1 - mantissa
        neg     A       A,X0
        mpyr    X0,X0,B         ;     B = (1 - mantissa)**2
        move    B,Y0
        asr     B               ;     B = B/2
        sub     B,A             ;     A = first and second order terms
        mpyr    X0,Y0,B                 ;     B = (1 - mantissa)**3
        move    #>0.33333333,Y0
        move    B,X0
        macr    -X0,Y0,A        ;
        move    A,X0            ;
        move    #$02E2AA,Y0             ;
        mpy     X0,Y0,A                 ;
```

```
add_exmove  R7,X1          ;
       move  #$10000,Y1    ;
       mpy   X1,Y1,B       ;
       move  B0,X0         ;.
       add   X0,A          ;
       rnd   A


       move  A1,X1         ;
       move  #$605460,Y1   ;
       mpyr  X1,Y1,A       ;
;                          ;

set_db                     ;

       rts

       end   usrfft
```