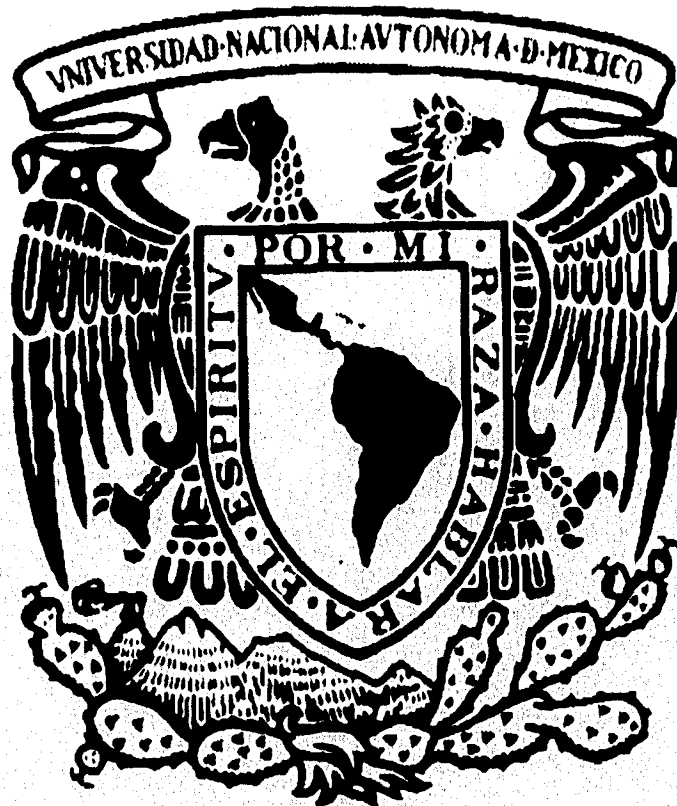
2

UNIVERSIDAD NACIONAL AUTÓNONA DE MÉXICO

FACULTAD DE INGENIERÍA



PROCESAMIENTO PARALELO EN ESTIMACIÓN ESPECTRAL DE SEÑALES UTILIZANDO UNA ARQUITECTURA HETEROGÉNEA

TESIS QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

PRESENTA

LUIS AGUIRRE TORRES

DIRECTOR:

DR. FABIÁN GARCÍA NOCETTI

JULIO DE 1996

TESIS CON
FALLA DE ORIGEN

TESIS CON
FALLA DE ORIGEN

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Dedico este trabajo

a mi papá, quien siempre ha estado presente y a quien le debo toda mi vida

a mi mamá, por todo lo que significa en mi vida

a Krysia porque este trabajo representa la culminación de una etapa que hemos compartido y que es el inicio de una vida juntos

a mis hermanos Sandra, Laura, Gabriela y Jorge

a todos mis amigos

Agradezco

Al Dr. Fabián García Nocetti, por la asesoría y apoyo brindados para la elaboración de este trabajo.

Al Departamento de Electrónica y Automatización del IIMAS, por la oportunidad de realizar este trabajo de investigación.

A Marco Antonio Vigueras por su colaboración en este trabajo.

Al P. A. R. A. de la Facultad de Ingeniería por el apoyo que me brindó los cinco años que duraron mis estudios de licenciatura.

ÍNDICE

1 INTRODUCCIÓN 2

1.1 INTRODUCCIÓN GENERAL 2

1.2 OBJETIVOS 3

1.3 CONTENIDO DE LA TESIS 3

1.4 REFERENCIAS 5

2 ANTECEDENTES DE PROCESAMIENTO PARALELO 7

2.1 INTRODUCCIÓN 7

2.2 CLASIFICACIÓN 9

2.2.1 SISD (SIMPLE INSTRUCTION, SIMPLE DATA) 9

2.2.2 SIMD (SIMPLE INSTRUCTION, MULTIPLE DATA) 10

2.2.3 MISD (MULTIPLE INSTRUCTION, SIMPLE DATA) 11

2.2.4 MIMD (MULTIPLE INSTRUCTION, MULTIPLE DATA) 11

2.3 CLASIFICACIÓN DE TOPOLOGÍAS 12

2.3.1 RED DE PROCESADORES TOTALMENTE CONECTADA 12

2.3.2 TOPOLOGÍA DE ESTRELLA 13

2.3.3 TOPOLOGÍA DE BUS (PIPELINE) 13

2.3.4 TOPOLOGÍA DE ANILLO 14

2.3.5 TOPOLOGÍA DE MALLA 14

2.3.6 TOPOLOGÍA DE ÁRBOL 15

2.4 FAMILIA DE PROCESADORES INMOS TRANSPUTER 15

2.4.1 CANALES DE COMUNICACIÓN PUNTO A PUNTO 18

2.5 PROGRAMACIÓN PARALELA 20

2.5.1 OCCAM 21

2.5.2 C PARALELO 22
2.6 REFERENCIAS 23
3 ANTECEDENTES DE ESTIMACIÓN ESPECTRAL 26
3.1 INTRODUCCIÓN: HISTORIA 26
3.2 CONCEPTOS BÁSICOS 29
3.2.1 SECUENCIAS ALEATORIAS 29
3.2.2 PROCESOS ESTACIONARIOS 30
3.2.3 ERGODICIDAD 30
3.2.4 RUIDO BLANCO 31
3.2.5 DENSIDAD DE POTENCIA ESPECTRAL 31
3.3 ESTIMACIÓN ESPECTRAL CONVENCIONAL 34
3.3.1 PROCEDIMIENTO DE BARTLETT 35
3.3.2 PROCEDIMIENTO DE BLACKMAN-TUKEY 35
3.3.3 PROCEDIMIENTO DE WELCH 36
3.3.4 UTILIZACIÓN DE LA TRANSFORMADA RÁPIDA DE FOURIER 36
3.3.5 VENTAJAS Y DESVENTAJAS DE LA ESTIMACIÓN ESPECTRAL CLÁSICA 36
3.4 ESTIMACIÓN ESPECTRAL PARAMÉTRICA 37
3.5 MÉTODOS PARAMÉTRICOS AUTORREGRESIVOS 41
3.5.1 MÉTODO DE YULE-WALKER O DE AUTOCORRELACIÓN 42
3.5.2 MÉTODO DE COVARIANCIA 44
3.5.3 MÉTODO DE COVARIANCIA MODIFICADA 45
3.6 REFERENCIAS 46
4 NODO DE PROCESAMIENTO HETEROGÉNEO 50
4.1 INTRODUCCIÓN 50
4.2 PROCESADOR DIGITAL DE SEÑALES, FAMILIA TMS320 50
4.3 DISEÑO DEL NODO DE PROCESAMIENTO HETEROGÉNEO 54
4.3.1 PROCESADORES 54

4.3.2 INTERFAZ POR MEMORIA COMPARTIDA 55
4.3.3 INTERFAZ POR CANAL DE COMUNICACIÓN 56
4.3.4 TEORÍA DE OPERACIÓN DE LA INTERFAZ DEL NODO DE PROCESAMIENTO HETEROGÉNEO 58
4.4 DESARROLLO DE LOS PROGRAMAS DE COMUNICACIÓN 59
4.5 CAMBIO DE FORMATO IEEE-TMS320 60
4.5.1 FORMATO ANSI IEEE 754-1985 61
4.5.2 FORMATO TMS320C30 62
4.6 REFERENCIAS 63

5 IMPLEMENTACIÓN DEL ESTIMADOR ESPECTRAL PARAMÉTRICO DE COVARIANCIA MODIFICADA 65

5.1 MÉTODO DE COVARIANCIA MODIFICADA 65
5.2 IMPLEMENTACIÓN PARALELA DEL ESTIMADOR ESPECTRAL PARAMÉTRICO DE COVARIANCIA MODIFICADA 69
5.2.1 PROCESAMIENTO DE UN SÓLO SEGMENTO DE DATOS 70
5.2.2 PROCESAMIENTO DE VARIOS SEGMENTOS DE DATOS 71
5.2.3 IMPLEMENTACIÓN HOMOGÉNEA UTILIZANDO UNA TOPOLOGÍA PIPELINE 72
5.2.4 IMPLEMENTACIÓN HETEROGÉNEA UTILIZANDO UNA TOPOLOGÍA PIPELINE 76
5.3 REFERENCIAS 80

6 ANÁLISIS DE RESULTADOS 84

6.1 SPEED-UP, EFICIENCIA Y FRACCIÓN SERIAL 84
6.2 RESOLUCIÓN 87
6.3 TIEMPOS DE PROCESAMIENTO 89
6.4 DESEMPEÑO 94
6.5 REFERENCIAS 101

7 CONCLUSIONES GENERALES 103

7.1 CONCLUSIONES 103

7.2 REFERENCIAS 104

APÉNDICE A

APÉNDICE B

APÉNDICE C

CAPÍTULO 1

CAPÍTULO 1

INTRODUCCIÓN

1.1 INTRODUCCIÓN GENERAL

El cálculo de la Densidad de Potencia Espectral (PSD) de procesos estocásticos, está normalmente basado en métodos que utilizan la Transformada Rápida de Fourier (FFT) [2]; estos métodos presentan ciertas limitaciones en su desempeño, tales como la resolución en frecuencia que distorsiona el espectro de la respuesta [2][6].

Los avances en el campo de Estimación Espectral han determinado que utilizando métodos de Estimación Espectral Paramétrica se pueden lograr mejoras significativas en la resolución del espectro en frecuencia, sin embargo, la utilización de estos métodos requiere un cálculo computacional más intensivo [5][3].

Algunos trabajos previos han estudiado el desempeño de distintas implementaciones de algoritmos de Estimación Espectral en arquitecturas homogéneas utilizando procesadores tipo Transputer [5][1]. En particular, el método Paramétrico de Covariancia Modificada, fue identificado como costo-efectivo en términos de su complejidad computacional [4][7].

Los Transputers son procesadores que pueden computar y coordinar eficientemente operaciones paralelas de tareas *irregulares*, en particular cuando éstas son consideradas de granularidad media [2]. Sin embargo, las demandas especiales de algunas operaciones de procesamiento de señales de granularidad fina asociadas a los métodos paramétricos, han revelado inconvenientes en la utilización de este tipo de arquitectura [2][7]. Recientes investigaciones han mostrado la posibilidad de integrar Transputers y DSPs con el fin de construir una arquitectura heterogénea para aplicaciones de procesamiento de señales [6].

El Departamento de Electrónica y Automatización del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas de la UNAM, cuenta con un grupo de investigación enfocado al desarrollo y evaluación de nuevas alternativas para el procesamiento de señales Doppler de ultrasonido, utilizando algoritmos de estimación espectral paramétricos en

distintas arquitecturas de procesamiento paralelo. El propósito de este trabajo de tesis es el diseño y evaluación de un nuevo esquema de procesamiento, utilizando una arquitectura heterogénea compuesta por una red de procesadores Transputer T805 y un DSP TMS320C30, en una topología pipeline, y haciendo una nueva partición del algoritmo de covariancia modificada; esto con el fin de reducir los tiempos de ejecución obtenidos al utilizar inicialmente una arquitectura homogénea basada solamente en transputers, y poder obtener un análisis espectral cualitativo de la señal en tiempo real.

1.2 OBJETIVOS

El objetivo principal de este trabajo es el diseño, desarrollo e implementación de un esquema de estimación espectral de señales, utilizando una arquitectura heterogénea de procesamiento paralelo integrada por transputers y un procesador digital de señales.

Tres objetivos particulares se desprenden del objetivo principal. Estos son:

- Implementar un algoritmo de estimación espectral paramétrico, basado en el método de covariancia modificada, en una arquitectura heterogénea de procesamiento paralelo.
- Evaluar el desempeño del esquema de procesamiento heterogéneo, comparándolo con un esquema homogéneo basado solamente en transputers.
- Aplicar el esquema heterogéneo propuesto al caso de procesamiento paralelo de señales Doppler de ultrasonido.

1.3 CONTENIDO DE LA TESIS

El presente trabajo de tesis se dividió en siete capítulos. En los primeros cuatro capítulos se definen los conceptos básicos que sirven de antecedentes para conocer y comprender las distintas alternativas de diseño con las que se contaba. En los capítulos quinto y sexto se describe ampliamente el esquema heterogéneo propuesto, su implementación y evaluación, haciendo una comparación con su contraparte homogénea.

En el capítulo siete, se dan a conocer las conclusiones generales del trabajo de investigación y se hacen algunas propuestas para un trabajo futuro en el campo de la estimación espectral paramétrica.

Capítulo 1. En este capítulo se presenta una introducción general al trabajo de investigación, dando a conocer la motivación, los objetivos y el contenido por capítulo de la tesis.

Capítulo 2. En el segundo capítulo se definen los conceptos de paralelismo y concurrencia; también se hace referencia a la familia de procesadores INMOS Transputer y a la familia de procesadores digitales de señales (DSP) de Texas Instruments TMS320, mostrando las características principales, tanto internas como externas, que permiten su interconexión para formar redes de procesadores utilizando distintas configuraciones.

Capítulo 3. En este capítulo se muestran las alternativas de diseño del nodo de procesamiento heterogéneo, así como sus características más importantes a nivel de software y de hardware.

Capítulo 4. Se presentan algunos antecedentes de estimación espectral y se define el concepto de densidad de potencia espectral (PSD); se muestran los métodos convencionales basados en el análisis de Fourier, así como los métodos paramétricos, mencionando sus ventajas y desventajas con respecto a los métodos convencionales y, por último, se muestran algunos métodos paramétricos autorregresivos (AR) incluyendo el método de covariancia modificada, que ha sido implementado en este trabajo.

Capítulo 5. Se presentan de manera general los esquemas de implementación del método de covariancia modificada desarrollados en trabajos previos de investigación, basados en una arquitectura homogénea. Se describe el esquema heterogéneo que se propone implementar, mostrando los resultados obtenidos al aplicarlo a señales Doppler de ultrasonido.

Capítulo 6. Se definen algunos conceptos importantes para medir el desempeño del esquema implementado; se hace una evaluación de los resultados obtenidos, mencionando las ventajas y desventajas que ofrece esta nueva aproximación al compararla con los esquemas presentados en el capítulo 5.

Capítulo 7. En el último capítulo se mencionan las conclusiones generales del trabajo de tesis y se proponen algunas alternativas para el procesamiento de señales, que podrían incrementar el desempeño de los esquemas presentados.

1.4 REFERENCIAS

[1]GARCIA NOCETTI, D. F., SOLANO GONZÁLEZ, J., BENÍTEZ PÉREZ, H., Parallel Implementation of Parametric Spectral Estimation for Doppler Blood Flow Instrumentation, The Proceedings of the 6th International Conference on Signal Processing Applications & Technology, Vol I, pp. 613-617, Boston, Massachusetts, USA, 1995.

[2]GARCIA NOCETTI, D. F., SOLANO GONZÁLEZ J., MARTÍNEZ FLORES, J., RAMOS HERNÁNDEZ, D., Heterogeneous Parallel Architecture for Improving Signal Processing Performance in Doppler Blood Flow Instrumentation, 10th International Parallel Processing Symposium, Honolulu Hawaii, USA, 1996.

[3] KAY, Steven M., Modern Spectral Estimation Theory & Application, Prentice-Hall Signal Processing Series, Alan V. Oppenheim, Series Editor, USA, 1988.

[4]RUANO, M. G., Investigation of Real-Time Spectral Analysis Techniques for Use with Pulsed Ultrasonic Doppler Bood-Flow Detectors, Thesis submited to the University College of North Wales, Bangor, U. K., 1992.

[5]RUANO, M. G., GARCÍA NOCETTI, D. F., FISH, P. J., FLEMING, P. J., Alternative parallel implementation of an AR-modified covariance spectral estimator for diagnostic ultrasonic blood flow studies, Parallel Computing 19, pp. 463-476, North-Holland, 1993.

[6]SOLANO GONZÁLEZ, J., GARCÍA NOCETTI, D. F., RODRÍGUEZ VÁZQUEZ, K., Parallel Genetic Algorithms in Spectral Estimation of Doppler Signals, 3rd IFAC Workshop on Algorithms and Architectures for Real-Time Control, Ostende, Bélgica, 1995.

[7]SOLANO GONZÁLEZ, J., GARCÍA NOCETTI, D. F., RODRÍGUEZ VÁZQUEZ, K., RAMOS HERNÁNDEZ, D., Parallel Genetic Algorithms in Autorregressive Modelling Using A Heterogeneous Architecture, 13th IFAC World Congress, San Francisco, USA, 1996.

CAPÍTULO 2

CAPÍTULO 2

ANTECEDENTES DE PROCESAMIENTO PARALELO

2.1 INTRODUCCIÓN

La arquitectura clásica de cualquier computadora en la actualidad, se basa en la idea de Von Neumann, de que un programa de computadora puede ser almacenado en memoria, ejecutado por una unidad de procesamiento, donde la entrada y salida de datos se realiza a través de un subsistema único. Esta arquitectura básica se encuentra presente en cualquier computadora personal, en algunos casos con pequeñas variaciones. El elemento que interconecta la memoria, la unidad de procesamiento y el subsistema de E/S, es conocido como bus; una pieza de hardware capaz de reconocer en todo momento la unidad a la cual deben ser enviados los datos [6]. Conforme los distintos elementos que componen la computadora se vuelven más potentes y trabajan a mayor velocidad, se presenta lo que se conoce como cuello de botella, debido a la competencia por recursos que se da entre las distintas partes. Dentro de las soluciones típicas para este tipo de problema, está el interponer una memoria cache a la memoria principal, con la finalidad de obtener mayor velocidad de acceso al código y los datos más utilizados; otra de las soluciones para eliminar o reducir el cuello de botella de Von Neumann es realizar un pipeline de las instrucciones, simultáneamente con una vectorización de operaciones en la unidad central de procesamiento [6].

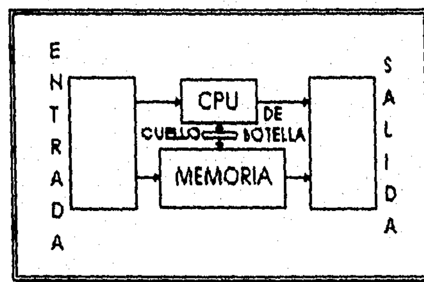


figura 2.1.1
Diseño de Von Neumann

La memoria cache permite cargar y almacenar datos e instrucciones a una mayor velocidad. Normalmente es de menor capacidad que la memoria dinámica, y debido a esto, se deben establecer prioridades sobre los datos que deben mantenerse en memoria, identificando los elementos mayormente utilizados por el sistema [6].

El pipeline de instrucciones consiste en subdividir las instrucciones complejas en otras más simples, enfocando distintas partes del hardware para llevar a cabo cada operación en una secuencia predefinida. Debido a esto, cuando se ejecutan varias operaciones similares, el procesador en pipeline opera como un procesador paralelo después de un periodo de inicialización donde son cargadas las operaciones más simples. Como un ejemplo se puede considerar que una operación compleja está compuesta de las siguientes etapas: tomar el código de operación de la memoria, decodificarla, generar la dirección de datos, tomar de memoria el operando, ejecutar la operación, almacenar el operando, actualizar el contador de programa. Todas estas etapas pueden ejecutarse en pipeline utilizando dispositivos específicos para cada una de ellas. A partir de la idea de ejecutar en pipeline las distintas etapas de cada instrucción surge la arquitectura RISC, donde un conjunto reducido de instrucciones simples son ejecutadas de manera rápida; y las operaciones complejas son generadas a partir de estas instrucciones simples [6][8].

La vectorización es otra solución para el cuello de botella de Von Neumann, ya que se incrementa la velocidad de cómputo. Operaciones típicas, como sumar dos vectores, pueden llevarse a cabo en paralelo utilizando hardware específico. Normalmente, las unidades de procesamiento se añaden al sistema tradicional, para ejecutar operaciones de punto flotante y operaciones con enteros en forma de vector [7][8].

Estas variantes de la arquitectura propuesta por Von Neumann, coinciden en la adición de elementos de procesamiento en todos los casos, originando un tipo particular de multiprocesamiento. Sin embargo, todas estas adiciones, carecen de flexibilidad, y el usuario está limitado en la búsqueda de soluciones para algunas arquitecturas en particular [6].

Para evitar los problemas que presenta la arquitectura propuesta por Von Neumann, es necesario hacer algunas modificaciones al diseño original: primero, utilizar varias unidades de procesamiento, así como también un cierto número de elementos de memoria; segundo, incrementar el número de unidades de control, con la finalidad de

obtener varios canales de comunicación y de entrada/salida de datos e instrucciones; y finalmente, colocar todos estos elementos en una red interconectados entre sí. Esta nueva arquitectura es conocida como *non-von* debido a que no sigue el diseño original de Von Neumann [8].

Cuando se logra la interconexión de todos los elementos de una arquitectura *non-von*, es posible, con un conjunto de N procesadores, cada uno de ellos procesando a una velocidad máxima de T instrucciones por segundo, lograr una velocidad de computo de (N*T-V) instrucciones por segundo. Donde V representa un cierto costo de *overhead* debido a la arquitectura de cada procesador y a la comunicación entre estos. De esta manera, aun cuando la velocidad de procesamiento de cada procesador, visto de manera individual, sea relativamente baja, la velocidad máxima de procesamiento puede ser comparada con la de una supercomputadora, pero a un costo menor. Entonces, es posible definir a una computadora paralela, como una interconexión de procesadores, con unidades de memoria independientes, que trabajan en conjunto sobre un problema en particular [8].

2.2 CLASIFICACIÓN

Las computadoras y sistemas de cómputo pueden clasificarse de varias maneras, una de ellas es de acuerdo con la estructura de procesamiento que manejan. Flynn divide de esta manera a las computadoras en cuatro grupos: SISD, SIMD, MISD y MIMD [2][5].

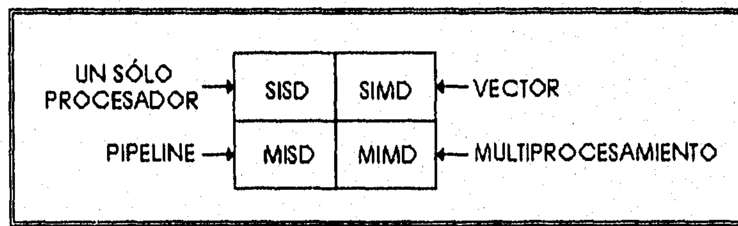


figura 2.2.1
Taxonomía de Flynn

2.2.1 SISD (SIMPLE INSTRUCTION, SIMPLE DATA)

Es en esta clasificación, donde se consideran las computadoras que siguen el diseño original de John Von Neumann, las cuales cuentan con un sólo procesador y una unidad de memoria [5].

En este modelo, se toma una sola instrucción del programa a la vez, y ésta, únicamente opera sobre un bloque de datos. Aun cuando los sistemas operativos multi-tareas dan a este tipo de arquitectura la idea de concurrencia, no es posible alcanzar un paralelismo verdadero [8].

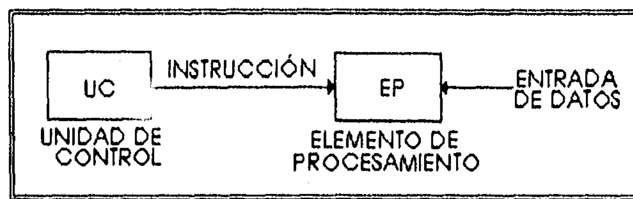


figura 2.2.2
Arquitectura SISD

2.2.2 SIMD (SINGLE INSTRUCTION, MULTIPLE DATA)

En este tipo de arquitectura de procesamiento paralelo, una sóla unidad de control, manda instrucciones a cada elemento de procesamiento[5][7]. En la figura 2.2.3 se observa este tipo de arquitectura.

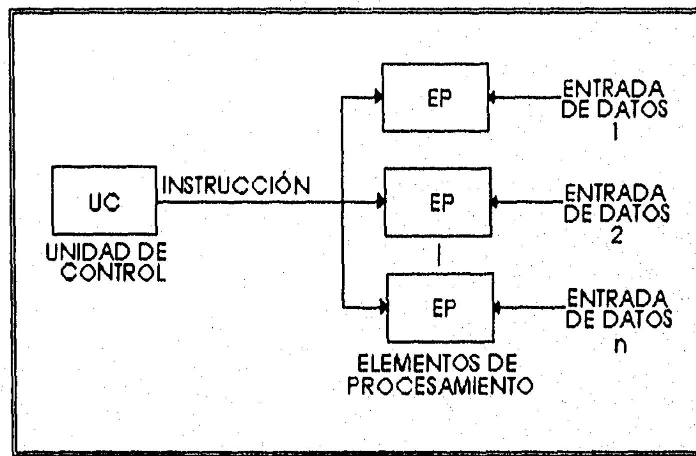


figura 2.2.3
Arquitectura SIMD

En una computadora paralela SIMD la misma instrucción se ejecuta de manera síncrona por todas las unidades de procesamiento. Un arreglo de procesadores (vector) es un ejemplo de este tipo de arquitectura, y es de gran utilidad cuando se implementan algoritmos regulares que involucran operaciones matriciales [5].

Una característica importante de esta clasificación, es que las computadoras SIMD, no pueden ejecutar diferentes instrucciones en el mismo ciclo de reloj, por lo que este tipo de arquitectura es la más adecuada para problemas que involucren paralelismo de datos, y no de código [7].

2.2.3 MISD (MULTIPLE INSTRUCTION, SINGLE DATA)

En esta arquitectura se encuentran varios procesadores trabajando de manera simultánea y ejecutando instrucciones distintas sobre el mismo conjunto de datos. Un ejemplo claro es un sistema paralelo trabajando en pipeline [5]. Cuando se trabaja con procesamiento de señales utilizando técnicas de procesamiento paralelo, este tipo de arquitectura es muy utilizado. En la figura 2.2.4 se presenta el esquema MISD, donde se observa que existe una unidad de control por cada elemento de procesamiento, y una sóla entrada de datos global[5][3].

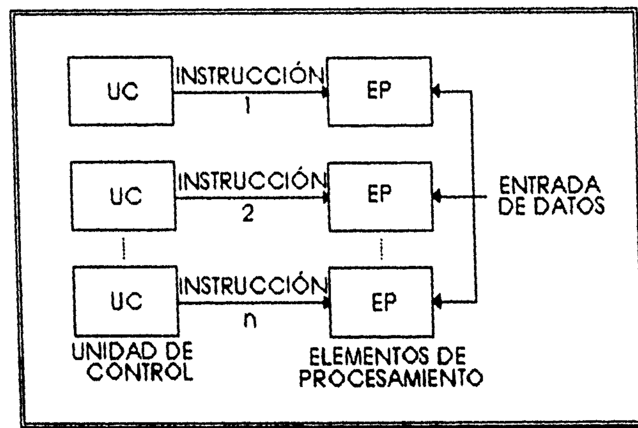


figura 2.2.4
Arquitectura MISD

2.2.4 MIMD (MULTIPLE INSTRUCTION, MULTIPLE DATA)

Este tipo de arquitectura, en comparación con la SIMD, presenta una estructura más general, además de que trabaja de manera asíncrona. Cada elemento de procesamiento ejecuta su propio programa con una unidad de control individual [5][7].

Al igual que la arquitectura SIMD, esta arquitectura es muy utilizada en programas que involucran paralelización de datos, pero en este caso, es posible hacer uso de condicionales debido a que existe una unidad de control por cada elemento de procesamiento [7].

Las máquinas MIMD, no necesitan conectarse de forma específica. En la siguiente sección se muestran algunas de las topologías más utilizadas para la interconexión de elementos de procesamiento.

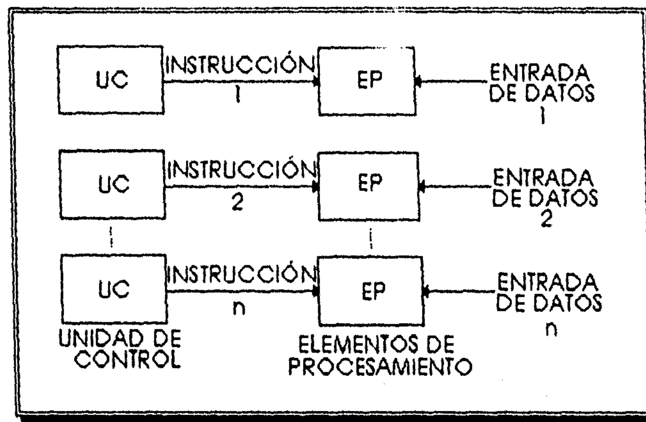


figura 2.2.5
Arquitectura MIMD

2.3 CLASIFICACIÓN DE TOPOLOGÍAS

En procesamiento paralelo, una topología de interconexión de redes, es una función de mapeo de los procesadores y memorias que intervienen en la red, sobre el mismo conjunto de procesadores y memorias. La topología describe como se conectan entre sí los distintos elementos de procesamiento de la red. Por ejemplo, en una topología totalmente conectada, cada procesador es mapeado a cada uno de los elementos de procesamiento restantes. Una topología de anillo es donde cada procesador *K*, está conectado únicamente con los procesadores (*k*+1) y (*k*-1) [8].

2.3.1 RED DE PROCESADORES TOTALMENTE CONECTADA

En este tipo de redes, cada procesador tiene comunicación directa con los procesadores restantes, y es ideal cuando se requiere transmitir mensajes de un procesador a otro en un sólo paso [7], tal y como se

muestra en la figura 2.3.1, donde se observan ocho procesadores totalmente conectados entre sí.

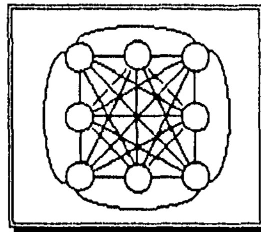


figura 2.3.1
Topología totalmente conectada

2.3.2 TOPOLOGÍA DE ESTRELLA

En la red de procesadores tipo estrella, existe un procesador central conectado a todos y cada uno de los procesadores restantes. La comunicación entre cualquier par de procesadores, se realiza a través del procesador central. Esto último trae como consecuencia, en algunas ocasiones, un cuello de botella en el procesador central, debido a la competencia por los recursos [7][2].

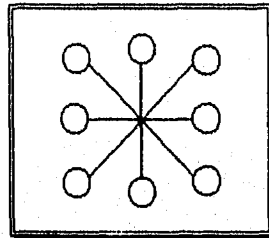


figura 2.3.3
Topología de estrella

2.3.3 TOPOLOGÍA DE BUS (PIPELINE)

Esta topología es la forma más simple de interconexión en una red de procesadores. Cada elemento de procesamiento (excepto el último), se comunica con el siguiente procesador, a la izquierda o derecha, por medio de un canal directo de comunicación. Este tipo de topología es muy utilizada en procesamiento de señales, voz e imágenes; debido a la posibilidad de trabajar en pipeline sobre varios segmentos de datos a la vez (figura 2.3.4) [7].

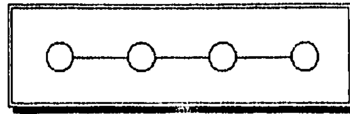


figura 2.3.4
Topología de bus

2.3.4 TOPOLOGÍA DE ANILLO

En algunas ocasiones, cuando se cuenta con una topología de bus, se conecta el último procesador de la red, con el primero formando una especie de anillo. De esta manera, la transmisión de mensajes entre procesadores, se realiza en secuencia, de forma inmediata con el procesador siguiente, hasta alcanzar su destino [7][2].

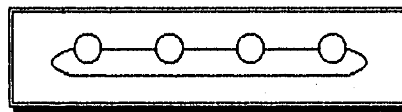


figura 2.3.5
Topología de anillo

2.3.5 TOPOLOGÍA DE MALLA

La topología de malla, es una extensión de la topología de bus en dos dimensiones. En este tipo de redes, cada procesador tiene comunicación directa con cuatro procesadores, a excepción del perímetro de la malla, donde solamente existe comunicación con tres de ellos. En la figura 2.3.6, se muestra una malla de 16 procesadores. Del mismo modo, la topología de anillo, puede extenderse a dos dimensiones como se muestra en la figura 2.3.7 [7].

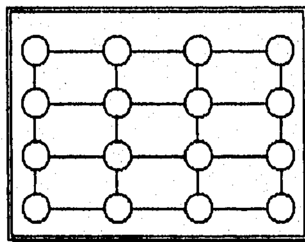


figura 2.3.6
Topología de malla como
extensión de una topología de bus

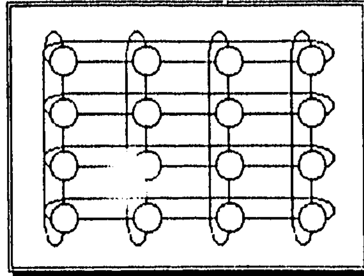


figura 2.3.7
Topología de malla como
extensión de una topología de anillo

2.3.6 TOPOLOGÍA DE ÁRBOL

En la figura 2.3.8 se muestra una topología de árbol. En ésta, se tiene un procesador conectado a uno o más elementos de procesamiento; donde cada uno de éstos, se comunica de la misma manera con sus descendientes. Las topologías de estrella y de bus, son casos especiales de este tipo de topología [7][2].

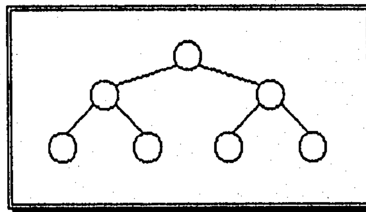


figura 2.3.8
Topología de árbol

2.4 FAMILIA DE PROCESADORES INMOS TRANSPUTER

La principal limitante de la arquitectura de Von Neumann es la existencia de un bus único de datos entre el procesador central y la memoria. Debido a esto, los intentos de extender la arquitectura de Von Neumann a sistemas de multiprocesamiento se han realizado sin mucho éxito. Por ello se ha desarrollado la familia de procesadores Transputer de 16 Y 32 bits, explícitamente diseñada para soportar procesamiento paralelo en aplicaciones de tiempo real, tanto a nivel de software como a nivel de hardware. [4][5].

El Transputer (contracción de las palabras TRANsistor-compuTER) es un microprocesador en un sólo chip desarrollado por INMOS Ltd, disponible desde 1985, fecha en la que se presentó el primer modelo IMS T414. Estos procesadores, fueron los primeros diseñados específicamente para soportar sistemas de procesamiento paralelo. Dentro de las características más sobresalientes que presenta su diseño, está la existencia de hardware específico para ejecutar de manera concurrente distintos procesos en un sólo procesador; links bidireccionales de comunicación, con el fin de conectar dos transputers; acceso directo a memoria; memoria interna de gran capacidad. Todas estas características contribuyen eficientemente a la implementación de tareas de procesamiento paralelo [3].

El Transputer es un procesador que explota al máximo la tecnología VLSI (*very large scale of integration*). Una característica importante de este tipo de tecnología es la comunicación con dispositivos externos, que por lo general es más lenta que la comunicación interna. Por otra parte, dentro de una computadora, toda operación que realiza el procesador central, involucra la utilización de memoria externa, debido a esto, el transputer incluye ambos elementos (memoria y procesador central) dentro del mismo circuito integrado (*on-chip memory*) [12][13].

Sin embargo, la velocidad de comunicación entre dispositivos electrónicos se ve disminuida cuando varios elementos compiten por la utilización de un bus compartido. Por esto, el Transputer cuenta con links de comunicación serial punto a punto, para comunicar de manera directa dos procesadores, lo cual permite la construcción de sistemas de multiprocesamiento a un costo relativamente bajo [12][13].

La influencia de la tecnología RISC en el transputer, se refleja en un conjunto reducido de instrucciones, permitiendo una mayor velocidad de procesamiento cuando se ejecutan instrucciones simples [10][12].

En 1985, se introdujo el primer transputer de 32 bits, el IMS T414, con memoria interna de 2 kbytes de alta velocidad, interfaz de memoria externa de 32 bits y cuatro links de comunicación. Este procesador trabaja de 15 a 20 MHz, y puede realizar hasta 10 MIPS en programas secuenciales. Posteriormente, se introdujo un procesador de 16 bits, el IMS T212, muy parecido en diseño al T414, pero con una longitud de palabra e interfaz de memoria externa de 16 bits [11][13].

En 1987 se presentaron mejoras al diseño del T414, y se introdujo el T800, cuya característica principal es el hecho de que puede incrementar

la eficiencia de los sistemas de computo en paralelo debido a que cuenta con una unidad de pur ɔ flotante que maneja hasta 1.5 MFLOPS. La adición de la UPF normalmente requiere el doble de espacio del procesador, pero en este caso, el T800 es solamente mayor en tamaño al T414 en un 20%; además trabaja a una velocidad de 20, 25 o hasta 30 MHz [13], lo que constituye una ventaja adicional.

Otro miembro de la familia INMOS Transputer disponible actualmente es el T805. Éste cuenta con 4 kbytes de memoria interna, unidad de punto flotante capaz de manejar hasta 4.3 MFLOPS; una longitud de palabra e interfaz de memoria de 32 bits y cuatro canales de comunicación que trabajan a 5, 10 ó 20 Mbits por segundo [10][13].

La familia Transputer de INMOS cubre procesadores de 16 bits (serie T2XX) y de 32 bits (serie T4XX y T8XX). Las series de 32 bits difieren entre sí por algunas características: la memoria interna de la T4XX es de 2 Kbytes, en contraste con la de la serie T8XX que es de 4 Kbytes; la primera cuenta con sólo dos links de comunicación, mientras que la segunda incluye cuatro; sin embargo, la diferencia más significativa es que la serie T8XX incluye una unidad de punto flotante, lo cual aumenta la velocidad y eficiencia de los sistemas de cómputo paralelos[13].

Aun cuando existen estas diferencias entre las series T4XX y T8XX, ambos circuitos son compatibles pin a pin; además se pueden formar configuraciones mixtas utilizando los links de comunicación de ambos chips [12].

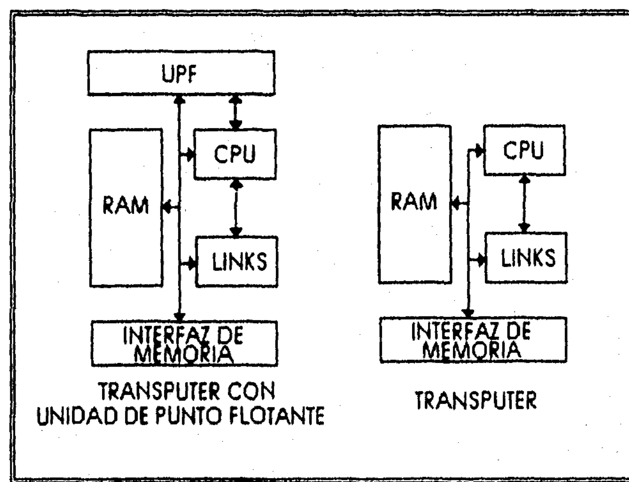


figura 2.4.1
Diagrama de interconexión
del transputer

La última generación de transputers de INMOS, es el T9000, diseñado en 1994. Éste cuenta con un procesador de 32 bits, con una unidad de punto flotante de 64 bits; además de una memoria cache de 16 kbytes. Incluye cuatro links de comunicación bidireccionales y un VCP[1] (*virtual channel processor*) que permite comunicar eficientemente dos procesadores T9000. Todos estos componentes en un mismo circuito integrado. Las ventajas principales que presenta esta nueva serie de procesadores, son: aumento de velocidad de reloj a 50 MHz, memoria cache interna, y canales virtuales de comunicación punto a punto[10][11].

2.4.1 CANALES DE COMUNICACIÓN PUNTO A PUNTO

La arquitectura del transputer simplifica el diseño de sistemas de multiprocesamiento, debido a que posee links de comunicación punto a punto. Todos los modelos del transputer poseen uno o más links estándares de comunicación, los cuales pueden utilizarse para hacer una conexión directa con otros componentes. Esto permite la construcción de redes de transputers de tamaño y topología variable [10].

Existen algunas ventajas de la comunicación punto a punto sobre la utilización de buses de datos: aun cuando el sistema involucre un gran número de procesadores, el mecanismo de comunicación no se verá afectado en velocidad o desempeño al agregar procesadores; de igual forma, el ancho de banda de los canales de comunicación no se satura al aumentar el número de transputers en la red.

Para lograr que exista comunicación sincronizada entre procesadores es necesario que por cada mensaje enviado, se mande una señal de reconocimiento indicando que se recibió el mensaje sin errores. Consecuentemente, es necesario que cada link sea bidireccional [10].

Para comunicar dos transputers vía canal de comunicación punto a punto es necesario conectar la interfaz del link del primer transputer a la del segundo, utilizando dos lineas unidireccionales a través de las cuales los datos se transmiten en forma serial. Cada vez que se transmite información de esta manera, la señal lleva consigo información de control además del mensaje; esto con la finalidad de cumplir con un protocolo de comunicación, lo cual permite que se lleve a cabo la comunicación entre

[1]Un link virtual que realiza una conexión lógica entre dos procesadores, y se mapea a un link físico del procesador.

procesadores con distinta longitud de palabra, por ejemplo un T800 y un T212 [12][13].

Cada mensaje se transmite como una secuencia de bytes, por lo que se requiere un buffer de un byte de longitud en el transputer receptor, para asegurar que el mensaje haya sido transmitido correctamente. Cada byte que se transmite, consta de dos bits de inicialización, ocho bits de mensaje y un bit indicando el fin de la transmisión (figura 2.4.2). Después de transmitir, el transputer transmisor espera una señal de reconocimiento por parte del transputer receptor, la cual consiste en un bit de inicio en alto y un bit en cero, indicando que la información fue recibida y que el link está libre para iniciar otra transmisión. El link transmisor retoma el resto del mensaje, hasta que la señal de reconocimiento es recibida [12].

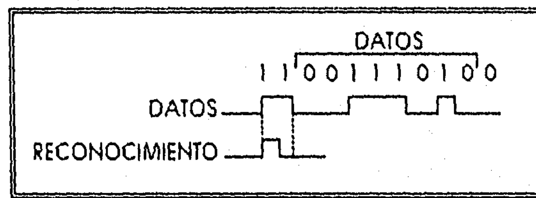


figura 2.4.2
Protocolo de transmisión

Todos los transputers soportan una frecuencia de comunicación de almenos 10 Mbit/segundo; sin embargo, algunos modelos soportan frecuencias mayores. Al mantener una frecuencia de comunicación estándar, es posible la construcción de sistemas de tamaño y desempeño mixto [12][13].

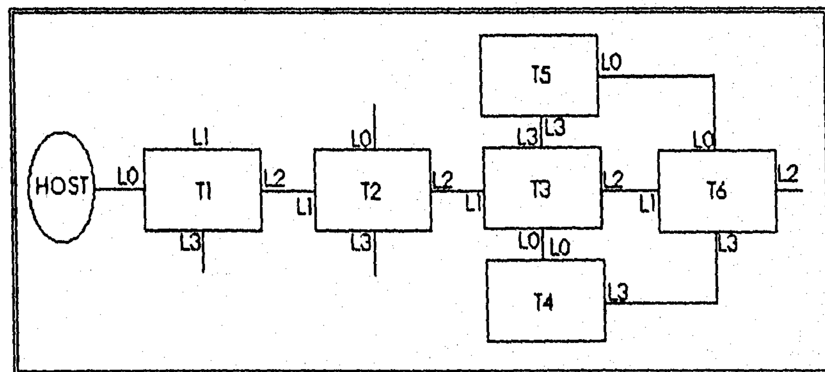


figura 2.4.3
Ejemplo de una red de procesadores utilizando los links de comunicación

En particular el IMS T805 cuenta con cuatro links bidireccionales de comunicación, que soportan el estándar de INMOS de 10 Mbits/segundo; pero, adicionalmente, pueden trabajar a 5 ó 20 Mbits/segundo para procesadores que trabajan a 20 ó 25 MHz, y 20 Mbits/segundo para dispositivos más veloces. En la figura 2.4.3 se muestra un ejemplo de interconexión de varios procesadores utilizando los links de comunicación externa [10][12][13].

2.5 PROGRAMACIÓN PARALELA

Existen varios lenguajes de programación secuencial, cuya principal característica es el orden estricto en el que se ejecutan las distintas partes que componen el programa. A diferencia de la programación paralela, el tiempo total de procesamiento corresponde a la suma del tiempo de ejecución de cada parte del programa. Esto se debe a que los lenguajes de programación secuencial nacieron junto con las computadoras que seguían el modelo de Von Neumannn que, como se mencionó anteriormente, cuenta con una limitante por la competencia de recursos [10][9].

El paralelismo, surge como una necesidad para algunas aplicaciones que requieren procesamiento en tiempo real, e involucra varios aspectos importantes tales como concurrencia, comunicación entre procesos, sincronización, nivel de paralelismo, granularidad e implementación paralela.

La programación concurrente es el nombre con el que se conocen las técnicas de programación para expresar el potencial del paralelismo y resolver problemas de sincronización y comunicación. Un programa concurrente es una colección de procesos secuenciales autónomos ejecutándose lógicamente en paralelo[10].

La comunicación entre procesos es un aspecto importante en la programación paralela. Dentro de un programa se pueden definir varios procesos que deben ejecutarse en forma paralela. Estos, en la mayoría de los casos, no pueden trabajar de manera aislada, sino que requieren comunicarse entre sí por medio de canales de comunicación como se verá posteriormente. Debido a esta necesidad inherente a la programación paralela, es necesario que exista una sincronización entre

procesos para evitar que exista lo que se conoce como un *deadlock*. La sincronización entre procesos se clasifica de la siguiente manera: asíncrono, síncrono y de invocación remota. En el modelo asíncrono, se envían datos de un proceso a otro pero no se lleva a cabo verificación alguna de que se hayan recibido. En el segundo, el proceso que envía el mensaje, no continúa trabajando hasta que el proceso receptor recibe los datos de manera correcta y completos. En el modelo de invocación remota, el emisor continúa cuando el receptor le contesta [9][10].

El nivel de paralelismo soportado puede ser de dos tipos: anidado y básico. En el primero, los procesos son definidos en cualquier nivel del texto del programa e incluso es posible definir procesos que se ejecuten internamente, es decir, dentro de otros procesos. En el segundo, los procesos son definidos fuera del texto del programa [10].

La granularidad a nivel software, se refiere al número de procesos contenidos dentro de un programa. Si un programa contiene varios procesos simples, se dice que es de granularidad fina; por el contrario, si sólo existen unos cuantos procesos dentro del programa, pero la correcta ejecución de cada uno de ellos es necesaria para la inicialización y terminación del programa global, se dice que es un programa concurrente de granularidad gruesa [10].

Con el surgimiento de procesadores diseñados específicamente para soportar paralelismo y concurrencia, fue necesario que se desarrollaran lenguajes de programación que explotaran estas características. Algunos ejemplos son el lenguaje Occam, Ada, Pascal concurrente, Modula-2 y CSP. El lenguaje Occam en particular, proporciona un eficiente soporte para el desarrollo de sistemas largos y complejos, con librerías de concurrencia y protocolos de comunicación y sincronización de procesos [10].

2.5.1 OCCAM

Occam es un lenguaje de programación diseñado específicamente para soportar concurrencia y paralelismo. Por lo mismo es, en principio, el lenguaje ideal para aplicaciones científicas , de control de procesos y muchas otras áreas de la ingeniería que requieren procesamiento paralelo en tiempo real [4][5].

La base del diseño del lenguaje Occam es el modelo CSP (Communicating Sequential Processes) . Éste es un modelo generalizado de concurrencia basado en la idea de ejecución de procesos de manera independiente, intercambiando datos unos con otros a través de canales de comunicación. De la misma manera, Occam es un lenguaje que permite la ejecución de procesos independientes y, haciendo uso de librerías de concurrencia, permite su ejecución en forma concurrente en un sólo procesador o en paralelo en varios procesadores. Un programa en Occam, que no es otra cosa que una colección de procesos, puede ser alojado en uno o varios procesadores transputer. En el primer caso, cuando se ejecuta un programa en un sólo procesador, se realiza lo que se conoce como pseudo-paralelismo, queriendo decir con esto que debido al diseño del hardware del procesador, los procesos comparten el tiempo del procesador ocurriendo un cierto traslape entre estos, dando como resultado una ejecución rápida y confiable. Cuando un programa se distribuye en varios procesadores, los procesos que lo componen se ejecutan de manera paralela (en el caso de existir procesos internos, se ejecutarían de manera concurrente en el mismo procesador), explotando al máximo la arquitectura del transputer [4][5][10].

El lenguaje Occam, además de las múltiples herramientas que ofrece para elaborar programas concurrentes, cuenta con un lenguaje de configuración para mapear la estructura lógica de un programa paralelo o concurrente, en una red física de procesadores. En este lenguaje, se definen los canales de comunicación utilizando los links del transputer y se asignan a cada procesador los distintos procesos que componen el programa [9].

2.5.2 C PARALELO

Los lenguajes secuenciales pueden ser extendidos a arquitecturas de procesamiento paralelo. Tal es el caso del lenguaje C, que cuenta con extensiones de paralelismo en forma de librerías separadas, conteniendo nuevas funciones y operadores totalmente compatibles con el estándar ANSI C, donde el paralelismo es introducido en forma de funciones [10].

El lenguaje C paralelo, al igual que el lenguaje Occam, se basa en el modelo CSP para crear entidades llamadas procesos que se comunican entre sí por medio de canales de comunicación [1].

El conjunto de herramientas que ofrece el lenguaje C paralelo, incluye un lenguaje de configuración que permite describir de manera independiente el hardware y el software, para posteriormente unirlos por medio de un mapeo. De esta forma es posible formar redes de procesadores utilizando distintas topologías, donde se ejecuten procesos en paralelo y logrando la comunicación entre estos por medio de canales de comunicación declarados y mapeados a los links de cada transputer [1][14].

El lenguaje C, a pesar de no ser un lenguaje que naciera simultáneamente con los procesadores paralelos, permite explotar al máximo las ventajas que ofrece la arquitectura del transputer, ya que cuenta con distintas librerías de concurrencia y con un lenguaje de configuración, por lo que se presenta como una buena opción para el desarrollo de aplicaciones de procesamiento paralelo de señales en tiempo real [1].

2.6 REFERENCIAS

[1] ANSI C Toolset, User manual, INMOS, SGS-Thomson Microelectronics, 1993.

[2] BERTSEKAS, D., TSITSIKLIS, J., Parallel and Distributed Computation, Numerical Methods, Prentice Hall, USA, 1989.

[3] COCK, R., Parallel Programs for the Transputer, Prentice Hall, Englewood Cliffs, N. J., USA, 1991.

[4] GALLETLY, J., Occam 2, Pitman Publishing, Great Britain, 1990.

[5] GARCIA, F., FLEMING, P., Parallel Processing in Digital Control, Advances in Industrial Control, Springer-Verlag, Germany, 1992.

[6] HEIDRICH, D. and GROSSETIE, J. C., Computing with T.Node Parallel Architecture, Kluwer Academic Publishers for the Commission of the European Communities, Brussels and Luxemburg, 1991.

[7] KUMAR, V., GRAMA, A., GUPTA, A., KARYPIS, G., Introduction to Parallel Computing, Design and Analysis of Algorithms, The Benjamin/Cummings Publishing Company, USA, 1994.

[8] LEWIS G., EL-REWINI, H., Introduction to Parallel Computing, Prentice Hall International Editions, USA, 1992.

[9] Occam 2 Toolset, User manual, INMOS, SGS-Thomson Microelectronics, 1993.

[10] RAMOS, D., Diseño e Implementación de un Nodo Heterogéneo de Procesamiento Digital de Señales Utililando un Transputer y un DSP, Tesis de licenciatura, UNAM, México, 1995.

[11] The Transputer Databook, INMOS, SGS-Thomson Microelectronics, third edition, 1992.

[12] The Transputer Development and IQ Systems Databook, INMOS, SGS-Thomson Microelectronics, second edition, 1991.

[13] The Transputer Applications Notebook, Architecture and Software, INMOS, SGS-Thomson Microelectronics, first edition, 1989.

[14] THIÉBAUT, D., Parallel Programming in C for the Transputer, USA, 1995.

CAPÍTULO 3

CAPÍTULO 3
ANTECEDENTES DE ESTIMACIÓN ESPECTRAL

3.1 INTRODUCCIÓN: HISTORIA

El análisis espectral está relacionado con la caracterización del contenido en frecuencia de una señal; su campo de aplicación es muy variado, incluyendo el procesamiento de señales de ultrasonido, acústica, sismología, espectrometría, entre otros [8]. Existen varias técnicas para llevar a cabo este tipo de análisis, destacando la siguiente clasificación general: métodos no paramétricos o basados en análisis de Fourier y métodos paramétricos [8].

La historia de la estimación espectral tiene sus raíces en tiempos remotos, cuando en el año 600 A. C. el gran matemático griego Pitágoras con su trabajo sobre las leyes de la armonía de la música, fuera el primero en estudiar un fenómeno físico que involucrara análisis espectral [12].

En tiempos modernos, el que se considera como el primer científico en realizar estudios en el campo del análisis espectral es Isaac Newton, quien introdujo por primera vez el término científico *spectrum* utilizando la palabra en latín para una imagen. En inglés la palabra *spectral* corresponde al adjetivo de la palabra *specter* que quiere decir fantasma. En sentido estricto, el término "espectral" podría sustituirse por "fantasmal", sin embargo, actualmente se puede utilizar siempre que se refiera a algún fenómeno físico, o modifique de alguna manera a la palabra "estimación" [12].

En el siglo XVII, cuando la teoría del cálculo infinitesimal fue presentada por Newton y Leibnitz, la observación de los fenómenos naturales parecía indicar la existencia de una relación constante entre algunas variables físicas. Esto fue reforzado por la formulación de leyes dando explicación a estos fenómenos tomando como base ecuaciones diferenciales, siendo el ejemplo más claro las leyes de Newton. La idea de una función que se modificara de manera caprichosa o aleatoria no era considerada por ninguno de los grandes científicos de la época [12].

En 1807, el científico francés Jean Baptiste Joseph de Fourier propuso que cualquier señal de duración finita, aun cuando no se trate de una

señal continua, puede ser expresada como una sumatoria infinita de senos y cosenos [8]. Debido a que los científicos de la época pensaban que una combinación de senos y cosenos correspondía siempre a una función analítica continua en todo punto e infinitamente diferenciable, cuestionaron la validez del teorema de Fourier. Las series de Fourier representan un conjunto de funciones ortogonales, por lo que toda expansión en términos de series de senos y cosenos (o cualquier tipo de funciones ortogonales), es llamada serie de Fourier [12].

La primer aplicación importante de la expansión en series de Fourier fue realizada por Schuster en 1898, dando a conocer el primer método de análisis espectral numérico, al cual llamó método del Periodograma [12][8].

Este método consiste en obtener los valores de $P(e^{j\omega})$ a partir de una serie de muestras de una señal discreta de la siguiente manera:

$$P(e^{j\omega}) = \frac{1}{N}\left|\sum_{n=0}^{N-1} x[n]e^{-j\omega n}\right|^2 \qquad .....(3.1)$$

Si las muestras $x[n]$ corresponden a una señal senoidal con frecuencia ω_1, entonces el periodograma mostrará un pico en $\omega=\omega_1$, valor que corresponde a la frecuencia de la senoidal con mayor aportación a la señal. Sin embargo, cuando este método fue aplicado a otras series de tiempo, el periodograma se comportaba de manera errática sin mostrar algún pico dominante. A pesar de este problema, el periodograma permaneció como la única herramienta numérica para análisis espectral de señales discretas en el tiempo[8][12].

En 1930 Wiener, en su artículo *Generalized harmonic analysis* [18], estableció el primer método de análisis espectral de procesos aleatorios basado en la transformada de Fourier continua, definiendo la función de autocorrelación de señales aleatorias y mostrando su relación con la densidad de potencia espectral por medio de la transformada de Fourier. En 1958 R. B. Blackman y J. Tukey, basándose en el trabajo de Wiener, presentaron un nuevo método para obtener la densidad de potencia espectral al computar la función de autocorrelación de la secuencia de datos, para posteriormente aplicar la transformada de Fourier a la función resultante [1][8].

La siguiente aportación importante en análisis espectral, fue el algoritmo de la transformada rápida de Fourier (FFT) propuesto por Cooley

y Tukey en 1965 [4]. Con el desarrollo de esta nueva forma de computar la transformada de Fourier, resurgió el interés en el campo de la estimación espectral e inc uso, convirtió a la versión modificada del algoritmo de Schuster en el método de análisis espectral más popular de la actualidad [8].

Debido al comportamiento errático del periodograma de Schuster cuando se aplicaba a secuencias aleatorias, en 1965 Yule propuso utilizar un modelo paramétrico para el análisis de este tipo de señales [19]. El trabajo de Yule se basó en la discretización del movimiento de un péndulo con amortiguamiento, representado por la siguiente ecuación en diferencias homogénea:

$$s[n]+a_1 s[n-1]+a_2 s[n-2]=0 \quad .....(3.2)$$

donde $s[n]$ representa la amplitud del péndulo en el instante n [8].

La solución a esta ecuación en diferencias es la respuesta al impulso del péndulo, la cual corresponde a una función senoidal amortiguada. Debido a los errores existentes en la medición de la trayectoria del péndulo, Yule propuso modelar las mediciones utilizando una función excitadora. Esta idea se expresa en la siguiente ecuación en diferencias no homogénea:

$$s[n]+a_1 s[n-1]+a_2 s[n-2]=e[n] \quad .....(3.3)$$

donde $e[n]$ es el ruido blanco excitando al péndulo.

Con este experimento Yule concluyó que la amplitud y fase del movimiento del péndulo varían continuamente según los valores de la secuencia $e[n]$ en un instante determinado. Debido a los resultados obtenidos, propuso un modelo paramétrico finito con sólo polos y excitado por una secuencia de ruido blanco [19]. Con la secuencia de valores $s[n]$, haciendo un análisis de regresión, Yule encontró los parámetros del modelo, por lo que se le llamó modelo autorregresivo. La densidad de potencia espectral es entonces obtenida a partir de los parámetros del modelo autorregresivo y la varianza de ruido blanco. El trabajo de Yule fue el primero relacionado a estimación espectral paramétrica, pero no fue considerado hasta finales de la década de los 60's. En 1967, J. P. Burg en su artículo *Maximum entropy spectral analysis* [2], propuso el método de análisis espectral llamado de máxima entropía basado en los métodos autorregresivos desarrollados previamente, y fue considerado como una

extensión formal del método de Yule que, debido a la formalidad y elegancia con que fue presentado, propició que los métodos basados en la obtención de modelos paramétricos fueran más populares [8].

A partir de 1967, la utilización de modelos paramétricos en estimación espectral ha sido objeto de estudio y análisis por parte de distintos grupos de investigación. Un gran número de publicaciones han sido presentadas en congresos internacionales mostrando las ventajas de los distintos métodos, nuevas aplicaciones y nuevas formas para el cálculo de los parámetros [13]. En los artículos de Robinson [12] y Marple [9] se profundiza más en el desarrollo de las distintas teorías de análisis espectral.

3.2 CONCEPTOS BÁSICOS

Los distintos tipos de señales existentes pueden clasificarse de distintas maneras según sus características. La clasificación más general, divide a las señales en determinísticas (aquellas cuyo curso puede ser determinado por medio de algún tipo de análisis lineal), aleatorias (aquellas donde el orden individual de los datos puede ocurrir de cualquier manera), continuas, discretas o una combinación de éstas [13].

La mayoría de las señales encontradas en la práctica se encuentran disponibles para su análisis únicamente durante periodos cortos de tiempo, tal es el caso de las señales ultrasónicas. Cuando este tipo de señales se ven afectadas por agentes externos (ruido) en forma aleatoria, se convierten en secuencias de datos aleatorias [13]. Debido a esto, las señales de este tipo no pueden caracterizarse en forma simple, por lo que la estimación espectral se presenta como una herramienta importante para la caracterización de señales aleatorias.

3.2.1 SECUENCIAS ALEATORIAS

Si se considera una secuencia de datos $x[n]$ tal que el valor de x, para cualquier entero n, es un valor aleatorio; entonces la secuencia $x[n]$ es llamada secuencia aleatoria o señal discreta aleatoria. Si el parámetro n representa tiempo, $x[n]$ puede considerarse como serie de tiempo, y el modelo que representa a esta secuencia aleatoria, es conocido como proceso estocástico.

Una secuencia aleatoria o estocástica puede considerarse en cierta forma como una variable aleatoria, donde ambas se presentan como salida de un experimento donde existe un gran número de posibles resultados. En el caso de una variable aleatoria, el resultado se presenta como un sólo número, mientras que para el otro caso, la salida será una secuencia de valores.

Las señales o secuencias aleatorias pueden subclasificarse de distintas maneras, una de éstas es la que sigue la estructura probabilística de $x[n]$ según los distintos valores de la variable discreta n. Si algunas distribuciones de probabilidad no dependen de n, entonces el proceso es llamado estacionario, de otra forma toma el nombre de no-estacionario [15].

3.2.2 PROCESOS ESTACIONARIOS

El concepto de *estacionaridad* juega un papel importante en la descripción de procesos aleatorios, ya que indica la invariancia en el tiempo de algunas propiedades de procesos estocásticos. Existen funciones individuales dentro de un proceso aleatorio $x[n]$, que varían según lo hace el valor n, esto es, son funciones del tiempo; mientras que existen otros valores tales como la media, que permanecen constantes en el tiempo. En otras palabras, un proceso se dice estacionario si su función de distribución de probabilidad o algún valor esperado permanecen invariantes con respecto a una traslación en el eje del tiempo [15].

Los procesos estacionarios pueden ser de dos tipos: estacionarios en sentido estricto (SSS) o estacionarios en sentido amplio (WSS). El primero hace referencia a un proceso aleatorio $x[n]$, donde todas las funciones de distribución que describen al proceso permanecen invariantes con el tiempo. El segundo es una forma menos estricta de estacionaridad directamente relacionada con el primer valor esperado (media) y la función de autocorrelación. Un proceso $x[n]$ se dice WSS, si la media permanece constante y la función de autocorrelación es función únicamente de diferencias de tiempo y no del tiempo en sí.

3.2.3 ERGODICIDAD

En el procesamiento de señales aleatorias, normalmente se asume que se tiene un conocimiento previo de algunos valores estadísticos como

la media, función de autocorrelación y densidad de potencia espectral. En muchas ocasiones este conocimiento a priori no existe, por lo que la estimación de estos parámetros se convierte en un problema cuando se requiere caracterizar el proceso aleatorio. Desde un punto de vista práctico, sería ideal poder estimar estos valores utilizando solamente un conjunto de datos o una sola función del proceso aleatorio [15].

Un proceso se asume ergódico si sus características estadísticas a lo largo de todo el proceso son iguales a las de un sólo intervalo representativo (con un periodo fijo). Un proceso puede ser ergódico en cuanto a la media, a la función de autocorrelación y/o densidad de potencia espectral [15].

3.2.4 RUIDO BLANCO

En 1827, el científico dedicado a la botánica Robert Brown reportó por primera vez el fenómeno conocido como movimiento Browniano, al descubrir que pequeñas partículas sólidas suspendidas en agua exhibían movimientos irregulares e independientes, tal como el movimiento que presentan las partículas de polvo cuando se les observa a través de un rayo de luz. Un siglo más tarde, en 1923, Norbert Wiener desarrolló la teoría matemática que describe al movimiento Browniano, y que hoy en día es la base del modelo matemático que describe al ruido blanco [12].

El concepto de ruido blanco se aplica a cualquier proceso aleatorio con media cero y densidad de potencia espectral constante [12][16]. El nombre proviene de una analogía con el espectro ideal de la luz blanca. el cual presenta un mismo valor de energía a lo largo del eje de la frecuencia. Debido a que la densidad de potencia espectral es constante, la función de autocorrelación del ruido blanco es un impulso. En otras palabras, las muestras de ruido blanco no presentan correlación alguna [16].

3.2.5 DENSIDAD DE POTENCIA ESPECTRAL

El término de análisis espectral para algunas aplicaciones se refiere a análisis en frecuencia, por lo que la transformada de Fourier se vuelve una herramienta importante para algunas definiciones. La amplitud del espectro en frecuencia de señales de energía infinita, se define por medio de la integral de Fourier; el espectro que describe la distribución de

energía se obtiene con el módulo cuadrado de la amplitud del espectro. Las señales aleatorias poseen energía infinita, por lo que la integral de Fourier no puede ser aplicada de manera directa; sin embargo, algunos procesos aleatorios son considerados de energía finita, por lo que es posible determinar la densidad de potencia espectral para describir la distribución de potencia a lo largo del eje de la frecuencia [6].

Para señales aleatorias estacionarias (WSS), la densidad de potencia espectral se obtiene por medio de la transformada de Fourier S(f) de la función de autocorrelación $R(\tau)$, tal como se observa en las expresiones siguientes [6].

$$S(f) = \int_{-\infty}^{\infty} R(\tau) e^{-j2\pi f\tau} d\tau$$

$$R(\tau) = E\{x(t)x(t+\tau)\} \qquad .....(3.4)$$

donde E representa el valor esperado.

Cuando además se trata de un proceso ergódico, la función de autocorrelación se define de la siguiente manera [6]:

$$R(\tau) = \lim_{\theta \to \infty} \frac{1}{2\theta} \int_{-\theta}^{\theta} x(t)x(t+\tau) d\tau \qquad .....(3.5)$$

Por otro lado, si la densidad de potencia espectral de una señal se obtiene a partir de un número finito de muestras, la secuencia de valores $x(n)$ donde n puede tomar valores comprendidos en el intervalo $-\infty < n < \infty$ tiene como transformada de Fourier la siguiente expresión [11]:

$$X(f) = \sum_{n=-\infty}^{\infty} x(n) e^{-j2\pi fn} \qquad .....(3.6)$$

Del mismo modo, la función de autocorrelación de la señal muestreada se define de la siguiente manera:

$$r_{xx}(k) = \sum_{n=-\infty}^{\infty} x^{*}(n)x(n+k) \qquad .....(3.7)$$

Del mismo modo que para señales continuas, la densidad de potencia espectral puede ser obtenida al utilizar la transformada de Fourier de la función de autocorrelación.

$$S_{xx}(f) = \sum_{k=-\infty}^{\infty} r_{xx}(k) e^{-j2\pi fk} \qquad .....(3.8)$$

Las expresiones anteriores nos permiten distinguir dos métodos para calcular la densidad de potencia espectral de un proceso aleatorio. El primero, llamado método directo, consiste en computar directamente la transformada de Fourier de la señal $x(n)$; el segundo consiste en determinar la PSD de manera indirecta calculando primero la secuencia de autocorrelación de la señal, y posteriormente obtener su transformada de Fourier [13][11].

Otro concepto importante es la función de covariancia, la cual se define como la diferencia entre la función de autocorrelación en un instante t_1 y t_2, y el producto de la media en los mismos valores de tiempo. Cuando se considera un proceso WSS, el valor de la media es constante en dos instantes de tiempo distintos, y la función de autocorrelación depende únicamente de la diferencia de tiempo $\tau = t_2 - t_1$. En este caso, la función de covariancia queda definida de la siguiente manera:

$$c_{xx}(\tau) = r_{xx}(k) - |\mu|^2 \qquad .....(3.9)$$

Cuando el valor de τ es igual a cero es decir, cuando $t_2 = t_1$, la función de covariancia es igual a otro valor estadístico importante: la varianza [13].

$$\sigma_x^2 = c_{xx}(0) = r_{xx}(0) - |\mu|^2 \qquad .....(3.10)$$

Para un proceso aleatorio WSS con media cero, la función de covariancia y la función de autocorrelación se calculan de manera idéntica, y ambas son suficientes para caracterizar estadísticamente al proceso.

3.3 ESTIMACIÓN ESPECTRAL CONVENCIONAL

Los métodos de estimación espectral tradicionales están basados en los métodos directo e indirecto para calcular la PSD, sólo que ahora se realiza tomando un número finito de muestras.

Cuando se utiliza el método indirecto, la PSD se calcula por medio de la siguiente expresión:

$$S(f) = \sum_{k=-L}^{L} r_{xx}(k)e^{-j2\pi kf} \qquad .....(3.11)$$

donde los valores de r_{xx} fuera del intervalo [-L,L] son considerados como cero [13].

Dado que se cuenta con un número finito de datos disponibles, la función de autocorrelación queda definida de la siguiente manera:

$$r_{xx}(k) = \frac{1}{N-k}\sum_{n=0}^{N-k-1} x^{*}(n)x(n+k) \qquad 0 \leq k \leq N-1 \qquad .....(3.12)$$

La función de autocorrelación que se obtiene a partir de un número finito de muestras cuenta con algunas características importantes: La primera es que es una función conjugada simétrica y consistente [13], esto es, conforme N tiende a infinito, la varianza tiende a cero. La segunda es que existe un error en el cálculo de los valores, mismo que decrece conforme aumenta el número de muestras [13][11].

El método directo es conocido como el método del *periodograma* (desarrollado por Schuster en 1898), el cual es una función utilizada para detectar periodicidades ocultas que podrían estar presentes en una señal aleatoria desconocida. Dadas N observaciones $x_0,\ldots,x_{N-1}$ la densidad de potencia espectral estimada por medio del periodograma se define, para todas la frecuencias contempladas dentro del intervalo de muestreo Δt, por medio de la siguiente expresión [17]:

$$S_{PER}(k) = \frac{1}{N}\left|\sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}nk}\right|^2 = \frac{1}{N}|X(k)|^2 \qquad .....(3.13)$$

Una limitación seria que presenta este modelo es el hecho de que la señal y el ruido no pueden ser distinguidos uno del otro; por otro lado, el método del periodograma es un método inconsistente, debido a que la varianza no se reduce al aumentar el número de muestras [13][17]. Debido a esto, y con el fin de reducir la inestabilidad estadística que se presenta, es necesario calcular un promedio de valores de distintos periodogramas; sin embargo, esto resulta poco práctico cuando se requiere una estimación en tiempo real. Existen varios métodos para reducir las limitantes del modelo, los más importantes son los procedimientos de Bartlett, Blackman-Tukey y Welch [16].

3.3.1 PROCEDIMIENTO DE BARTLETT

Con el fin de reducir el valor de la varianza se calculan un número de periodogramas distintos y se hace una estimación en base al promedio de los valores obtenidos. Debido a que las propiedades estadísticas del periodograma no se alteran al incrementar el número de muestras, una forma efectiva de utilizar grandes segmentos de datos es dividiéndolos en pequeños segmentos y promediar los distintos periodogramas estimados [16]. Si $S_{per(m)}(k)$ representa cada periodograma obtenido, la estimación del periodograma de Bartlett se hace por medio de la siguiente expresión [16]:

$$S_{PERB}(k) = \frac{1}{M}\sum_{m=1}^{M} S_{PER(m)}(k) \qquad .....(3.14)$$

3.3.2 PROCEDIMIENTO DE BLACKMAN-TUKEY

Este procedimiento consiste en aplicar una ventana (Bartlett, Hamming, Hann, etc.) a los valores estimados de la función de autocorrelación antes de realizar la transformación. Del mismo modo, en el dominio de la frecuencia se puede lograr el mismo resultado al convolucionar la transformada de Fourier de la ventana de datos con el periodograma obtenido [16].

$$S_{PERB-T}(f) = \sum_{k=-M}^{M} w(k) r_{xx}(k) e^{-j2\pi kf}$$

$$S_{PERB-T}(k) = W(k) * S_{PER}(k) \qquad .....(3.15)$$

El aplicar una ventana a la función de autocorrelación se ve reflejado en una reducción de la varianza a cambio de un incremento en el error presente en la estimación [7]. En el apéndice C se presenta una lista de las ventanas más utilizadas y su respectiva transformada de Fourier.

3.3.3 PROCEDIMIENTO DE WELCH

Una última estrategia que se puede emplear con el fin de reducir la inestabilidad estadística que presenta el método del periodograma, es combinar los procedimientos de Bartlett y de Blackman-Tukey. La secuencia original de datos se divide en un número de segmentos traslapados entre sí, una ventana es aplicada a estos segmentos dando como resultado una mejor estimación del periodograma de Schuster.

3.3.4 UTILIZACIÓN DE LA TRANSFORMADA RÁPIDA DE FOURIER

La FFT es la herramienta más utilizada para computar el periodograma o cualquiera de sus variantes. Sin embargo, si el segmento de datos consiste en N datos, entonces una FFT de N puntos dará como resultado solamente N muestras. La transformada de Fourier puede ser evaluada con un número mayor de muestras al realizar un *padding* de ceros a la secuencia original, esto es, agregando un cierto número de elementos cuyo valor es siempre cero. Un padding de ceros no alterará la forma del espectro, sino que solamente permitirá evaluar el espectro estimado utilizando un conjunto de valores menos espaciados entre sí [16].

3.3.5 VENTAJAS Y DESVENTAJAS DE LA ESTIMACIÓN ESPECTRAL CLÁSICA

Los métodos de estimación espectral clásicos presentan algunas ventajas y desventajas sobre otros métodos. Las ventajas más importantes que se pueden mencionar son las siguientes: 1) resultan métodos computacionalmente eficientes si solamente se requiere de ventanas de datos reducidas (Blackman-Tukey) o si se utiliza la FFT (periodograma), 2) la PSD estimada es proporcional a la potencia de procesos senoidales, 3) resulta un modelo apropiado para algunas aplicaciones, debido a que puede representarse por medio de una suma de senoides armónicamente relacionadas. Las desventajas que presenta: 1) supresión de componentes

débiles en la primer armónica por la presencia de componentes fuertes en armónicas laterales, 2) la resolución en frecuencia está limitada por el número de datos disponibles independientemente de las características de la señal, 3) la necesidad de realizar un promedio para obtener un espectro estadísticamente consistente y 5) la aparición de valores negativos en la estimación de la PSD cuando se utiliza el procedimiento de Blackman-Tukey o el de Welch [10].

3.4 ESTIMACIÓN ESPECTRAL PARAMÉTRICA

En la sección anterior se demostró que el segundo momento estadístico de un proceso aleatorio puede representarse tanto por la función de autocorrelación, como por la función en el dominio de la frecuencia que define la densidad de potencia espectral PSD, en ambos casos tratándose de una descripción no paramétrica. Una alternativa para describir este tipo de procesos son los llamados métodos paramétricos.

Los métodos de estimación espectral convencionales aplican la transformada de Fourier a la secuencia original de datos, o a la secuencia de autocorrelación previamente calculada; ambas secuencias de datos se obtienen a partir de una ventana de datos finita, por lo que no representan fielmente al proceso original. Al utilizar una ventana para obtener una secuencia finita de datos, se hace la suposición de que todos los elementos que no están comprendidos dentro de ésta toman el valor de cero, lo cual es normalmente erróneo [7].

Debido a que normalmente se cuenta con más información del proceso que genera la secuencia de datos, es posible hacer una suposición más razonable que asignar el valor de cero a todos los elementos fuera de la ventana, lo cual permite la obtención de un modelo que represente, si no a los datos, si al proceso que los genera, eliminando la necesidad de utilizar ventanas como se hace en los métodos de estimación espectral convencionales [13][7].

La gran mayoría de los procesos aleatorios discretos encontrados en la práctica, pueden ser aproximados por una función de transferencia racional, donde la secuencia de entrada $u(n)$ y la de salida $x(n)$ se relacionan por medio de un modelo lineal general (figura 3.4.1), representado por medio de la siguiente ecuación en diferencias:

$$x[n] = -\sum_{k=1}^{p} a[k]x[n-k] + \sum_{k=0}^{q} b[k]u[n-k] \qquad .....(3.16)$$

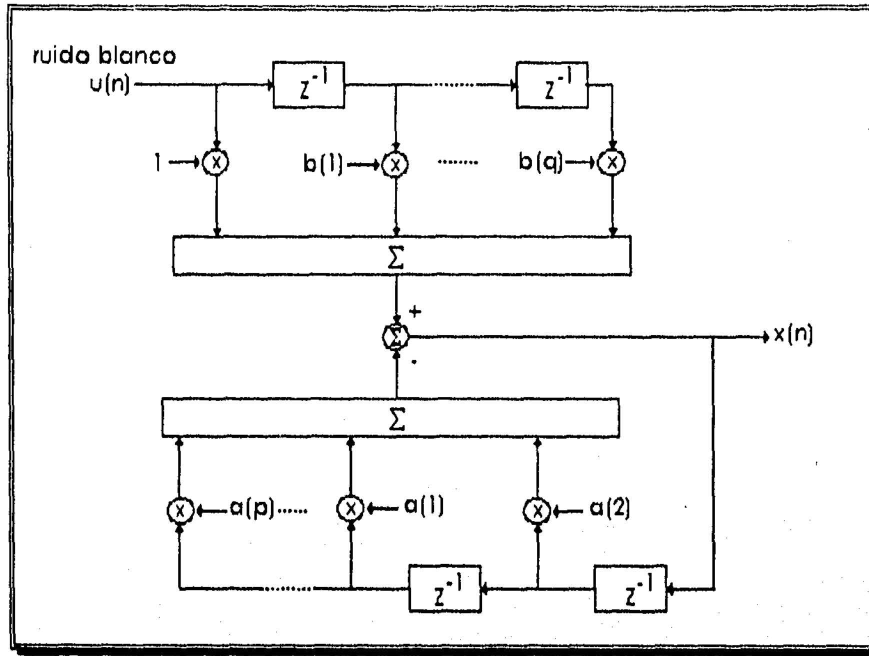


figura 3.4.1
Modelo ARMA

La función de transferencia del proceso ARMA queda entonces definida por:

$$H(z) = \frac{B(z)}{A(z)} \qquad .....(3.17)$$

donde

$$A(z) = \sum_{k=0}^{p} a[k]z^{-k}$$

$$B(z) = \sum_{k=0}^{q} b[k]z^{-k} \qquad .....(3.18)$$

Es importante mencionar la consideración que se hace con respecto a la rama AR, representada por $A(z)$, ubicando a todos sus ceros dentro del círculo unitario del plano Z garantizando la estabilidad del filtro con función de transferencia $H(z)$.

Por otro lado, la transformada Z de la función de autocorrelación $P_{xx}(z)$ a la salida de un filtro lineal, se relaciona con la secuencia de entrada $P_{uu}(z)$ por medio de la siguiente expresión:

$$P_{xx}(z) = H(z)H^*(1/z^*)P_{uu}(z) = \frac{B(z)}{A(z)}\frac{B^*(1/z^*)}{A^*(1/z^*)}P_{uu}(z) \qquad .....(3.19)$$

donde, haciendo que

$$z = e^{j2\pi f} \qquad -1/2 \le f \le 1/2 \qquad .....(3.20)$$

corresponde a la densidad de potencia espectral. En muchas ocasiones, el proceso de entrada se asume como una secuencia de ruido blanco con media cero y varianza σ^2, por lo que la PSD de la entrada corresponde al valor de σ^2. Entonces, la PSD de la salida del proceso ARMA queda de la siguiente manera:

$$P_{ARMA}(f) = P_{xx}(f) = \sigma^2\left|\frac{B(f)}{A(f)}\right|^2 \qquad .....(3.21)$$

donde, debido a que cualquier valor de ganancia puede incorporarse a σ^2, $a(0)=b(0)=1$.

Si todos los coeficientes $a(k)$ fueran cero, a excepto $a(0)=1$, entonces el proceso es conocido como modelo sólo-ceros o modelo MA de orden q.

$$x(n) = \sum_{k=0}^{q} b[k]u[n-k]$$

$$P_{MA}(f) = \sigma^2|B(f)|^2 \qquad .....(3.22)$$

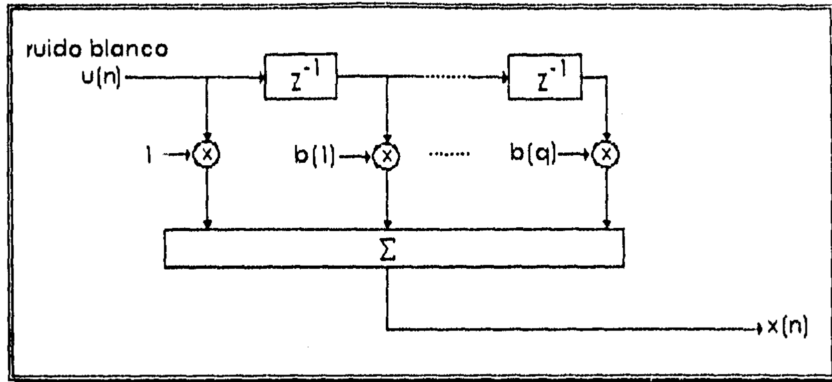


figura 3.4.2
Modelo MA

Si todos los coeficientes b(k) son considerados cero y, del mismo modo que en el caso anterior, b(0)=1, entonces el proceso es conocido como modelo de sólo-polos o AR de orden p. Donde x(n) es una regresión lineal de sí mismo y u(n) representa el error.

$$x[n] = -\sum_{k=1}^{p} a[k]x[n-k] + u[n]$$

$$P_{AR}(f) = \frac{\sigma^2}{|A(f)|^2} \qquad .....(3.23)$$

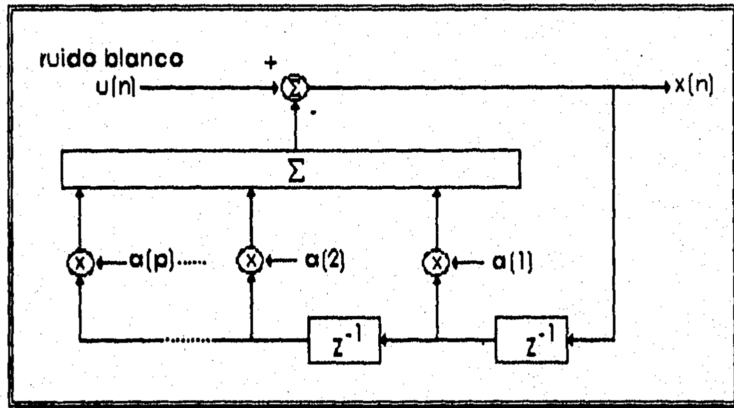


figura 3.4.3
Modelo AR

El problema que se presenta en la estimación espectral paramétrica es que no se sabe a priori que modelo utilizar, y una vez que esto se decide el número de parámetros que se requieren se vuelve un aspecto importante. Debido a qu... normalmente se cuenta con un conjunto limitado de datos, la precisión de la estimación será pobre si se estiman demasiados parámetros; sin embargo, tampoco resultaría computacionalmente eficiente estimar el menor número de parámetros posible; es necesario utilizar un criterio para determinar el número de parámetros que se requieren [13].

Los aspectos que se deben tomar en cuenta para elegir un modelo son si se trata de un proceso con picos "pronunciados", valles "profundos" o ambos. Si se busca estimar un proceso donde los picos contienen la información que se requiere para caracterizarlo, entonces se deberá utilizar un modelo que contenga polos, esto es, un modelo AR o ARMA. Por otro lado, si se trata de un proceso cuya información esté contenida en los valles, se deberá utilizar un modelo MA o ARMA.

3.5 MÉTODOS PARAMÉTRICOS AUTORREGRESIVOS

Los métodos paramétricos de estimación espectral cuentan con varias ventajas sobre los métodos convencionales, tal como se mencionó en la sección anterior. Dentro de estos, los más populares son los métodos autorregresivos, debido a que es posible calcular los parámetros del modelo resolviendo sistemas de ecuaciones lineales; en cambio, cuando se utilizan modelos MA o ARMA, es necesario la resolución de sistemas de ecuaciones no lineales para obtener los parámetros requeridos por cada modelo. Por otro lado, cuando se utilizan modelos paramétricos autorregresivos para la estimación de señales aleatorias WSS, es posible asegurar un resultado más exacto y preciso que con los métodos convencionales basados en la transformada de Fourier [7].

La generación de modelos autorregresivos de estimación espectral de señales aleatorias, ha sido el principal objetivo de varios trabajos de investigación en los últimos años [13][14]. El método más utilizado para llevar a cabo estas investigaciones ha sido el método de Burg propuesto por primera vez en 1967. Este algoritmo, además de proporcionar una mayor resolución espectral de la que se había obtenido hasta entonces, despertó el interés de un gran número de investigadores por encontrar nuevas técnicas para la obtención de los parámetros de la función

racional que representa al modelo AR. En su mayoría, estas investigaciones han sido enfocadas a corregir las deficiencias que el algoritmo de Burg presenta [3]. Información específica sobre el modelo de Burg, así como de muchos otros, puede ser obtenida en las referencias [7], [8] y [16].

Varios métodos AR para estimar la PSD de procesos aleatorios WSS han sido desarrollados hasta ahora, existiendo un comportamiento similar en todos los casos cuando se utilizan grandes secuencias de datos, pero para un número reducido de muestras, varias diferencias pueden ser encontradas [7]. En esta sección se presentan de manera muy general tres de estos métodos: Yule-Walker, covariancia y covariancia modificada; en las referencias citadas en el párrafo anterior es posible encontrar el desarrollo completo de estos métodos así como la obtención de otros modelos.

3.5.1 MÉTODO DE YULE-WALKER O DE AUTOCORRELACIÓN

Partiendo de una secuencia de datos conocida $x[n]$, donde n toma los valores de $n=0,1,\ldots,N-1$; los parámetros AR se determinan al minimizar la potencia del error estimado

$$\hat{\rho} = \frac{1}{N}\sum_{n=-\infty}^{\infty}\left|x[n]+\sum_{k=1}^{p}a[k]x[n-k]\right|^2 \qquad .....(3.24)$$

Las muestras fuera del intervalo $[0, N-1]$ son consideradas cero. La estimación del error ρ es minimizada al diferenciar la ecuación anterior con respecto a las partes real e imaginaria de $a[k]$, obteniéndose la siguiente expresión

$$\frac{1}{N}\sum_{n=-\infty}^{\infty}\left(x[n]+\sum_{k=1}^{p}a[k]x[n-k]\right)x^*[n-l]=0 \quad l=1,2,\ldots,p \qquad .....(3.25)$$

que en forma matricial puede expresarse de la siguiente manera:

$$\begin{bmatrix} r_{xx}[0] & r_{xx}[0] & \cdots & r_{xx}[-(p-1)] \\ r_{xx}[1] & r_{xx}[1] & \cdots & r_{xx}[-(p-2)] \\ \vdots & \vdots & & \vdots \\ r_{xx}[p-1] & r_{xx}[p-2] & \cdots & r_{xx}[0] \end{bmatrix} \begin{bmatrix} a[1] \\ a[2] \\ \vdots \\ a[p] \end{bmatrix} = - \begin{bmatrix} r_{xx}[1] \\ r_{xx}[2] \\ \vdots \\ r_{xx}[p] \end{bmatrix} \qquad .....(3.26)$$

donde $r_{xx}[k]$ representa a la función de autocorrelación estimada, y se define por medio de

$$r_{xx}[k] = \begin{cases} \frac{1}{N}\sum_{n=0}^{N-1-k} x^*[n]x[n+k] & \text{para } k = 0,1,\ldots,p \\ r_{xx}[-k] & \text{para } k = -(p-1),-(p-2),\ldots,-1 \end{cases} \qquad .....(3.27)$$

Algunos algoritmos para resolver el sistema de ecuaciones y encontrar los parámetros a[k] asegurando al mismo tiempo que los polos estén contenidos dentro del círculo unitario del plano Z, pueden encontrarse en la referencia [7].

La varianza de ruido blanco σ^2 (secuencia de entrada), puede ser calculada del mismo modo que ρ_{min}, por medio de la siguiente expresión:

$$\sigma^2 = \hat{\rho}_{min} = \frac{1}{N}\sum_{n=-\infty}^{\infty}\left|x[n] + \sum_{k=1}^{p} a[k]x[n-k]\right|^2$$

$$\sigma^2 = \frac{1}{N}\sum_{n=-\infty}^{\infty}\left[\left(x[n] + \sum_{k=1}^{p} a[k]x[n-k]\right)x^*[n] \right. \qquad .....(3.28)$$

$$\left. + \left(x[n] + \sum_{k=1}^{p} a[k]x[n-k]\right)\sum_{l=1}^{p} a^*[l]x^*[n-l]\right]$$

donde el segundo término de la sumatoria toma el valor de cero llevándonos a la ecuación final que define a σ^2

$$\sigma^2 = r_{xx}[0] + \sum_{k=1}^{p} a[k]r_{xx}[-k] \qquad .....(3.29)$$

El método de autocorrelación ofrece una pobre resolución comparado con otros estimadores espectrales, por lo que no es recomendado para segmentos cortos de datos [7].

3.5.2 MÉTODO DE COVARIANCIA

Al igual que en el método de autocorrelación, el estimador espectral basado en el método de covariancia, puede ser hallado al minimizar la estimación de la predicción del error ρ por medio de la siguiente expresión:

$$\hat{\rho} = \frac{1}{N-p}\sum_{n=p}^{N-1}\left|x[n] + \sum_{k=1}^{p} a[k]x[n-k]\right|^2 \qquad .....(3.30)$$

donde la única diferencia con el método utilizado anteriormente, es el rango de la sumatoria en la estimación del error. En este caso, se consideran solamente los datos observados, por lo que no se asigna ningún valor arbitrario al resto de la secuencia.

La minimización de la ecuación anterior puede llevarse a cabo de la misma manera como se hizo en el método anterior, por lo que los parámetros AR pueden ser calculados resolviendo el sistema de ecuaciones

$$\begin{bmatrix} c_{xx}[1,1] & c_{xx}[1,2] & .. & c_{xx}[1,p] \\ c_{xx}[2,1] & c_{xx}[2,2] & .. & c_{xx}[2,p] \\ . & . & . & . \\ . & . & . & . \\ c_{xx}[p,1] & c_{xx}[p,2] & .. & c_{xx}[p,p] \end{bmatrix} \begin{bmatrix} a[1] \\ a[2] \\ . \\ . \\ a[p] \end{bmatrix} = - \begin{bmatrix} c_{xx}[1,0] \\ c_{xx}[2,0] \\ . \\ . \\ c_{xx}[p,0] \end{bmatrix} \qquad .....(3.31)$$

donde

$$c_{xx}[j,k] = \frac{1}{N-p}\sum_{n=p}^{N-1} x^{*}[n-j]x[n-k] \qquad .....(3.32)$$

Del mismo modo, la varianza de ruido blanco estimada se calcula por medio de

$$\sigma^2 = \hat{\rho} = c_{xx}[0,0] + \sum_{k=1}^{p} a[k]c_{xx}[0,k] \qquad .....(3.33)$$

En este caso, y a diferencia del método de autocorrelación, no existe garantía de que los polos estén contenidos dentro del círculo unitario [7].

3.5.3 MÉTODO DE COVARIANCIA MODIFICADA

Al igual que en el método de covariancia, los parámetros AR se calculan en base a una minimización de la predicción del error estimado. Sólo que en este caso se realiza sobre un promedio de la predicción hacia atrás y hacia adelante. Una manera alternativa de ver esto, es reconociendo que la predicción del error estimado hacia atrás ρ^b se obtiene haciendo una rotación de la secuencia de datos y hallando su complejo conjugado, esto es, x´[0]=x[N-1], x´[1]=x[N-2],...,x´[N-1]=x[0]; y posteriormente aplicar el predictor hacia adelante ρ^f a esta nueva secuencia [7]. El promedio de las predicciones de error se define por medio de la siguiente expresión:

$$\hat{\rho} = \frac{1}{2}(\rho^b + \rho^f) \qquad \text{...(3.34)}$$

donde

$$\rho^f = \frac{1}{N-p}\sum_{n=p}^{N-1}\left|x[n] + \sum_{k=1}^{p} a[k]x[n-k]\right|^2$$

$$\text{.....(3.35)}$$

$$\rho^b = \frac{1}{N-p}\sum_{n=p}^{N-1-p}\left|x[n] + \sum_{k=1}^{p} a^*[k]x[n-k]\right|^2$$

Al minimizar la expresión anterior [7], se llega a que los parámetros a[k] pueden ser calculados resolviendo el sistema de ecuaciones

$$\begin{bmatrix} c_{xx}[1,1] & c_{xx}[1,2] & .. & c_{xx}[1,p] \\ c_{xx}[2,1] & c_{xx}[2,2] & .. & c_{xx}[2,p] \\ . & . & . & . \\ . & . & . & . \\ c_{xx}[p,1] & c_{xx}[p,2] & .. & c_{xx}[p,p] \end{bmatrix} \begin{bmatrix} a[1] \\ a[2] \\ . \\ . \\ a[p] \end{bmatrix} = - \begin{bmatrix} c_{xx}[1,0] \\ c_{xx}[2,0] \\ . \\ . \\ c_{xx}[p,0] \end{bmatrix} \qquad \text{.....(3.36)}$$

donde

$$c_{xx}[j,k]=\frac{1}{2(N-p)}\left(\sum_{n=p}^{N-1}x^*[n-j]x[n-k]+\sum_{n=0}^{N-1-p}x[n+j]x^*[n+k]\right) \qquad .....(3.37)$$

Y finalmente, la varianza de ruido blanco estimada, se obtiene de la siguiente expresión

$$\sigma^2=\hat{\rho}_{min}=\frac{1}{2(N-p)}\left[\sum_{n=p}^{N-1}\left(x[n]+\sum_{k=1}^{p}a[k]x[n-k]\right)x^*[n]\right. \qquad .....(3.38)$$
$$\left.+\sum_{n=0}^{N-1-p}\left(x^*[n]+\sum_{k=1}^{p}a[k]x^*[n+k]\right)x[n]\right]$$

que por último nos lleva a la expresión que define la varianza de ruido blanco

$$\sigma^2=c_{xx}[0,0]+\sum_{k=1}^{p}a[k]c_{xx}[0,k] \qquad .....(3.39)$$

El método de covariancia módificada no garantiza un filtro estable; sin embargo, en la mayoría de los casos, los polos caen dentro del círculo unitario. Este método, originalmente propuesto por Nuttall en 1976, ha sido estudiado e implementado en varios trabajos de investigación [13][14][5] demostrando que ofrece un espectro estable con alta resolución.

3.6 REFERENCIAS

[1] BLACKMAN, R. B., TUKEY, J. W., The Measurement of Power Spectra, Dover, New York, USA, 1958.

[2] BURG, J. P., Maximum entropy spectral analysis, Proc. 37th Meet. Soc. Explor. Geophys., Oklahoma City, USA, 1967.

[3] CADZOW, J., Spectral Analysis, Handbook of Digital Signal Processing Engeeniring Applications, Pp. 701-741, Edited by Elliot, D., Academic Press Inc., USA, 1987.

[4] COOLEY, J. W., TUKEY, J. W., An algorithm for machine calculation of complex Fourier series, Math. Comput. 19, 297-301, 1965.

[5] GARCIA, D. F., SOLANO, J., BENITEZ, H., Parallel Implementation of Parametric Spectral Estimation for Doppler Blood Flow Instrumentation, The Proceedings of the 6th International Conferencie on Signal Processing Applications & Technology, Boston, Massachusetts, USA, 1995.

[6] GUNTER, D., Spectral Estimation, Signal Processing II: Theories and Applications, Pp. 447-453, Elsevier Science Publishers B. V., North-Holland, 1983.

[7]KAY, S., Modern Spectral Estimation, Prentice Hall Signal Processing Series

[8] KUMARESAN, R., Spectral Analysis, Handbook of digital signal processing, Pp. 1143-1243, Edited by Mitra, S. and Kaiser J., Wilsy-Interscience publication, USA, 1993.

[9] MARPLE, L., KAY, S., Spectral Analysis a Modern Perspective, Proc. of the IEEE No.69, Pp. 1380-1419, 1981.

[10] MARPLE, S. L JR., Spectral Estimation with Applications, Modern Signal Processing, Pp. 129-154, Edited by Thomas Kailath, Hemisphere Publishing Corporation, USA, 1985.

[11] PROAKIS, J., RADER, C., LING, F., NIKIAS, C. L., Advanced Signal Processing, Macmillan Publishing Company, USA, 1992.

[12] ROBINSON, E., A Historical Perspective of Spectrum Estimation, Invited paper, Proceedings of the IEEE, Vol. 70, No. 9, September 1982.

[13] RUANO, M. G., Investigation of Real-Time Spectral Analysis Techniques for Use with Pulsed Ultrasonic Doppler Bood-Flow Detectors, Thesis submited to the University College of North Wales, Bangor, U. K., 1992.

[14] RUANO, M. G., GARCIA, D. F., FISH, P. J., Fleming P. J., Alternative parallel implementations of an AR-modified covaiance spectral estimator for diagnostic ultrasonic blood flow studies, Practical aspects and experiences, Parallel Computing 19, Pp. 463-476, North-Holland, 1993.

[15] SHANMUGAN, K. S., BREIPOHL, A. M., Random Signals Detection, Estimation and Analysis, USA, 1991.

[16] THERRIEN, C. W., Discrete Random Signals and Statistical Signal Processing, Prentice Hall Signal Processing Series, USA, 1992.

[17] VAITKUS, P. J., Cobbold, R. S. C., A Comparative Study and Assessment of Doppler Ultrasound Spectral Estimation Techniques Part I: Estimation Methods, Ultrasound in Med. & Biol. Vol. 14, No. 8, Pp. 661-672, USA, 1982.

[18] WIENER, N., Generalized harmonic analysis, Acta. Math. 55, 117-258, 1930.

[19] YULE, G. U., On a method of investigating periodicities in disturbed series with special reference to Wolfer's sun-spot numbers, Philos. Trans. R. Soc. London, Ser. A 226, 276-298, 1927.

CAPÍTULO 4

CAPÍTULO 4

NODO DE PROCESAMIENTO HETEROGÉNEO

4.1 INTRODUCCIÓN

El procesamiento de señales de Doppler de ultrasonido, involucra algoritmos de estimación espectral que requieren de un gran número de operaciones aritméticas, por lo que son considerados algoritmos computacionalmente intensivos. Debido a esto surge la necesidad de utilizar técnicas de procesamiento paralelo para obtener la densidad de potencia espectral, así como algunos otros elementos que nos permitan caracterizar este tipo de señales. Los transputers son procesadores diseñados específicamente para soportar concurrencia y paralelismo a nivel de software y de hardware [5]; sin embargo, este tipo de procesadores, considerados de granularidad media, presentan algunas deficiencias en tareas asociadas al procesamiento de señales [3], ya que su capacidad de comunicación se da solamente a nivel de procesos. Por otro lado, los procesadores digitales de señales, son procesadores de granularidad fina, ya que su capacidad de comunicación también se da a nivel de instrucción, lo cual satisface los requerimientos de algunas tareas asociadas al procesamiento digital de señales [3]. Tomando en cuenta las características de ambos procesadores, se desarrolló la idea de un nodo de procesamiento heterogéneo, cuyos módulos principales son: un procesador transputer T805, un DSP TMS320C30 y una interfaz de comunicación vía *link adaptor* [3][1][4].

4.2 PROCESADOR DIGITAL DE SEÑALES, FAMILIA TMS320

El procesamiento digital de señales abarca un amplio campo de aplicación, algunos ejemplos son el procesamiento de imágenes y de voz, flujometría y el filtrado digital de señales de ultrasonido. Éstas y muchas otras áreas, tienen en común las siguientes características: son algoritmos computacionalmente intensivos, por lo general requieren procesamiento en tiempo real, flexibilidad y obtención de ventanas de datos [2].

Debido a los requerimientos específicos de cada área de aplicación del procesamiento digital de señales, surge la necesidad de procesadores más especializados y diseñados específicamente para satisfacer los requerimientos asociados a este tipo de procesamiento. Uno de los desarrollos más importantes es el diseño de la familia TMS320 de DSP's (*Digital Signal Processors*) en un sólo circuito integrac [2][3].

El primer miembro de la familia TMS320 fue el TMS32010, presentado por Texas Instruments en 1982, el cual ejecuta hasta 5 millones de operaciones propias del procesamiento digital de señales en un segundo. A partir del diseño del TMS32010 se dieron algunas variantes que condujeron a la creación de los modelos TMS320C10, TMS320C15 y el TMS320C17. Posteriormente, se desarrolló la segunda generación de DSPs; siendo el TMS32020 el primero y base del diseño del TMS320C25, el cual es capaz de llevar a cabo 10 millones de instrucciones por segundo; además, cuenta con otras características que lo hacen más eficiente que sus predecesores, tales como espacio para memoria expandida, capacidad de realizar multiplicaciones y uso del acumulador en un mismo ciclo de reloj, multiprocesamiento y funciones expandidas para entrada y salida de datos. La tercer generación de la familia TMS320 (la serie TMS320C3X), tiene la capacidad de realizar 33 MFLOPS. La cuarta generación vino con el desarrollo del TMS320C40, el cual presentó la innovación de seis puertos de comunicación en el mismo chip y memoria global compartida. El desarrollo del TMS320C50 dio origen a otra división en la familia TMS320, al ser el DSP que marcó el estándar en microprocesadores de 16 bits de punto fijo. El último miembro de la familia presentado por Texas Instruments es el TMS320C8X, que cuenta con dos modelos: el TMS320C80 y el TMS320C82, los cuales integran cuatro DSPs y un procesador central RISC en un sólo circuito integrado [2].

La característica más sobresaliente de los Procesadores Digitales de Señales de la familia TMS320 es la capacidad de procesar operaciones aritméticas a gran velocidad, con el fin de satisfacer algoritmos matemáticamente intensivos que requieran de procesamiento en tiempo real. Esto se debe a que la familia TMS320 cuenta con un diseño especializado, con una arquitectura Harvard modificada, trabajo interno en pipeline, hardware multiplicador específico, instrucciones especiales y un rápido ciclo de instrucción. Todo esto con la finalidad de permitir que la mayoría de las operaciones asociadas al procesamiento de señales, se realicen en un sólo ciclo de reloj [2].

La aquitectura Harvard original ubica los datos en locaidades de memoria separadas de las que guardan el código del programa, lo cual permite que exista un traslape del ciclo de *fetch* y el de *execute*. La modificación que se hace en la familia TMS320 permite la transferencia entre el espacio de datos y de programa, maximizando la capacidad de procesamiento al mantener dos buses separados para ejecutar instrucciones a máxima velocidad [2].

El trabajo interno en pipeline se lleva a cabo con la finalidad de reducir al máximo el ciclo de instrucción. El pipeline se realiza a cualquier nivel, dependiendo de cada generación. La primera utiliza un nivel de profundidad igual a dos, la segunda un nivel tres y el resto un nivel cuatro. Lo que significa que cada dispositivo ejecuta desde dos hasta cuatro instrucciones en paralelo, y cada una de ellas se encuentra en una etapa distinta de ejecución. Por ejemplo, en un pipeline de nivel tres, los ciclos de fech, decode y execute, se realizan en paralelo con distinta instrucción cada uno, permitiendo un traslape en el procesamiento de cada una de ellas [2].

Un multiplicador dedicado es una característica importante que está directamente vinculada con el desempeño de un procesador digital de señales. Por esto, la familia TMS320 cuenta con hardware específico para realizar este tipo de operaciones aritméticas, permitiendo realizar instrucciones de multiplicación en un sólo ciclo de reloj. En particular, la instrucción MPY, carga los elementos que intervienen en la operación al multiplicador dedicado, donde se lleva a cabo la instrucción para posteriormente colocar el producto en un registro específico [2][3].

Otra característica especial de la familia de DSPs TMS320 es una serie de instrucciones especiales. Una de ellas es la instrucción DMOV, la cual se utiliza cuando se quiere diseñar un filtro digital y los datos deben ser recorridos un periodo de muestreo para recibir los datos actuales, logrando mediante esta instrucción un retraso en el dominio del tiempo. Otra operación especial es la que se lleva a cabo por medio de la instrucción LTD, la cual realiza lo mismo que las operaciones LT (carga el multiplicador en un registro), DMOV y APAC (suma un producto con el acumulador) en un sólo ciclo de reloj. A partir de la segunda generación de procesadores TMS320, se agregaron las instrucciones RPTK (repite la siguiente instrucción N+1 veces) y MACD (realiza las instrucciónes LT, DMOV, MPY y APAC en un ciclo de reloj) [2].

Debido al tiempo de duración del ciclo de instrucción, la familia de procesadores TMS320 está considerada dentro de las primeras opciones para aplicaciones de tiempo real. En la tabla 4.2.1 se muestra el tiempo de ejecución de cada modelo de la familia TMS320 [2][3].

MODELO	CICLO DE INSTRUCCIÓN (ns)
TMS320C10	160-200
TMS320C20	160-200
TMS320C25	100-125
TMS320C30	40-60
TMS320C31	33-60
TMS320C32	30-50
TMS320C40	33-50
TMS320C44	33-50
TMS320C50	50
TMS320C54X	20-25
TMS320C80	20
TMS320C82	20

tabla 4.2.1
Ciclo de instrucción

Otras características importantes, además de las ya mencionadas, son la velocidad, tipos de datos, acceso directo a memoria, capacidad de memoria RAM, ROM y EPROM. En la tabla 4.2.2 se muestran las características principales de algunos modelos de la familia TMS320 [3].

MODELO TMS320	TIPO DE DATOS	CICLO (ns)	MEMORIA			I/O		
			RAM	ROM	EXTERNA	PAR	SER	DMA
C10-25	ENTERO	160	144	1.5K	4K	8X16	-	-
20	ENTERO	200	544	-	128K	16X16	1	EXT
C25-50	ENTERO	80	544	4K	128K	16X16	1	EXT
C30	FLOTANTE	60	2K	4K	16M	16MX32	1	INT/EXT
C30-50	FLOTANTE	40	2K	4K	16M	16MX32	1	INT/EXT
C40-40	FLOTANTE	50	2K	4K	4G	4GX32	6	INT/EXT
C40-50	FLOTANTE	40	2K	4K	4G	4GX32	6	INT/EXT
C50-50	ENTERO	50	8.5K	2K	128K	16X16	1	EXT
C80	FLOTANTE	20	50K	2K	4G	4GX32	-	-
C82	FLOTANTE	20	44K	2K	4G	4GX32	-	-

tabla 4.2.2
Características de algunos modelos de la familia TMS320

4.3 DISEÑO DEL NODO DE PROCESAMIENTO HETEROGÉNEO

Debido a la experiencia derivada de trabajos de investigación en estimación espectral utilizando técnicas de procesamiento paralelo, en 1994 surgió el interés por integrar dentro de una misma plataforma de procesamiento paralelo un nuevo tipo de procesador con características complementarias a las del transputer [3]. El procesador que se consideró más adecuado para llevar a cabo esta idea, fue un procesador digital de señales, debido a que cuenta con características complementarias a las del transputer; además cuenta con hardware específico para realiza operaciones aritméticas de punto flotante de manera eficiente [2][6]. También se tomaron en cuenta varias alternativas de diseño [3], y se optó por utilizar un transputer T805, un DSP TMS320C30 y una interfaz que comunique a ambos procesadores.

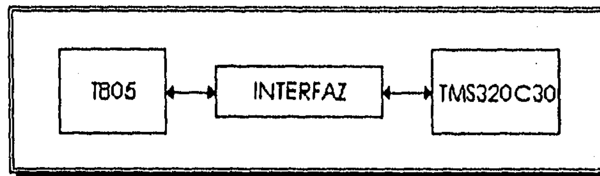


figura 4.3.1
Nodo de procesamiento heterogéneo

4.3.1 PROCESADORES

El transputer T805 es un procesador CMOS de 32 bits. Éste cuenta con una unidad de punto flotante de 64 bits que realiza operaciones de longitud simple o doble siguiendo el estándar ANSI IEEE 754-1985 a una velocidad de 3.3 MFLOPS para procesadores a 30 MHz; tiene una memoria RAM interna de 4kbytes y una interfaz de memoria externa de 32 bits que proporciona una velocidad de 40 Mbytes/seg. Una de las características más importantes que posee, son los cuatro links de comunicación punto a punto, que permiten la construcción de redes de transputers y facilitan la comunicación punto a punto sin necesidad de lógica externa. Estos links de comunicación ofrecen una velocidad estándar de 10 Mbits/seg, pero pueden soportar hasta 20 Mbits/seg (capítulo 2) [5].

El TMS320C30 es un procesador de punto flotante de 32 bits capaz de realizar hasta 33 MFLOPS siguiendo el estándar de punto flotante TMS320C30; opera a una velocidad de 16.7 millones de instrucciones por segundo con un ciclo de instrucción de 60 ns; cuenta también con un

espacio de memoria de 16 millones de palabras de 32 bits, y 6 mil palabras de 32 bits en memoria RAM y ROM dentro del mismo procesador. Su diseño contempla un bus interno que une los dispositivos periféricos por medio de un mapeo de memoria interna, y dos puertos seriales síncronos de E/S que operan a 2.5 Mbits/seg y que poseen un modo especial de protocolo que garantiza la sincronización de la información [2][6].

4.3.2 INTERFAZ POR MEMORIA COMPARTIDA

Se consideraron dos métodos de interconexión: interfaz por memoria compartida e interfaz por canales de comunicación. El primero está basado en un espacio de direcciones común con la finalidad de que ambos procesadores puedan accesar a una misma área de memoria; en el segundo los elementos de procesamiento no comparten memoria sino que son conectados a través de canales para transferencia de datos [3].

Para el diseño de la interfaz por memoria compartida se consideraron cuatro alternativas: *crossbar switches*, sistemas de memoria multipuerto, sistemas de bus compartido y redes multiestado [3].

Crossbar switches. Existen varias rutas separadas conectadas a cada banco de memoria y a cada procesador, por esto es posible realizar en todas las unidades de memoria accesos simultáneos, la única limitante que se puede presentar es cuando el mismo banco de memoria es requerido por varios procesadores al mismo tiempo.

Memoria multipuerto. Cuando se tienen dos procesadores interconectados por una memoria de puerto dual o doblepuerto, la comunicación puede darse al escribir o leer datos en la memoria compartida utilizando interrupciones para solicitar la atención de un procesador hacia la memoria. Los sistemas de control se encuentran concentrados en módulos de memoria los cuales cuentan con varios puertos de comunicación por los cuales se accesa a la información interna.

Bus compartido. Es una ruta de comunicación para conectar memorias y procesadores. Cuando dos o más elementos de procesamiento son conectados al mismo bus, debe asignarse un tiempo fijo para cada procesador con el fin de establecer la comunicación, del mismo modo, deben existir mecanismos de arbitraje para el manejo de

requerimientos simultáneos. Este último punto se vuelve un impedimento para la transferencia simultánea entre procesadores y memoria.

Redes multiestado. Se construyen utilizando un arreglo de bloques modulares, donde cada elemento puede ser configurado como conexión directa o cruzada. La terminal de entrada puede ser un elemento de procesamiento con memoria local y la terminal de salida puede ser un elemento de memoria global.

La causa por la que fue desechado el uso de memoria compartida ha sido la contención que se presenta entre los elementos de procesamiento cuando éstos requieren accesar a un área de memoria común en el mismo instante. Además, en un esquema de memoria compartida, al aumentar el número de procesadores ocurre una degradación en la eficiencia global debido a la competencia por la utilización del bus de acceso a memoria, ocasionando un tiempo ocioso en cada procesador [3].

4.3.3 INTERFAZ POR CANAL DE COMUNICACIÓN

El diseño de la interfaz por canal de comunicación aprovecha los links de comunicación serial del transputer, evitando así el cuello de botella que se presenta en los canales de comunicación de sistemas con memoria compartida [3].

Una conexión punto a punto, involucra una unidad transmisora, una receptora y un canal de interconexión, el cual está formado de un campo de información y una estructura de control que define el protocolo del canal para el flujo de información[3].

Los links del transputer son sistemas de comunicación full duplex de alta velocidad que permiten la transferencia de información entre los miembros de la familia INMOS transputer [5]. Estos canales de comunicación funcionan como interconectores de propósito general cuando se utiliza otro tipo de procesadores. Los adaptadores de canal IMS C011 y IMS C012 son dispositivos que habilitan el canal de comunicación serial para ser conectado a puertos paralelos [3][5]. Estos adaptadores, conocidos como link adaptors, manejan el protocolo de comunicación a través del link y los datos son leídos o escritos a través de un puerto.

Para el diseño de la interfaz por canal de comunicación se utilizó el IMS C011 el cual maneja dos modos de operación [5]:

Modo 1. El C011 convierte un canal de comunicación en dos interfaces independientes, una de entrada y otra de salida, mismas que pueden ser utilizadas de manera independiente por un dispositivo periférico.

Modo 2. Permite tener una interfaz entre el canal de comunicación serial y el bus de un microprocesador. En este modo, el C011 cuenta con dos salidas de interrupciones, una para indicar que la entrada de datos está disponible y la otra para indicar que el buffer de salida está vacío.

El C011 soporta la velocidad de comunicación estándar de 10 Mbits/seg y también de 20 Mbits/seg. Cuando se utiliza el modo 1 de operación se pueden conectar dos dispositivos C011 vía los puertos paralelos para realizar un cambio de frecuencia entre las diferentes velocidades de los links (5, 10 ó 20 Mbits/seg)[3][5].

El diseño final del nodo de procesamiento heterogéneo se basa en la utilización del link adaptor IMS C011 en el modo 1 de operación ya que convierte un link de comunicación en dos interfaces independientes de 8 bits, una de entrada y otra de salida.

Debido a que el transputer T805 y el link adaptor C011 pertenecen a la familia INMOS, la comunicación entre ambos es sincronizada utilizando el protocolo de comunicación descrito en la sección 2.3 del capítulo anterior. El protocolo de comunicación entre el link adaptor y el DSP se logró utilizando registros de corrimiento y lógica de control dedicada [3].

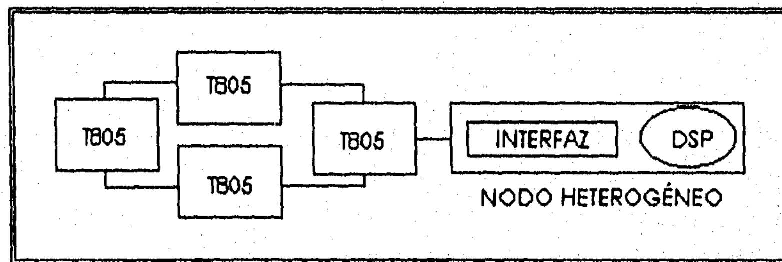


figura 4.3.2
Arquitectura heterogénea de procesamiento paralelo

El nodo de procesamiento heterogéneo dio origen a una arquitectura heterogénea de procesamiento paralelo conformada por un transputer T805, un DSP TMS320C30 y una interfaz de comunicación vía link adaptor IMS C011. Esta arquitectura surge como una propuesta para reducir los tiempos en el cálculo de funciones y operaciones específicas, propias del procesamiento digital de señales, y ser integrada a una plataforma de procesamiento paralelo como se muestra en la figura 4.3.2 [1][3][4].

4.3.4 TEORÍA DE OPERACIÓN DE INTERFAZ DEL NODO DE PROCESAMIENTO HETEROGÉNEO

En la figura 4.3.3 se muestra un diagrama de bloques de la interfaz de comunicación del NPH.

Para la comunicación y transferencia de datos entre el transputer y el DSP se lleva a cabo la siguiente secuencia: los datos son enviados por el transputer al módulo link adaptor utilizando uno de los links de comunicación serial. En este módulo los datos en formato serial son transformados a formato paralelo para ser transferidos al módulo de transmisión donde, por medio de lógica de control, son transformados una vez más a formato serial para ser enviados al DSP por medio del puerto serie. Una vez que los datos han sido procesados por el C30, estos son enviados desde el puerto serie al módulo de recepción, el cual convertirá los datos a un formato paralelo utilizando también lógica de control. Los datos con el formato paralelo son enviados de nuevo al módulo interfaz link adaptor el cual se encargará de transmitirlos serialmente al transputer [3].

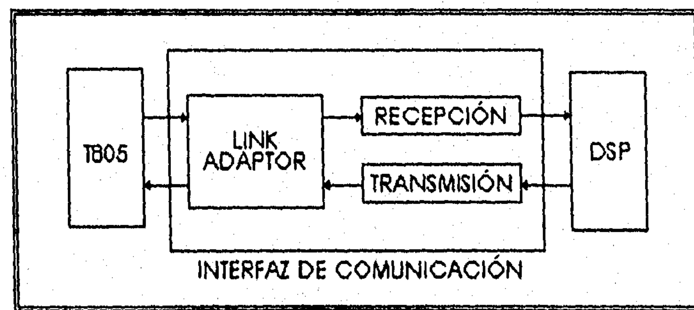


figura 4.3.3
Diagrama de bloques de la interfaz de comunicación del NPH

4.4 DESARROLLO DE LOS PROGRAMAS DE COMUNICACIÓN

El transputer es un procesador diseñado para soportar procesamiento paralelo y, debido a los links de comunicación serial que posee, permite construcción de sistemas multiprocesadores. Lenguajes de programación, tales como Occam y C paralelo, poseen librerías específicas para explotar al máximo estas capacidades, ya que cuentan con instrucciones para declarar canales de comunicación entre procesadores y transmitir información de un punto a otro en una red de procesadores transputer [5]. La interfaz de comunicación del nodo de procesamiento heterogéneo se divide en tres bloques principales tal como se presentó en la sección anterior. El primer bloque, es el link adaptor IMS C011. Éste, al igual que el transputer IMS T805, pertenece a la misma familia de procesadores INMOS [5], por lo que no es necesario realizar programas específicos de comunicación; solamente es necesario definir al NPH como un elemento externo que será conectado a la red de procesadores transputer vía link adaptor.

El DSP TMS320C30 es un procesador que soporta paralelismo interno, es decir, cuenta con instrucciones y hardware específico que le permiten realizar más de una operación en un sólo ciclo de reloj [2][6]. Debido a que se trata de un procesador de la familia TMS320 de Texas Instruments, no cuenta con librerías o links de comunicación serial que faciliten su interconexión con sistemas de transputers, por esto fue necesario la elaboración de programas de comunicación que recibieran la información y la transmitieran al transputer vía puerto serie.

Durante el diseño del hardware y la lógica que conforman el nodo de procesamiento heterogéneo, se realizaron programas de comunicación que permitieran la transmisión y recepción de información en el DSP. Estos programas fueron desarrollados con fines evaluativos y de manera muy específica, lo que los hacía poco flexibles para el desarrollo de aplicaciones posteriores.

Con la finalidad de hacer más flexible la comunicación entre procesadores utilizando el NPH, se dividió el proceso de comunicación en tres módulos principales: inicialización, recepción y transmisión de la información.

Inicialización. Al ejecutar un programa que requiera del NPH, es necesario inicializar el DSP, es decir, notificarle vía software que el

programa ha comenzado y que se iniciará el intercambio de información por el puerto serie. El programa desarrollado se basa en el programa de comunicación original implementado en [3]. En él se asignan las direcciones correspondientes al timer y al puerto serie del DSP y se espera la llegada de un dato (de un byte de longitud) indicando que el programa ha iniciado.

Recepción. El módulo de recepción se basa en el mismo programa que el de inicialización. En éste, después de asignar los valores correspondientes a las direcciones del timer y el puerto serie, se espera la llegada de un número **P** de datos provenientes del transputer en forma serial y en paquetes de ocho bits. Debido a que el DSP TMS320C30 utiliza un formato distinto al del transputer para notación de números reales (de punto flotante)[2][6], en este módulo se hace un cambio de formato de IEEE a TMS320 para que el DSP pueda operar sobre los datos.

Transmisión. La transmisión de datos del DSP al transputer se realiza de la misma manera que el proceso de recepción; se transmiten **P** datos en forma serial y en paquetes de ocho bits. Antes de mandar los datos al transputer se realiza un cambio de formato de TMS320 a IEEE.

Los programas de inicialización, recepción y transmisión se realizaron en forma separada y fueron escritos en lenguaje ensamblador para el DSP; son controlados por un cuarto programa en C para el TMS320C30 y son llamados como funciones externas. Los datos que se reciben y mandan (módulos de recepción y transmisión) se almacenan en un vector **data_input** declarado como variable global en los cuatro programas. El número de datos que se reciben y transmiten se asigna desde el programa en C por medio de la variable **P.** Los programas, así como algunos comentarios referentes a su elaboración se presentan en el apéndice B.

4.5 CAMBIO DE FORMATO IEEE-TMS320

Como se mencionó anteriormente, el transputer y el DSP manejan distintas representaciones para números de punto flotante, por lo que es necesario realizar conversiones de formato cuando se requiere enviar datos del transputer al DSP y viceversa.

En los módulos de recepción y transmisión es necesario hacer un cambio de formato IEEE-TMS320 ó TMS320-IEEE según sea el caso. Esta

conversión se logró implementando el algoritmo de conversión desarrollado por Texas Instruments en 1990 [2].

4.5.1 FORMATO ANSI IEEE 754-1985

En el formato IEEE, la mantisa se representa en la forma de magnitud signada; además no todos los valores se determinan utilizando la misma fórmula ya que el exponente se utiliza, en ocasiones, para ubicar números codificados de manera distinta. Por ejemplo, si el valor del exponente es 255, el valor no se considera un número, lo cual es muy útil cuando se requiere enviar comandos o instrucciones a otros procesos. Otro ejemplo podría ser cuando el valor del exponente es cero, lo que define a la mantisa como desnormalizada, esto es, el número representado en formato IEEE, tiene un valor inferior al menor número que puede representarse con formato TMS320 [2].

El estándar 745-1985 está definido para números de punto flotante de precisión sencilla, sencilla extendida, doble y doble extendida. En la figura 4.5.1 se muestra el formato de precisión sencilla según el estándar IEEE 754-1985 para representación de números de punto flotante [2].

El campo s corresponde al bit de signo de la mantisa (0 equivale a positivo y 1 a negativo). El exponente se expresa en un campo de 8 bits y la parte fraccionaria en uno de 23 bits. La representación decimal de un número expresado de esta forma, se da según uno de los siguientes casos [2].

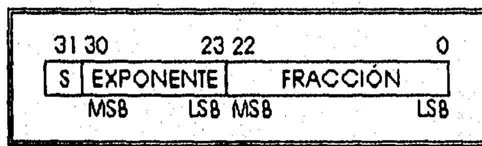


figura 4.5.1
Estándar IEEE para números de punto flotante

Caso 1. Si el exponente es igual a 255 y la parte fraccionaria es distinta de cero, entonces la representación no corresponde a un número.

Caso 2. Si el exponente es igual a 255 y la parte fraccionaria igual a cero, se trata de un valor infinito (positivo o negativo).

Caso 3. Si el exponente es mayor a cero y menor a 255, entonces la representación decimal se obtiene a partir de la siguiente expresión:

$$v = (-1)^{s} 2^{EXP-127} (1.\mathbf{FRAC})$$

Donde s corresponde al bit de signo y toma valores de 0 ó 1, **EXP** y **FRAC** corresponden a la representación decimal del exponente y la parte fraccionaria respectivamente.

Caso 4. Si el exponente es igual a cero y la fracción diferente de cero, entonces se trata de un número desnormalizado definido por la siguiente expresión:

$$v = (-1)^{s} 2^{EXP-126} (0.\mathbf{FRAC})$$

Caso 5. Si el exponente y la fracción son iguales a cero, la representación decimal corresponde al valor cero.

4.5.2 FORMATO TMS320C30

El formato TMS320 para números flotantes es una representación más simple que el estándar IEEE y casi todos los números representados de esta manera (ANSI IEEE) pueden convertirse a formato TMS320 sin perder precisión; tanto el exponente como la mantisa se expresan utilizando el complemento a dos. En la figura 4.5.2 se muestra la representación de un número flotante de precisión sencilla según el estándar TMS320 [2].

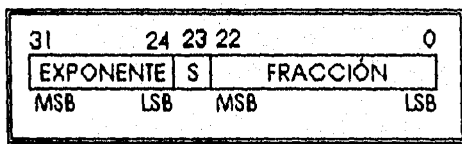


figura 4.5.2
Representación TMS320
para números de punto flotante

La representación decimal del número de punto flotante representado utilizando el estándar TMS320 se obtiene utilizando la siguiente expresión:

$$v = \{(-2)^{S} + (.FRAC)\} 2^{EXP}$$

La representación del cero para este formato, se hace de manera simple, a diferencia del IEEE donde puede expresarse de dos maneras: +0 ó -0.

4.6 REFERENCIAS

[1]GARCÍA NOCETTI, D. F., SOLANO GONZÁLEZ, J., MARTÍNEZ FLORES, J., RAMOS HERNÁNDEZ, D., Heterogeneous Parallel Architecture for Improving Signal Processing Performance in Doppler Blood Flow Instrumentation, 10th International Parallel Processing Symposium, Honolulu Hawaii, USA, 1996.

[2]PAPAMICHALIS, P. E., Digital Signal Processing Applications with the TMS320 Family, Vol. 3, Ed. Prentice Hall and Digital Signal Processing Series, Texas Instruments, USA, 1990.

[3]RAMOS HERNÁNDEZ, D., Diseño e implementación de un Nodo Heterogéneo de Procesamiento Digital de Señales Utilizando un Transputer y un DSP, Tesis, Facultad de Ingeniería, UNAM, México, 1995.

[4]SOLANO GONZÁLEZ, J., GARCÍA NOCETTI, D. F., RODRÍGUEZ VÁZQUEZ, K., RAMOS HERNÁNDEZ, D., Parallel Genetic Algoritms in Autorregressive Modelling Using a Heterogeneous Architecture, 13th IFAC World Congress, San Francisco, USA, 1996.

[5]The Transputer Databook, INMOS, SGS-Thomson Microelectronics, third edition, 1992.

[6]TMS320C3X, User's Guide, Texas Instruments, Digital Signal Products,1992.

CAPÍTULO 5

CAPÍTULO 5

IMPLEMENTACIÓN DEL ESTIMADOR ESPECTRAL PARAMÉTRICO DE COVARIANCIA MODIFICADA

5.1 MÉTODO DE COVARIANCIA MODIFICADA

Los métodos de estimación espectral convencionales presentados en el capítulo 3, utilizan la Transformada Rápida de Fourier sobre los datos contenidos en la ventana de muestreo. Fuera de ésta, los valores se consideran cero, provocando cierta distorsión en el espectro estimado de la señal. Los métodos paramétricos desechan esta suposición, y predicen los valores fuera del segmento de datos con base en los datos contenidos en él [4][11][14].

Como se vio en el capítulo 3, la modelación paramétrica en estimación espectral consiste en escoger un modelo apropiado, estimar los parámetros del modelo y sustituir estos valores en la expresión que define la PSD según el modelo seleccionado[7].

La modelación paramétrica se divide en tres tipos: modelos autorregresivos (AR), de movimiento promedio (MA) y autorregresivos de movimiento promedio (ARMA). Los modelos AR son utilizados cuando se tiene una señal con picos pronunciados y valles no muy profundos. Los MA se utilizan cuando la señal presenta valles profundos y picos no muy pronunciados. El último modelo, el ARMA, es un complemento de los dos anteriores[7]. Un caso particular es la señal Doppler de ultrasonido [1][5][6], la cual presenta picos en toda su extensión y debido a esta característica, los modelos más apropiados para estimar este tipo de señales, son el AR y el ARMA. En publicaciones previas relacionadas con estimación espectral, se investigó y evaluó el desempeño de varios estimadores espectrales, identificando al estimador espectral paramétrico de covariancia modificada como costo-efectivo [11].

La estimación de la densidad de potencia espectral (PSD) de una señal utilizando un modelo autorregresivo, involucra el cálculo de p coeficientes que minimicen la suma de los errores entre la señal estimada y los datos existentes. Posteriormente, estos parámetros son sustituidos en la

expresión matemática que representa la PSD según el modelo seleccionado[3]. El método de covariancia modificada hace el cálculo de estos parámetros minimizando el promedio del error medio cuadrático hacia adelante y hacia atrás haciendo uso de las muestras de datos disponibles[11][3][7], tal como se muestra en la sección 3.5.3, donde después de minimizar la ecuación 3.34 y algunas simplificaciones, la predicción óptima del error medio se obtiene resolviendo el sistema de ecuaciones definid por las expresiones 3.36 y 3.37.

Del mismo modo, la varianza de ruido blanco estimada se obtiene a partir de la expresión 3.39.

Debido a que la matriz de covariancia es hermitiana ($C_{xx}[j,k]=C_{xx}[k,j]$) y positiva semidefinida [7], es posible utilizar el método de Choleski para la resolución del sistema de ecuaciones.

Representando el sistema de ecuaciones como una ecuación matricial se tiene:

$$\mathbf{C}\cdot\mathbf{a}=\mathbf{b} \qquad ....(5.1)$$

donde **C** es la matriz de covariancia, **a** el vector de parámetros y **b** el vector del lado derecho. El primer paso es la descomposición de la matriz **C** de la siguiente manera:

$$\mathbf{C}=\mathbf{L}\mathbf{D}\mathbf{L}^{H} \qquad ....(5.2)$$

donde, **L** es una matriz triangular inferior de **n X n** elementos con 1's en la diagonal principal y **D** es una matriz diagonal con n elementos reales positivos en su diagonal principal.

Sustituyendo (5.2) en (5.1):

$$\mathbf{L}\mathbf{D}\mathbf{L}^{H}\mathbf{a}=\mathbf{b} \qquad ....(5.3)$$

y haciendo que:

$$\mathbf{y}=\mathbf{D}\mathbf{L}^H\mathbf{a} \qquad \text{....(5.4)}$$

el sistema se puede ahora expresar como:

$$\mathbf{L}\mathbf{y}=\mathbf{b} \qquad \text{....(5.5)}$$

En forma matricial:

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ . & . & . & . \\ . & . & . & . \\ l_{21} & l_{21} & & 1 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ b_n \end{bmatrix} \qquad \text{....(5.6)}$$

Debido a que se trata de una matriz triangular inferior, la solución se obtiene fácilmente:

$$y_1 = b_1$$

$$y_k = b_k - \sum_{j=1}^{k-1} l_{kj} y_j \qquad k = 2,3,\dots,n \qquad \text{....(5.7)}$$

Una vez obtenido $\mathbf{y}$ se llega a:

$$\mathbf{L}^H\mathbf{a}=\mathbf{D}^{-1}\mathbf{y} \qquad \text{....(5.8)}$$

Donde, de nuevo debido a que $\mathbf{L}^H$ es una matriz triangular superior, los parámetros que dan solución al sistema se obtienen de:

$$a_n = \frac{y_n}{d_n}$$

$$a_k = \frac{y_k}{d_k} - \sum_{j=k+1}^{n} l_{jk}^{*} a_j \qquad k = n-1, n-2, \ldots, 1 \qquad \ldots(5.9)$$

Los elementos de **L** y **D** se calculan por medio de las siguientes expresiones:

$$d_1 = a_{11}$$

$$\text{para } i = 1,2,\ldots,n,$$

$$l_{ij} = \begin{cases} \dfrac{a_{i1}}{d_1} & \text{para } j = 1 \\[2ex] \dfrac{a_{ij}}{d_j} - \displaystyle\sum_{k=1}^{j-1} \frac{l_{ik} d_k l_{jk}^{*}}{d_j} & \text{para } j = 2,3,\ldots, i-1 \end{cases}$$

$$d_i = a_{ii} - \sum_{k=1}^{i-1} d_k \left| l_{ik} \right|^2 \qquad \ldots(5.10)$$

Por último, se debe calcular la Densidad de Potencia Espectral (PSD) de la señal [11]. Ésta es obtenida utilizando la expresión (5.11).

$$P_{AR}(f_n) = \frac{\sigma^2}{\left| A(f_n) \right|^2} = \frac{\sigma^2}{\left| 1 + \sum_{k=1}^{p} a[k] \cdot z^{-k} \Big|_{z=e^{j2\pi f_n}} \right|^2} \qquad \ldots(5.11)$$

El denominador de esta última expresión se evalúa sobre el rango de frecuencias normalizado de [-1/2, 1/2] en los puntos $f_n = (-1/2) + (n-1)/N$, donde $n = 1,2,\ldots,N$. Esto se logra utilizando la Transformada Discreta de Fourier o su implementación discreta, la Transformada Rápida de Fourier (FFT)[11][4].

5.2 IMPLEMENTACIÓN PARALELA DEL ESTIMADOR ESPECTRAL PARAMÉTRICO DE COVARIANCIA MODIFICADA

En esta sección se presenta la implementación de un estimador espectral paramétrico con base en el algoritmo de covariancia modificada desarrollado en [7]. Este método fue seleccionado por la resolución espectral que se obtiene al aplicarse a señales Doppler de ultrasonido[3], objeto de estudio del proyecto de investigación global que envuelve este trabajo de tesis. Debido a que el algoritmo es computacionalmente intensivo y debe ser implementado en tiempo real, se utilizaron técnicas de procesamiento paralelo. Dos implementaciones son presentadas: la primera, utilizando una arquitectura homogénea basada en una plataforma de procesadores tipo transputer T805; y la segunda en una arquitectura heterogénea conformada por una red de procesadores tipo transputer T805 y un DSP TMS320C30.

Dos implementaciones han sido hasta ahora desarrolladas en plataformas de procesamiento en paralelo. La primera considera el procesamiento de un segmento de datos haciendo una partición del algoritmo para el cálculo de la matriz de covariancia. Cada procesador esclavo es controlado por un procesador maestro (figura 5.2.1) que al mismo tiempo controla la distribución de tareas y la comunicación con el host [11][12]. La segunda toma varios segmentos consecutivos de datos, donde el espectro de cada segmento es calculado por un procesador distinto (figura 5.2.2) [3].

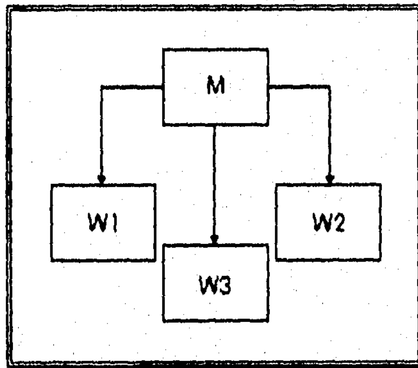


figura 5.2.1
Distribución del algoritmo
en una topología de árbol

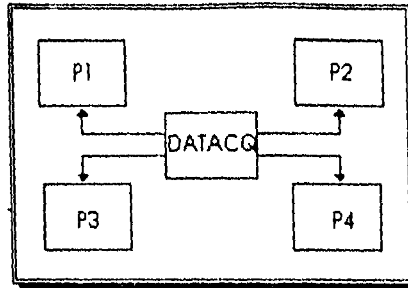


figura 5.2.2
Distribución de n ventanas
de datos en n procesadores

Ambas implementaciones fueron sobre plataformas de procesamiento paralelo conformadas por una red de transputers T800-20. Los tiempos de ejecución fueron medidos variando el orden del modelo (2, 4, 6, 8 y 10), así como el tamaño de la ventana de datos (64, 128, 256 y 512). La duración de cada segmento de datos fue de 10 ms, y tanto la frecuencia de muestreo como el largo de la secuencia de datos fueron ajustados para satisfacer la primer condición.

5.2.1 PROCESAMIENTO DE UN SÓLO SEGMENTO DE DATOS

El desempeño de uno o varios procesos ejecutándose en una arquitectura de procesamiento paralelo se mide con base en la razón de tiempo de cómputo por tiempo de comunicación R/C. Esta relación expresa si existe o no un *overhead* de comunicaciones, es decir, cuanto tiempo se requiere para la comunicación entre procesos y procesadores. Cuando este valor es muy elevado, quiere decir que el paralelismo se esta realizando de manera eficiente. Por el contrario, si es muy bajo, implica un paralelismo prácticamente ineficiente. Esta relación es también una medida de la granularidad de tareas. Cuando ésta es alta, se dice que es de granularidad gruesa, si es baja, será de granularidad fina [12].

En este caso, la implementación del estimador espectral se realizó sobre un segmento de datos. El paralelismo se lleva a cabo haciendo una partición del algoritmo y distribuyendo cada parte en distintos procesadores. Esta partición se realizó de dos formas distintas. La primera con granularidad fina y la segunda con granularidad media orientada específicamente a estimación de señales Doppler [12][11].

El programa fue escrito en lenguaje occam e implementado en una plataforma de procesamiento paralelo homogénea, utilizando una topología de árbol como se observa en la figura 5.2.

5.2.2 PROCESAMIENTO DE VARIOS SEGMENTOS DE DATOS

Esta aproximación se realizó procesando en paralelo distintos segmentos consecutivos de datos, obteniendo por separado la PSD de cada segmento. A diferencia de la aproximación anterior, cada uno de los procesadores contiene una copia del algoritmo de estimación espectral. Debido a que el cálculo de la densidad de potencia espectral de cada ventana de datos se realiza en procesadores separados, el número de segmentos procesados en paralelo dependerá totalmente del número de procesadores disponibles.

Una de las ventajas de llevar a cabo esta distribución, es que solamente se requiere comunicación de cada elemento de procesamiento con el que distribuye las ventanas de datos a cada procesador disponible [3].

Por medio de la implementación de este algoritmo se obtuvieron resultados muy interesantes, razón por la que se ha tomado como punto de partida de este trabajo de tesis. Los tiempos de procesamiento, según el orden del modelo y el tamaño de la ventana de datos, se muestran en la tabla 5.2.1. Los resultados muestran el tiempo de ejecución del estimador espectral sobre cuatro segmentos de datos en cuatro procesadores trabajando de manera paralela.

	orden del modelo				
ventana	2	4	6	8	10
64	4197	5490	6999	8916	11259
128	8970	11120	13544	16345	19572
256	19492	23416	27617	32178	37148
512	42220	49697	57432	65512	73988

tabla 5.2.1
Procesamiento de cuatro segmentos consecutivos
Tiempos de procesamiento en μs.

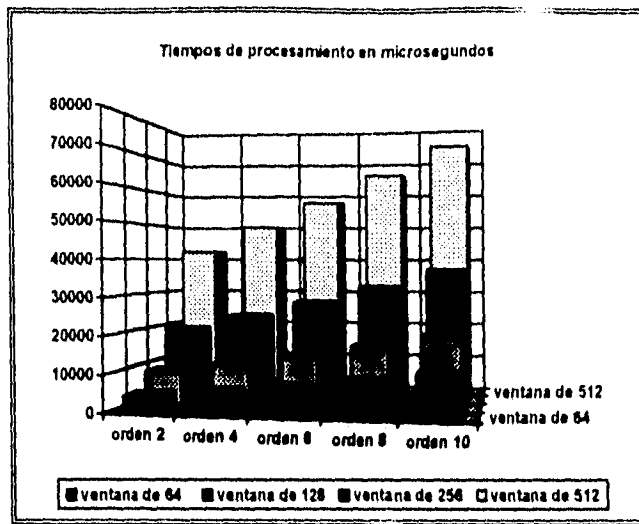


figura 5.2.3
Cuatro procesadores en paralelo

El tiempo de procesamiento de cada segmento se obtiene dividiendo el tiempo total entre el número de segmentos. Se observa que el tiempo para procesar cada ventana se reduce notablemente en comparación con el tiempo requerido para la versión secuencial del algoritmo (tabla 5.2.2). Esto se debe a que el algoritmo es distribuido de manera idéntica en cada procesador, por lo que el sistema se encuentra totalmente balanceado[3].

	orden del modelo				
ventana	2	4	6	8	10
64	16178	21960	27996	35664	45036
128	35880	44480	54176	65380	78288
256	77968	93664	110468	128712	148592
512	168880	198788	229728	262048	295952

tabla 5.2.2
Versión secuencial del algoritmo.
Tiempos de procesamiento en µs.

5.2.3 IMPLEMENTACIÓN HOMOGÉNEA UTILIZANDO UNA TOPOLOGÍA PIPELINE

Los datos reportados en las implementaciones anteriores muestran altos niveles de eficiencia y balanceo de cargas, pero la ejecución en casi todos los casos no se realiza en tiempo real (10 ms). Debido a esto se

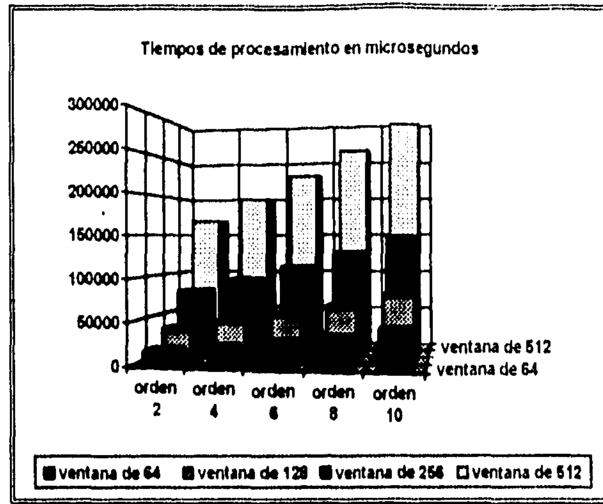


figura 5.2.4
Versión secuencial del algoritmo

buscaron nuevas alternativas para reducir este tiempo, aunque posiblemente la eficiencia fuera menor y existiera un ligero desbalanceo en el sistema.

El algoritmo de covariancia modificada, como ya se mencionó, es computacionalmente intensivo. La tabla 5.2.3 muestra el número de operaciones de punto flotante requeridas en la primera parte del algoritmo [11].

BLOQUE	FLOPS
matriz de covariancia	$3p^2N$
vector del lado derecho	$4pN$
solución del sistema (Choleski)	p^3

tabla 5.2.3
Número de operaciones de punto flotante requeridas por la primera parte del algoritmo

Una vez calculados los parámetros del método de covariancia modificada, se debe determinar tanto la varianza de ruido blanco como la FFT con el fin de obtener la densidad de potencia espectral de la señal. Al igual que en la tabla anterior, en el siguiente cuadro se muestra el número de operaciones de punto flotante necesarias para el cálculo de la PSD.

BLOQUE	FLOPS
varianza de ruido blanco	4pN
FFT	$5N\log_2 N$
PSD	N

tabla 5.2.4
Número de operaciones de punto flotante requeridas por la segunda parte del algoritmo

Para el cálculo de la FFT, es necesario realizar $(N/2)\log_2 N$ multiplicaciones complejas y $(N)\log_2 N$ sumas complejas [9]. Tomando en cuenta que cada multiplicación involucra 4 multiplicaciones y 2 sumas reales, y que cada suma compleja requiere de 2 sumas reales, en total para el cálculo de la FFT es necesario realizar $(2N)\log_2 N$ multiplicaciones y $(3N)\log_2 N$ sumas reales.

Debido al número de operaciones de punto flotante necesarias para el calculo de la PSD de la señal Doppler, no ha sido posible ejecutar el proceso en tiempo real aún cuando se distribuye el mismo algoritmo en distintos procesadores. Para solucionar este problema se tomaron en cuenta varias alternativas. Una de ellas fue dividir el algoritmo en dos bloques principales: el cálculo de los parámetros del método de covariancia modificada y el cálculo de la PSD en base a estos parámetros; cada uno de ellos en diferente procesador. En la primera parte, se calcula la matriz de covariancia, el vector del lado derecho y la resolución del sistema por el método de Choleski [7]. En la segunda parte, se determina la varianza de ruido blanco, la FFT y por último se sustituyen los valores correspondientes en el modelo de la PSD del método utilizado. Con esta división, el número de operaciones de punto flotante se divide en forma casi simétrica en dos procesadores.

Para que la segunda parte pueda comenzar a ejecutarse, es necesario que el primer procesador envíe al segundo los parámetros calculados y el primer renglón de la matriz. Esta es la razón por la que el segundo procesador no puede empezar a trabajar hasta que el primero haya terminado por completo. Para lograr tiempo real, ambos procesos deben ejecutarse en paralelo en una topología pipeline (capítulo 2) como se muestra en la figura 5.2.5. Debido a la separación del algoritmo, se tuvo que ajustar la implementación de tal forma que fuera posible la partición del mismo.

El sistema se implementó en una red de transputers T805-30, y utilizando el lenguaje C paralelo. Se utilizó este lenguaje con fines evaluativos además de que, a diferencia del lenguaje Occam, permite llevar a cabo la partición del algoritmo de covariancia modificada en forma relativamente sencilla. Otra consideración importante es que, para todos los casos, se hizo la FFT con 256 puntos, con el fin de obtener mayor resolución en el espectro en frecuencia [4].

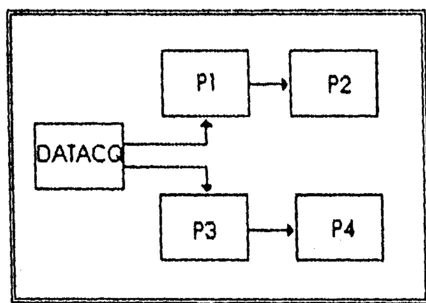


figura 5.2.5
Topología utilizada en la
implementación homogénea

En los resultados obtenidos mostrados a continuación se observa que el algoritmo presenta mejoras en tiempo de ejecución con respecto a las implementaciones anteriores.

	orden	del	modelo		
ventana	2	4	6	8	10
64	11362	11572	11846	12222	12743
128	11362	11572	11846	12222	12743
256	11362	11572	11846	12222	12743
512	11362	11572	13322	16742	20132

tabla 5.2.5
Tiempos de procesamiento en μs para una
partición del algoritmo en una topología pipeline

Debido a la topología pipeline utilizada en esta implementación, el tiempo total de procesamiento se rige en función del proceso que más tiempo requiere para su ejecución. En este caso, el proceso en el que se calculan la varianza de ruido blanco, la FFT y la PSD, es el que determina el tiempo total de procesamiento; salvo en los casos en que el tamaño de la

ventana es de 512 datos y el orden del modelo es mayor o igual que 6, donde el primer bloque es el que requiere de más operaciones de punto flotante.

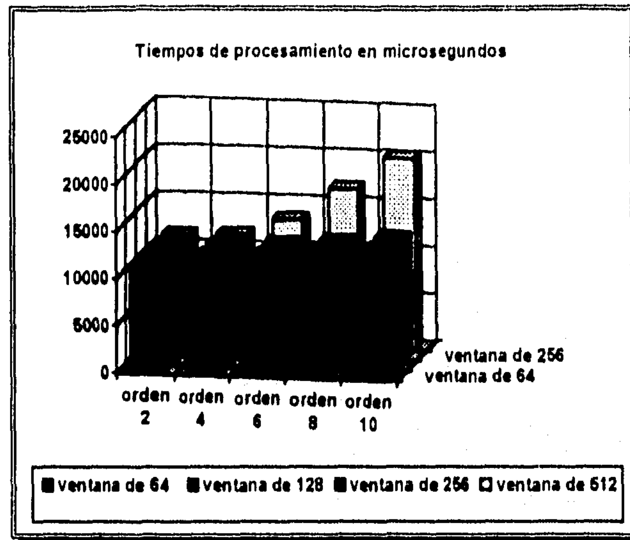


figura 5.2.6
Partición del algoritmo en una topología pipeline

5.2.4 IMPLEMENTACIÓN HETEROGÉNEA UTILIZANDO UNA TOPOLOGÍA PIPELINE

Hasta ahora, la experiencia derivada de trabajos previos de investigación en estimación espectral, ha demostrado que la utilización de una arquitectura homogénea de procesamiento paralelo ofrece ciertas ventajas al computar y coordinar, eficientemente, operaciones de procesamiento de señales consideradas de granularidad media [4][14]. Sin embargo, algunas aplicaciones, como es el caso de la estimación espectral paramétrica, requieren de operaciones regulares de granularidad fina, donde este tipo de arquitectura resulta poco eficiente [4][10]. Por esta razón se hace evidente la necesidad de incorporar un procesador con características complementarias a las del transputer.

Debido a que el transputer es un procesador de granularidad media o gruesa, se incorporó a esta arquitectura un procesador complementario, es decir, de granularidad fina [10][4]. La adición de un DSP a la red original de transputers dio origen al nodo de procesamiento heterogéneo (capítulo 4).

El nodo de procesamiento heterogéneo integra un transputer T805 y un procesador digital de señales TMS320C30. La comunicación se hace p medio de un link del transputer, por lo que los tres links restantes pueden s utilizados por otros procesadores para accesar al nodo[10][14].

Al utilizar un DSP como parte del nodo de procesamiento heterogéneo, es posible realizar en éste aquellas tareas de granularidad fina que resultan poco eficientes en el transputer. Sin embargo, existe lo que se conoce como *overhead* de comunicaciones entre el transputer y el DSP. Esto se ve reflejado en la disminución de eficiencia en la implementación, punto que será analizado en el siguiente capítulo.

Para hacer una comparación directa con la implementación homogénea realizada con anterioridad, se continuó con la misma distribución del algoritmo en una topología pipeline con igual número de elementos de procesamiento (3 procesadores transputer y un DSP) como se muestra en la figura 5.2.7.

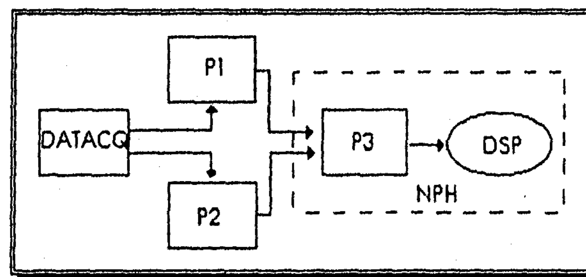


figura 5.2.7
Topología utilizada en la
Implementación heterogénea

Los tiempos de ejecución se muestran en la tabla 5.2.6.

El objetivo de la implementación heterogénea es el procesamiento en tiempo real de cualquier ventana de datos. Se observa en la tabla 5.3.6 que no todos los valores son menores a 10 ms, sin embargo el tiempo de procesamiento se redujo notablemente y solamente son cuatro los casos donde el procesamiento no se realiza en tiempo real. También se observa que el tiempo en los modelos de orden 2 y 4 de cada ventana son iguales, esto se debe a que el nodo de procesamiento heterogéneo es el que determina el tiempo máximo de procesamiento del sistema trabajando en pipeline. Para el resto, es el cálculo de la matriz de covariancia el que lo determina.

	orden	del	modelo		
ventana	2	4	6	8	10
64	5964	6235	6897	7438	8528
128	5963	6231	6694	7456	8527
256	5958	6218	6910	9007	11252
512	6040	9739	13512	17411	21438

tabla 5.2.6
Implementación heterogénea
Tiempos de procesamiento en μs

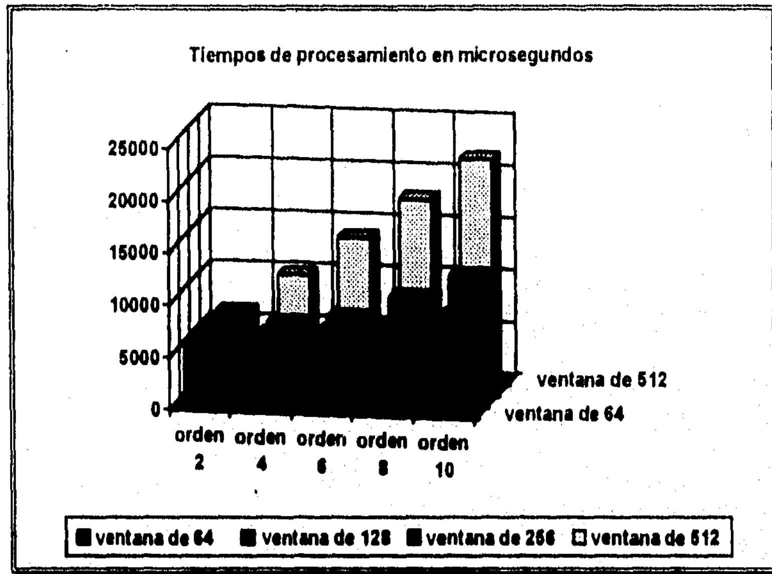


figura 5.2.8
Implementación heterogénea

Tomando en cuenta los tiempos de ejecución para cada ventana de datos, la implementación heterogénea cumple, en la mayoría de los casos, con su principal objetivo . Sin embargo, con el fin de obtener el menor tiempo de procesamiento posible, se diseñó otra aproximación añadiendo un procesador transputer T805 a la red para distribuir en tres procesadores el cálculo de la matriz de covariancia de distintas ventanas. En la figura 5.2.9 se muestra el esquema empleado.

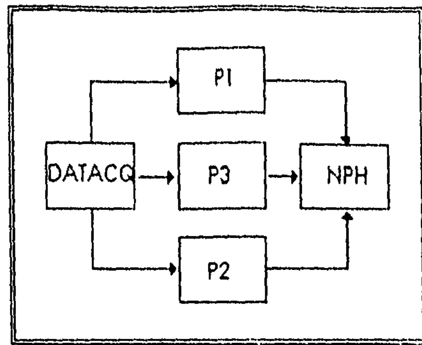


figura 5.2.9
Topología utilizada en la segunda
versión de la implementación heterogénea

Los tiempos de procesamiento (tabla 5.2.7) fueron reducidos en dos de los cuatro casos señalados anteriormente, con lo que se logró que, en el 90% de los casos, el procesamiento se realizara en tiempo real.

	orden del modelo				
ventana	2	4	6	8	10
64	5954	6016	6070	6166	6209
128	5939	6012	6066	6134	6216
256	5944	6036	6101	6162	**7221**
512	5944	6019	**8511**	**11075**	**13745**

tabla 5.2.7
Segunda implementación heterogénea
Tiempos de procesamiento en μs

En la figura 5.2.10 se observa que para cualquier orden del modelo donde la ventana es menor a 512 datos, el procesamiento se realiza en tiempo real, esto es, por debajo de los 10 ms.

En el siguiente capítulo se evalúan las dos últimas implementaciones y se comparan con las realizadas anteriormente. Este análisis se hace tomando en cuenta criterios de eficiencia, speed-up y fracción serial, para cada uno de los algoritmos propuestos.

ESTA TESIS NO DEBE
SALIR DE LA BIBLIOTECA

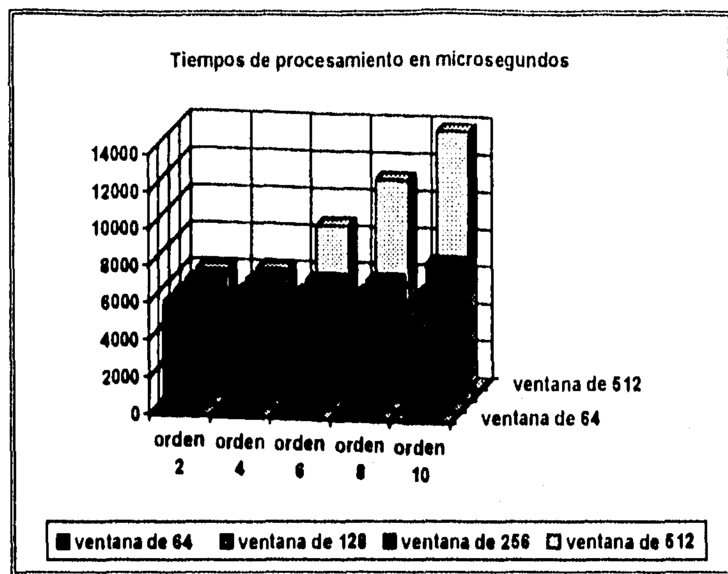


figura 5.2.10
Segunda implementación heterogénea

5.3 REFERENCIAS

[1]BURNS, Peter N., Doppler Ultrasound, IEEE Short Course, Seattle, USA, 1995.

[2]FISH, P. J., Developments in Acustics and Ultrasonics, Eds. M. J. W. Povey and D. J. McClements, IOP Publishing, Bristol, U. K.,1992.

[3]GARCIA N., D. F., SOLANO G., J., BENÍTEZ P., H., Parallel Implementation of Parametric Spectral Estimation for Doppler Blood Flow Instrumentation, The Proceedings of the 6th International Conference on Signal Processing Applications & Technology, Vol I, pp. 613-617, Boston, Massachusetts, USA, 1995.

[4]GARCIA NOCETTI, D. F., SOLANO GONZÁLEZ J., MARTÍNEZ FLORES, J., RAMOS HERNÁNDEZ, D., Heterogeneous Parallel Architecture for Improving Signal Processing Performance in Doppler Blood Flow Instrumentation, 10th International Parallel Processing Symposium, Honolulu Hawaii, USA, 1996.

[5]GIDDENS, D. P. and KITNEY, R. I., Blood flow disturbances and spectral analysis, Noninvasive diagnostic techniques in vascular sisease, Chapter 8, The C. V. mosby Company, ST. Louis-Toronto-Princeton, 1985.

[6]JOHNSTON, K. W. and KASSAM, M. S., Processing Doppler signals and analysis of peripheral :rterial waweforms: problems and solutions, Noninvasive diagnostic techniques in vascular sisease, Chapter 7, The C. V. mosby Company, ST. Louis-Toronto-Princeton, 1985.

[7]KAY, Steven M., Modern Spectral Estimation, Theory & Application, Prentice-Hall Signal Processing Series, Alan V. Oppenheim, Series Editor, USA, 1988.

[8]PAPAMICHALIS, P. E., Digital Signal Processing Applications With the TMS320 Family, Vol. 3, Ed. Prentice Hall and Digital Signal Processing Series, Texas Instruments, USA, 1990.

[9]PROAKIS, J., MANOLAKIS, D. G., Digital Signal Processing Principles, Algorithms and Applications, Second Edition, Macmillan Publishing Company, USA, 1992.

[10]RAMOS HERNÁNDEZ, D., Diseño e Implementación de un Nodo Heterogéneo de Procesamiento Digital de Señales Utilizando un Transputer y un DSP, Tesis, Facultad de Ingeniería, UNAM, México, 1995.

[11]RUANO, M. G., Investigation of Real-Time Spectral Analysis Techniques for Use with Pulsed Ultrasonic Doppler Bood-Flow Detectors, Thesis submited to the University College of North Wales, Bangor, U. K., 1992.

[12]RUANO, M. G., GARCÍA NOCETTI, D. F., FISH, P. J., FLEMING, P. J., Alternative parallel implementation of an AR-modified covariance spectral estimator for diagnostic ultrasonic blood flow studies, Parallel Computing 19, pp. 463-476, North-Holland, 1993.

[13]SOLANO GONZÁLEZ, J., GARCÍA NOCETTI, D. F., RODRÍGUEZ VÁZQUEZ, K., Parallel Genetic Algorithms in Spectral Estimation of Doppler Signals, 3rd IFAC Workshop on Algorithms and Architectures for Real-Time Control, Ostende, Bélgica, 1995.

[14]SOLANO GONZÁLEZ, J., GARCÍA NOCETTI, D. F., RODRÍGUEZ VÁZQUEZ, K., RAMOS HERNÁNDEZ, D., Parallel Genetic Algorithms in Autorregressive Modelling Using A Heterogeneous Architecture, 13th IFAC World Congress, San Francisco, USA, 1996.

[15]VAITKUS, P. J. and COBBOLD, R. S. C., A Comparative Study and Assessment of Doppler Ultrasound Spectral Estimation Techniques Part I:

Estimation Methods, Ultrasound in Med. & Biol., Vol 14, No. 8, pp. 661-672, USA, 1988.

[16]VAITKUS, P. J., COBBOLD, R. S. C. and JOHNSTON, K. W., A Comparative Study and Assessment of Doppler Ultrasound Spectral Estimation Techniques Part II: Methods and Results, Ultrasound in Med. & Biol., Vol 14, No. 8, pp. 673-688, USA, 1988.

CAPÍTULO 6

CAPÍTULO 6

ANÁLISIS DE RESULTADOS

En el presente capítulo, se analizan los resultados obtenidos al implementar el estimador espectral paramétrico de covariancia modificada, en una arquitectura heterogénea de procesamiento paralelo. Para esto, se desarrolló una aproximación homogénea del mismo algoritmo en una topología pipeline, y ambas se comparan con el esquema original presentado en la sección 5.2.

Posteriormente se realiza una comparación de la resolución que presenta esta nueva aproximación y la que se obtiene al utilizar sólo procesadores tipo transputer; además, se determinan las ventajas que presenta y se medirá su desempeño, para lo cual es necesario definir tres conceptos muy útiles para la evaluación del esquema presentado: eficiencia, *speed-up* y fracción serial.

6.1 *SPEED-UP*, EFICIENCIA Y FRACCIÓN SERIAL

Estos conceptos son los tres aspectos esenciales para determinar el grado de aprovechamiento de los recursos de una arquitectura de procesamiento paralelo cuando se están ejecutando uno o varios procesos en una red de procesadores.

El término *speed-up* hace referencia al tiempo que requiere un procesador para ejecutar secuencialmente una tarea, entre el tiempo que requieren p procesadores para llevar a cabo en paralelo el mismo proceso.

$$s = \frac{T(1)}{T(p)} \qquad \text{....(6.1)}$$

La eficiencia de un algoritmo es el cociente que resulta de dividir el tiempo de ejecución del proceso secuencial, entre el tiempo que

requieren varios procesadores, multiplicado por el número de procesadores que intervienen, esto es, el *speed-up* entre el número de procesadores, y se define de la siguiente manera:

$$e = \frac{T(1)}{pT(p)} = \frac{s}{p} \qquad .... (6.2)$$

Cuando la eficiencia es cercana a la unidad, quiere decir que se está aprovechando el hardware de manera eficiente. Si por el contrario, es un número más pequeño, indicaría que no se están aprovechando las ventajas que ofrece una arquitectura de procesamiento paralelo[4].

De lo anterior se deduce que el incremento en el número de procesadores implica una reducción en el tiempo de ejecución. El *speed-up* es el que indica qué tanto se reduce el tiempo al añadir procesadores. Cuando es lineal, quiere decir que se está paralelizando el algoritmo o los datos de manera correcta. Cuando se tiene el caso de una eficiencia baja y un *speed-up* no lineal, es necesario calcular un tercer parámetro: la fracción serial.

Un algoritmo no siempre se puede paralelizar por completo. Existe siempre una parte del programa que se necesita ejecutar de manera secuencial. De aquí, la fracción serial se define como sigue [4]:

$$T(p) = T_s + \frac{T_p}{p} \qquad .... (6.3)$$

donde T_s es el tiempo de ejecución de la parte serial y T_p el tiempo que se requiere para procesar la parte paralelizable del algoritmo.

Si $p=1$, entonces la ecuación (6.3) puede expresarse de la siguiente manera:

$$T(1) = T_s + T_p \qquad .... (6.4)$$

Por otro lado, la fracción serial se define como:

$$f = \frac{T_s}{T(1)} \qquad \text{.... (6.5)}$$

y sustituyendo (6.4) y (6.5) en (6.3)

$$T(p) = T(1)f + \frac{T(1)(1-f)}{p} \qquad \text{.... (6.6)}$$

que en términos de *speed-up*

$$\frac{1}{s} = f + \frac{1-f}{p} \qquad \text{.... (6.7)}$$

y finalmente, resolviendo para **f**

$$f = \frac{1/s - 1/p}{1 - 1/p} \qquad \text{.... (6.8)}$$

siendo esta última expresión la que define a la fracción serial como una función del *speed-up* y el número de procesadores [4].

Cuando se hace una distribución de ventanas de datos en varios procesadores, existe un balanceo de cargas en el sistema, ya que cada unidad de procesamiento cuenta con exactamente el mismo código. Cuando se hace una división del algoritmo y se distribuye en distintos procesadores, es difícil hacer una distribución totalmente equitativa. Cuando no es posible realizar esta distribución, el valor de *speed-up* se reduce notablemente y la fracción serial se incrementa de igual manera. Los efectos de un desbalanceo de cargas se relaciona directamente con un cambio irregular en la fracción serial conforme el número de procesadores aumenta. Cuando existe un desbalanceo total de cargas, la fracción serial será cercana a 1, indicando que existe un problema en la distribución, de tareas, mismo que no es posible detectar al calcular la

eficiencia o *speed-up* [4].

Otro problema que se puede detectar utilizando la fracción serial, es la existencia de un *overhead* de comunicación entre procesadores, que es una función monótona creciente de **p** (número de procesadores). Conforme aumenta dicho *overhead*, el *speed-up* se reduce provocando un ligero incremento en la fracción serial cada vez que se añade un procesador al sistema. Cuando esto se presenta, se dice que la granularidad de las tareas o procesos es muy fina. Por último, cuando el valor de **f** es constante, y la eficiencia es baja, se debe a que el algoritmo ha llegado a su límite de paralelización.

6.2 RESOLUCIÓN

Distintos trabajos de investigación en estimación espectral [2][5][6], coinciden en la utilización de métodos paramétricos para obtener la mayor resolución del espectro en frecuencia. Esto se confirma con la aproximación homogénea presentada en el capítulo anterior, donde la resolución es mejor que la obtenida con los métodos convencionales [2]. El esquema heterogéneo presentado en el mismo capítulo, involucra la adición de un nodo de procesamiento heterogéneo, con la finalidad de reducir los tiempos de ejecución obtenidos en la implementación homogénea. Los tiempos de procesamiento se reducen notablemente y la resolución obtenida en la primer aproximación se conserva. La razón de esto es que el mismo algoritmo para el cálculo de la FFT se utiliza en ambos casos; sin embargo, al ser el DSP un procesador diseñado específicamente para realizar tareas de procesamiento de señales [3], es posible procesar la señal en tiempo real en la mayoría de los casos.

Tomando el caso de una ventana de 256 datos, en la figura 6.2.1 se muestra el espectro en frecuencia que se obtiene utilizando ambos esquemas variando el orden del modelo (2, 4, 6, 8 y 10). Las figuras de la izquierda corresponden a la aproximación heterogénea y las de la derecha a la homogénea. Se observa que para todos los casos, la resolución se conserva aún con la adición del nodo de procesamiento heterogéneo.

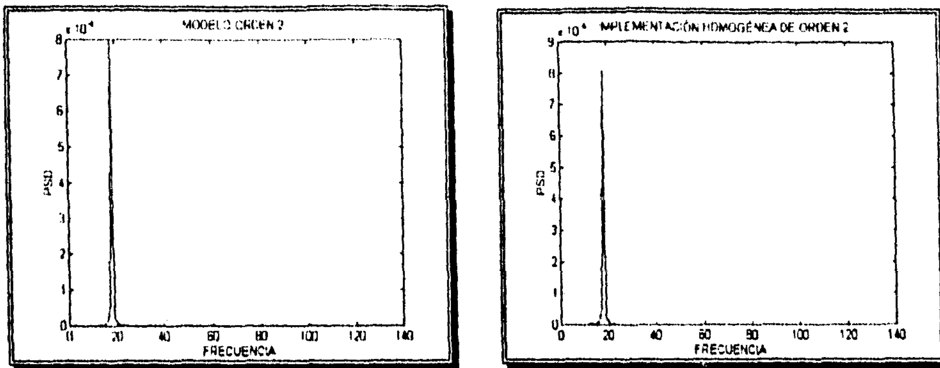


figura 6.2.1a
Modelo de orden 2

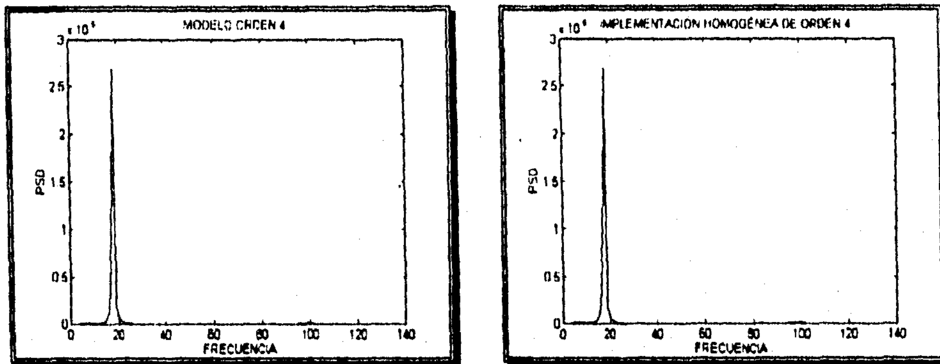


figura 6.2.1b
Modelo de orden 4

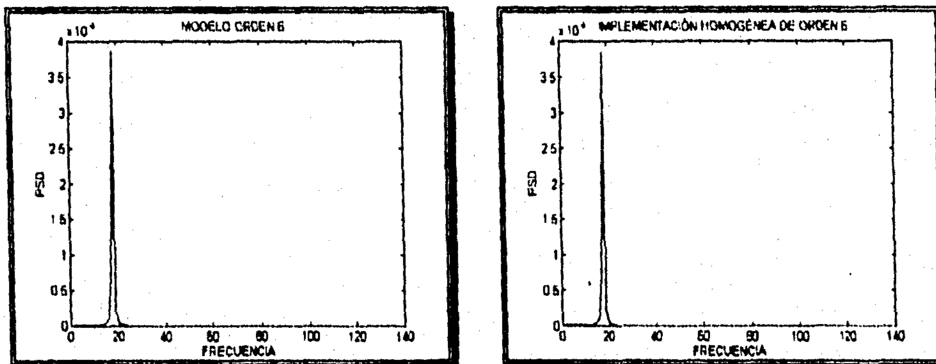


figura 6.2.1c
Modelo de orden 6

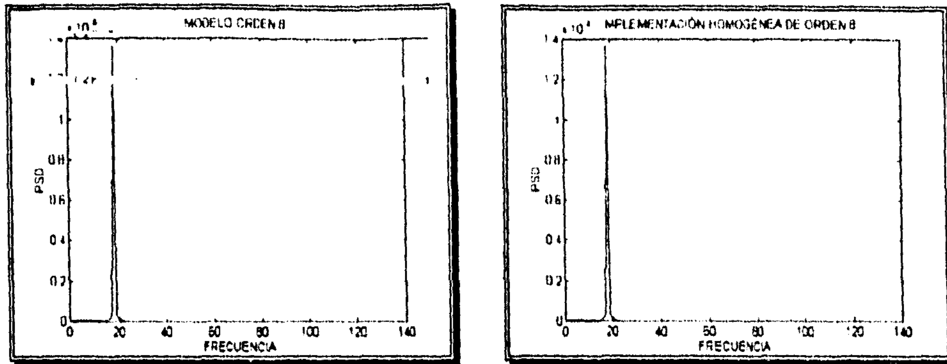


figura 6.2.1d
Modelo de orden 8

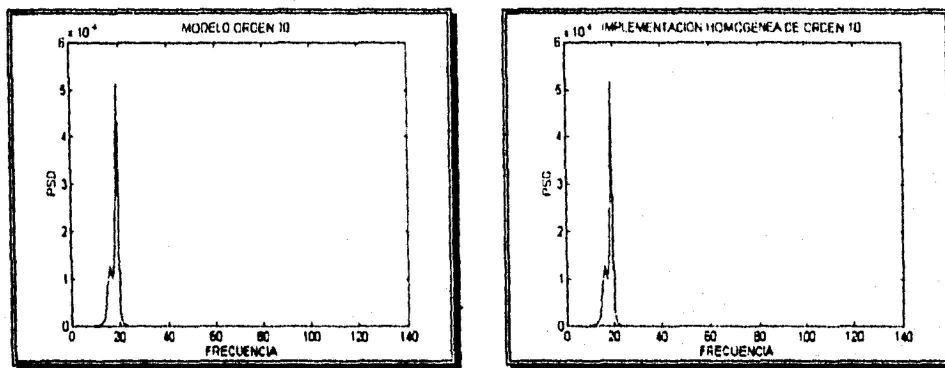
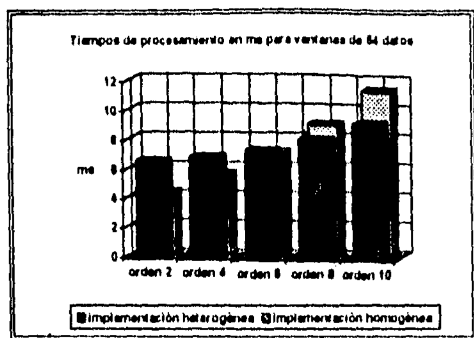


figura 6.2.1e
Modelo de orden 10

6.3 TIEMPOS DE PROCESAMIENTO

Uno de los objetivos principales de este trabajo de tesis, es lograr procesamiento en tiempo real al implementar un algoritmo paramétrico de estimación espectral en una arquitectura heterogénea de procesamiento paralelo. Con la adición del nodo de procesamiento heterogéneo, se lograron reducir los tiempos de ejecución en todos los casos logrando, en la mayoría de ellos, tiempo real. En la figura 6.3.1 se hace una comparación directa entre los dos esquemas. En el caso de una ventana de 64 datos, los tiempos que se obtienen son mayores cuando se utiliza un modelo de orden 2, 4 ó 6, sin embargo para modelos de orden 8 y 10, el tiempo de ejecución disminuye. Para las ventanas restantes (128, 256 y 512 datos), el tiempo de ejecución utilizando el NPH, es menor que el esquema homogéneo original.



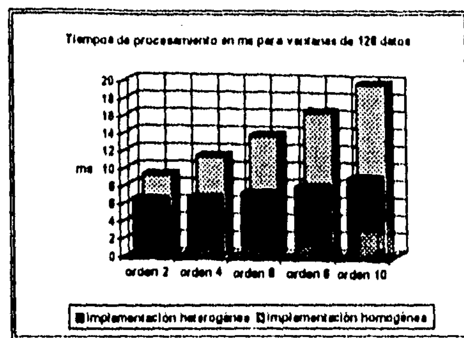
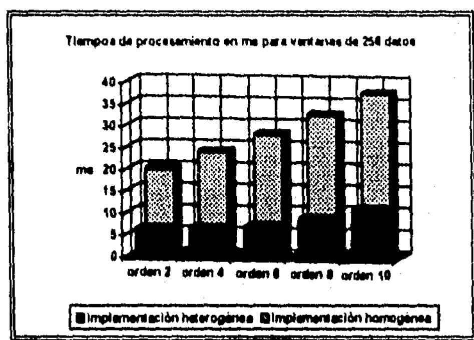


figura 6.3.1a　　　　figura 6.3.1b

Comparación de los tiempos de procesamiento



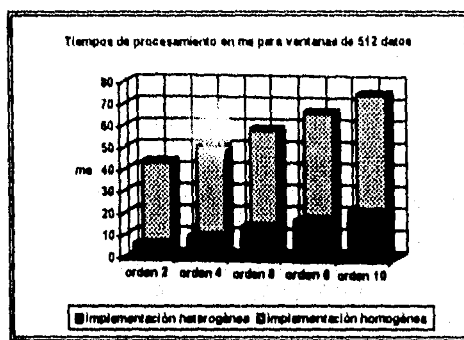


figura 6.3.1c　　　　figura 6.3.1d

Comparación de los tiempos de procesamiento

Esta disminución en los tiempos de ejecución se debe a que el cálculo de la PSD, se realiza en el DSP. En el esquema original, el cálculo de la FFT, toma entre 21 ms y 22.5 ms, según el orden del modelo y el tamaño de la ventana de datos; mientras que en el DSP, este cálculo toma entre 5 y 6.5 ms. Además, la partición del algoritmo permite que el cálculo de la PSD se lleve a cabo utilizando una topología pipeline.

La implementación homogénea original utiliza N módulos de procesamiento para N ventanas consecutivas de datos [2][3]. El espectro de cada segmento de datos es calculado por completo en un sólo procesador. Los tiempos de ejecución, así como el tiempo de adquisición y despliegue de datos se muestran en la figura 6.3.2, tomando como base un segmento de datos de 256 muestras y utilizando un modelo de orden 6. En este caso, la adquisición de los datos se realiza cada 10 ms y cada ventana se distribuye en un procesador distinto, el cual, al concluir el cálculo de la densidad de potencia espectral, envía los valores obtenidos a otro procesador que se encarga del despliegue.

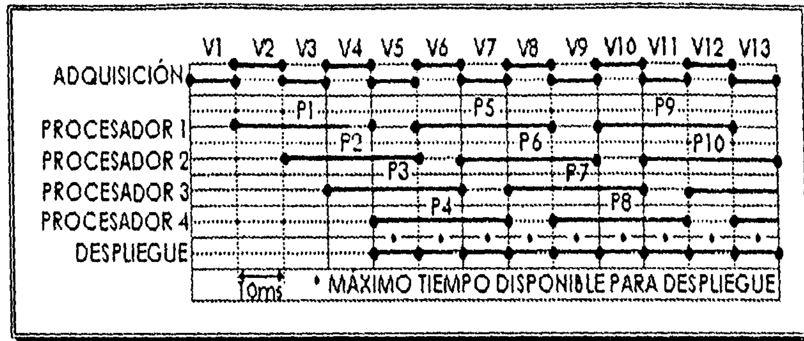


figura 6.3.2
Diagrama de tiempos de la
Implementación homogénea

Se observa que el procesamiento de cada ventana de datos se realiza en aproximadamente 50 ms. Por lo que es necesario distribuir los datos en cuatro procesadores, para que de esta manera, el tiempo de ejecución por segmento de datos sea de 10 ms para este caso en particular.

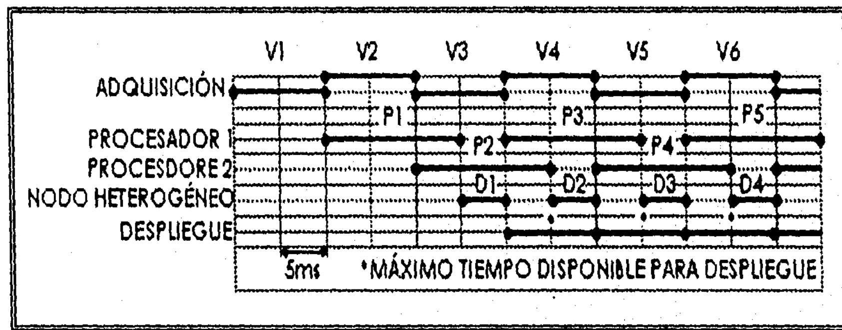


figura 6.3.3
Diagrama de tiempos de la
Implementación heterogénea

En la figura 6.3.3, se presenta el diagrama de tiempos del esquema heterogéneo, en el cual se observa como trabajan los procesadores y el nodo de procesamiento heterogéneo en pipeline. Al tomar la primer ventana de datos, ésta se manda a un procesador (P1), el cual hace el cálculo de la matriz de covariancia y de los parámetros del método. De igual manera, se envía la ventana siguiente a un segundo procesador (P2) que realiza las mismas funciones que el primero, al terminar éste de hacer el cálculo, envía sus resultados al nodo heterogéneo, donde se hace la FFT y el cálculo final de la PSD de la señal; una vez que manda los datos al

nodo, está listo para recibir otra ventana de datos tan pronto como ésta se encuentre disponible. El nodo envía sus resultados a un último procesador encargado del despliegue del espectro en pantalla. Mientras esto se lleva a cabo, el segundo procesador procesa otro segmento de datos y, cuando finalmente determina los parámetros, los envía al nodo, el cual está libre debido a que trabaja a una velocidad mayor que la de los dos procesadores y puede comenzar con el cálculo de la PSD de este segmento. El sistema continúa trabajando de esta manera, procesando varios segmentos de datos consecutivos, en una topología pipeline. Todo esto, reduce el tiempo ocioso de cada procesador en comparación con el esquema homogéneo.

En general, en la aproximación heterogénea se reduce el tiempo de procesamiento de la FFT y PSD, sin embargo, cuando el tiempo requerido para el cálculo de la matriz de covariancia es mayor que el de la FFT, el tiempo total de procesamiento se incrementa conforme aumenta el orden del modelo y el tamaño de la ventana de datos, el cual se rige por el tiempo requerido para el cálculo de los parámetros del método (figura 6.3.4).

En las figuras 6.3.4, se observa que el tiempo total de procesamiento, utilizando el esquema heterogéneo con una topología pipeline, se rige por el tiempo que toma el cálculo de la FFT y PSD en el nodo heterogéneo, para ventanas de 64 y 128 datos para cualquier orden del modelo. En el caso de ventanas de 256 y 512 datos, el tiempo lo determina el cálculo de los parámetros del método de covariancia modificada utilizando modelos mayores a 6 para el primer caso, y de orden 2 para el segundo.

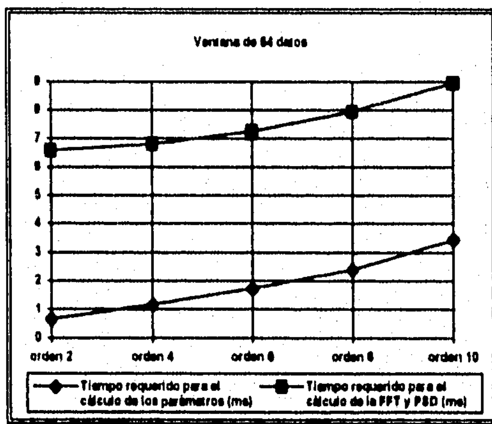


figura 6.3.4a

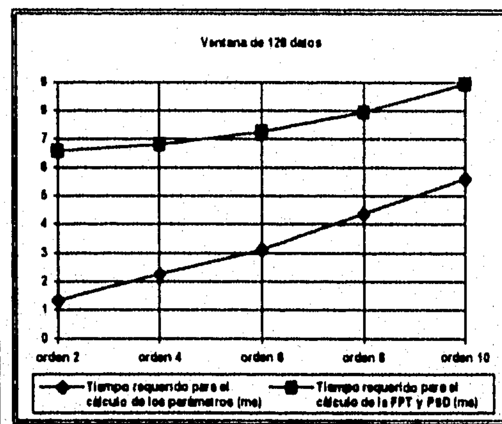


figura 6.3.4b

Tiempos de procesamiento de las distintas partes del algoritmo para distintas ventanas de datos

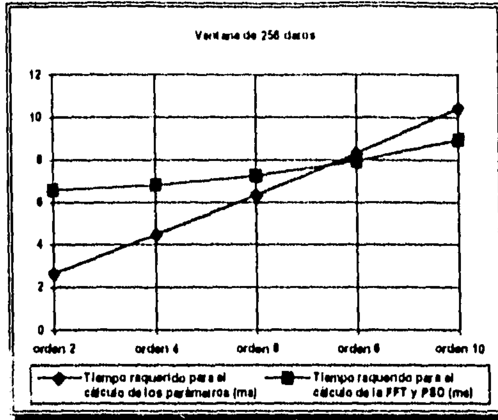


figura 6.3.4c

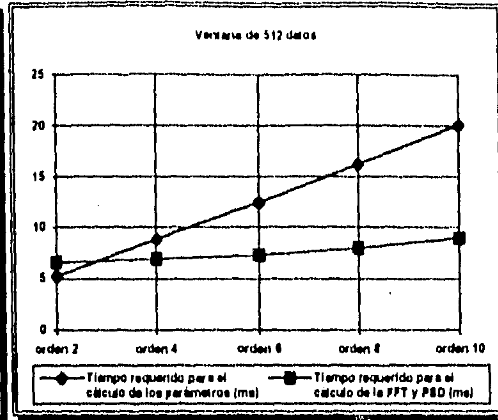


figura 6.3.4d

Tiempos de procesamiento de las distintas partes del algoritmo para distintas ventanas de datos

En el capítulo 5 se mostró una variante del esquema heterogéneo original, éste distribuye tres ventanas de datos en tres procesadores distintos, para que posteriormente, el NPH haga el cálculo de la FFT y PSD para cada segmento de datos. Esto, se refleja en una disminución en los tiempos de ejecución, pero, tiene el inconveniente de que el tiempo ocioso de cada procesador se incrementa ligeramente; además, para ventanas de datos menores a 256, el sistema presenta un desbalanceo de cargas. Esto último se observa cuando se determina la eficiencia y la aceleración del sistema (*speed-up*), y se ve aún más claro al realizar el cálculo de la fracción serial. Las figuras 6.3.5, presentan una comparación del tiempo de procesamiento para la obtención de los parámetros del método, con el tiempo requerido para el cálculo de la FFT y PSD en el nodo de procesamiento heterogéneo.

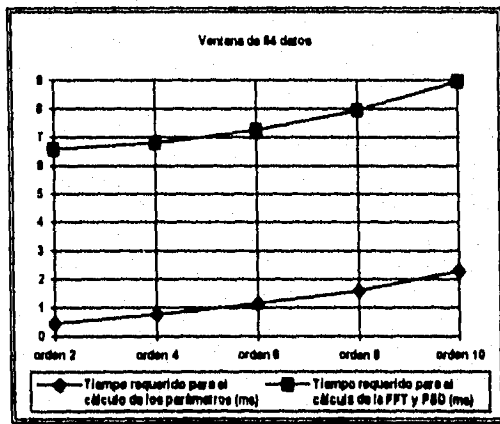


figura 6.3.5a

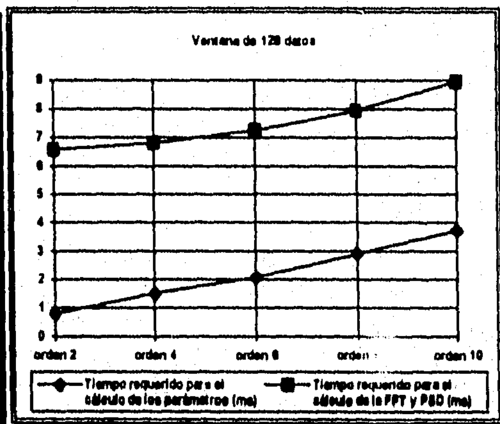
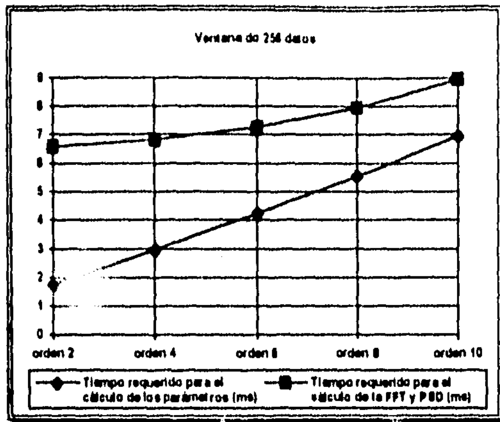


figura 6.3.5b



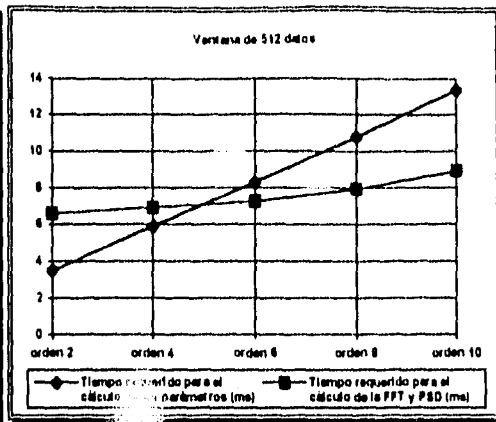


figura 6.3.5c figura 6.3.5d

Tiempos de procesamiento de las distintas partes del algoritmo para distintas ventanas de datos (segunda versión de la aproximación heterogénea)

Por la diferencia entre los tiempos de procesamiento, se observa que para esta variante del esquema heterogéneo, el sistema es más eficiente para ventanas de 256 y 512 datos, pero se hace más ineficiente para ventanas de 64 y 128 datos.

6.4 DESEMPEÑO

El esquema homogéneo original, fue implementado en una red de procesadores en una topología de árbol, donde los datos se distribuyen en varios procesadores, los cuales poseen el mismo código y algoritmo, para que una vez obtenida la PSD, se envíe a otro procesador encargado del despliegue de la señal. El esquema que se propone en este trabajo contempla dos puntos esenciales: el primero, la partición del algoritmo en sus partes principales, para de este modo, ejecutar el proceso en pipeline; el segundo, contempla la adición de un nodo de procesamiento heterogéneo. Ambas modificaciones al esquema original, se llevan a cabo con el fin de reducir el número de nodos de procesamiento para lograr ejecución en tiempo real.

En las secciones anteriores, se demostró que con el mismo número de elementos de procesamiento, el tiempo de ejecución se reduce en casi todos los casos. Esto se debe principalmente a que el algoritmo fue implementado en pipeline, permitiendo la adición de un nodo de

procesamiento heterogéneo a la red original de procesadores tipo Transputer.

Las topologías de pipeline, presentan algunas limitaciones en comparación con otros modelos. Una de estas limitantes, es la comunicación secuencial necesaria entre los módulos de procesamiento que intervienen. Debido a que los datos se transmiten de un procesador a otro, ninguno de estos puede proceder hasta que el anterior ha completado su tarea y enviado los datos al siguiente. Esto implica que, si a algún procesador se le asigna menor cantidad de trabajo computacional, aun cuando termine de ejecutar las tareas asignadas, tendrá que esperar al procesador que más tiempo requiera para realizarlas. Con el fin de garantizar la eficiencia al utilizar este tipo de arreglo de procesadores, es necesario que se le asigne a cada uno de ellos la misma cantidad de trabajo. Si ésta es menor para alguno de ellos, el sistema trabajará a menor velocidad y sería difícil lograr la sincronización entre procesos [1].

La eficiencia de una arquitectura de procesamiento paralelo en pipeline, depende principalmente de la distribución del algoritmo en los procesadores que conformen el pipeline. El programa debe dividirse en el mayor número de partes posible, y cada una de ellas debe requerir del mismo trabajo computacional.

El algoritmo de estimación espectral utilizado para determinar la PSD de la señal, se dividió en dos partes básicamente, tal como se vió en el capítulo anterior. Con esto, el algoritmo de covariancia modificada llega a un límite de separación, lo cual no permite una mayor paralelización.

El esquema heterogéneo, presentó el menor tiempo de procesamiento, comparado con las otras aproximaciones, sin embargo, debido a que la implementación se llevó a cabo utilizando una topología pipeline, se esperaba una reducción en la eficiencia del sistema [1], tal como se muestra en la tabla 6.4.1.

	orden del modelo				
ventana	2	4	6	8	10
64	0.444	0.490	0.534	0.572	0.600
128	0.517	0.608	0.687	0.747	0.789
256	0.665	0.844	0.962	0.911	0.884
512	0.947	0.839	0.795	0.774	0.764

tabla 6.4.1
Eficiencia del sistema

En la figura 6.4.1, se presentan esquemáticamente los valores de eficiencia para cada caso. Para una ventana de 64 datos, la eficiencia es inferior al 0.5 para cada orden del modelo. De igual manera se comporta el sistema cuando se tienen 128 datos por ventana; aunque en este caso, el valor de eficiencia es cada vez mayor, según se incrementa el orden. Para 256 datos, se llega a un máximo de eficiencia para orden 6, a partir de este punto, el valor decrece. Por último, para 512 datos, el sistema presenta su mayor eficiencia cuando se tiene el modelo de menor orden, punto en el que decrece conforme el orden del modelo aumenta.

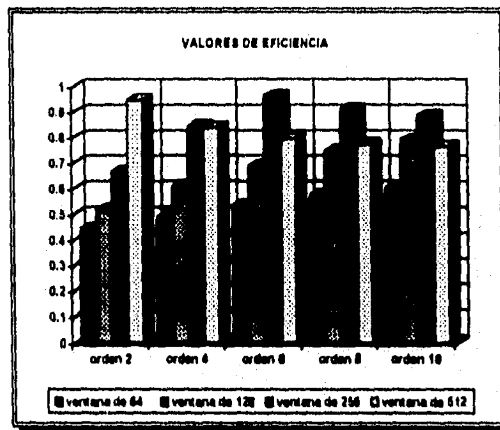


figura 6.4.1
Eficiencia del sistema

Como se mencionó anteriormente, el punto máximo de eficiencia de la aproximación heterogénea, lo encontramos para el caso de una ventana de 256 datos y un orden del modelo igual a 6. Es precisamente en este caso cuando el sistema se encuentra mejor balanceado.

Estos valores, normalmente indicarían que se están desaprovechando los recursos de la arquitectura de procesamiento paralelo, pero no es posible obtener tal conclusión únicamente al determinar los distintos valores de eficiencia. Para esto, es necesario determinar la fracción serial.

Cuando se tiene un programa corriendo en una topología pipeline, existe la desventaja de que no es posible que cada uno de los elementos de la red de procesadores comience a trabajar inmediatamente. Debido a que los datos son procesados por cada elemento de manera secuencial, un procesador particular de la red, no comenzará a trabajar hasta que los anteriores hayan terminado de realizar el procesamiento

sobre el primer segmento de datos. El problema de "llenar" el pipeline, se refleja en una reducción en la eficiencia del sistema [1]. Además, debido a la cantidad de datos enviados de un procesador a otro, el tiempo de procesamiento total, se ve afectado por un *overhead* de comunicación. En la tabla 6.4.2 se muestran los valores de fracción serial para las distintas ventanas de datos.

	orden del modelo				
ventana	2	4	6	8	10
64	0.751	0.681	0.624	0.583	0.556
128	0.644	0.549	0.485	0.446	0.423
256	0.501	0.395	0.346	0.366	0.377
512	0.352	0.397	0.419	0.431	0.436

tabla 6.4.2
Fracción serial

Se observa que para una ventana de 64 datos, la fracción serial es mayor a 0.5 en todos los casos. Debido a que se utilizan tres elementos de procesamiento, la fracción serial muestra que existe un desbalanceo de cargas, es decir, al menos un procesador, realiza más de la mitad del trabajo computacional, mientras que los otros dos hacen el resto. Esto se debe en gran parte a que, para este caso en particular, el NPH requiere de más tiempo de cómputo que los dos procesadores encargados del cálculo de la matriz de covariancia, tal y como se observa en la figura 6.3.4a.

Cuando se utiliza un segmento de 128 datos, este valor, es también mayor a 0.3 en todos los casos, sin embargo, existe una disminución en la fracción serial conforme el orden del modelo se incrementa. Esta disminución se debe a que el trabajo se distribuye de mejor manera que en el caso anterior. La diferencia en tiempo de procesamiento entre el NPH y los otros dos procesadores, es menor; y como se observa en la figura 6.3.4b, el tiempo de procesamiento de la FFT y PSD se incrementa en menor proporción que el requerido para el cálculo de los parámetros del filtro.

Para una ventana de 256 datos, se presenta el mejor balanceo del sistema. Esto se observa en la tabla 6.4.2, en la que para un modelo de orden 2, la fracción serial es igual a 0.501, pero para un modelo de orden 4, 6, 8 y 10, este valor varía entre 0.346 y 0.395. Esto indica que existe una distribución casi equitativa entre los tres procesadores. En la figura 6.3.4c se

observa que el tiempo que requiere el NPH para el cálculo de la FFT y PSD es casi igual al requerido por los otros procesadores para el cálculo de la matriz de covariancia y la resolución del sistema por Choleski. De hecho, ambas gráficas se intersecan entre un modelo de orden 6 y uno de orden 8. Esto nos indica que el sistema se encuentra mejor balanceado en ese punto.

Con un segmento de 512 dat, la fracción serial se incrementa conforme lo hace el orden del modelo. Sin embargo, este valor tiende a estabilizarse y se aproxima a un valor de 0.43. Esto indica que el sistema esta bien balanceado, pero debido a que el número de datos que se transmiten es mayor que en los casos anteriores, existe un ligero *overhead* de comunicaciones, reflejándose en una disminución en la eficiencia del sistema. En la figura 6.4.2, se observa claramente que el valor de fracción serial para todos los casos, tiende a estabilizarse, por lo que el sistema se encuentra mejor balanceado conforme el orden del modelo aumenta.

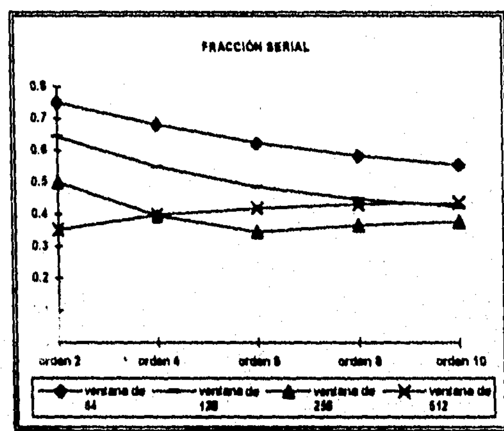


figura 6.4.2
Fracción serial

Una variante del esquema heterogéneo fue presentada en el capitulo 5, donde los valores de eficiencia obtenidos en él se presentan en la tabla 6.4.3. En ésta, se observa un incremento en la eficiencia en la mayoría de los casos. Para un modelo de orden 2, la eficiencia se incrementa conforme lo hace el tamaño de la ventana. Lo mismo ocurre para modelos de orden 4 y 6. En los últimos dos casos, la eficiencia se incrementa junto con el número de datos por segmento, hasta que se tiene una ventana de 256 datos, punto en el que el valor decrece de manera poco significativa.

	orden	del	modelo		
ventana	2	4	6	8	10
64	0.333	0.381	0.442	0.517	0.618
128	0.390	0.471	0.569	0.681	0.811
256	0.500	0.652	0.818	0.998	0.998
512	0.722	0.995	0.947	0.912	0.894

tabla 6.4.3
Eficiencia del sistema

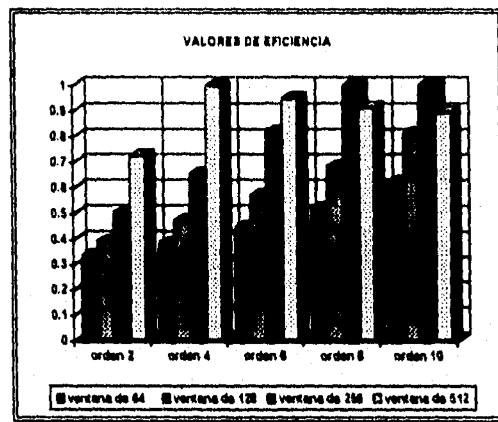


figura 6.4.3
Eficiencia del sistema (segunda versión de la aproximación heterogénea)

Los valores de eficiencia indican que en esta variante del modelo heterogéneo se están aprovechando mejor los recursos de la computadora. Sin embargo, para modelos de orden inferior a 6, la eficiencia es baja en comparación con los otros casos. Para determinar las razones, se hace un análisis de la fracción serial, de la misma manera que se realizó para el esquema original.

	orden	del	modelo		
ventana	2	4	6	8	10
64	0.750	0.657	0.566	0.483	0.405
128	0.642	0.530	0.439	0.367	0.308
256	0.500	0.383	0.306	0.250	0.250
512	0.346	0.250	0.264	0.274	0.280

tabla 6.4.4
Fracción serial

La fracción serial para un modelo de orden 2, utilizando tres procesadores Transputer T805 y un NPH, es mayor o igual a 0.5 cuando se

utilizan segmentos de 64, 128 y 256 datos. Este valor nos indica que el esquema empleado, no es el adecuado al menos para este modelo en particular, ya que está muy alejado del valor de 0.25, el cual indicaría un aprovechamiento óptimo de los recursos, además de una distribución ideal del algoritmo en los cuatro módulos de procesamiento. Cuando se utiliza un segmento de 512 datos, el valor es más cercano al ideal, sin embargo todavía existe una pequeña diferencia, la cual probablemente se deba al gran número de datos que se transmiten de un procesador a otro, provocando un ligero *overhead* de comunicación.

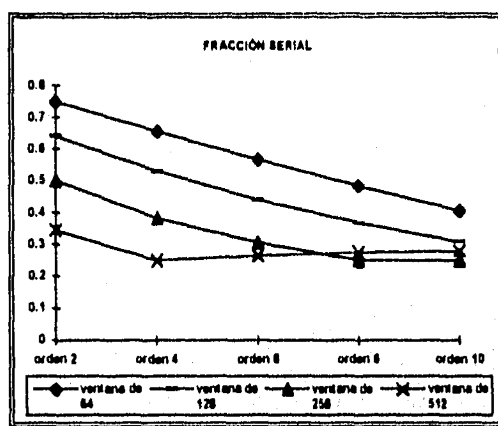


figura 6.4.4
Fracción serial (segunda versión de la aproximación heterogénea)

Para modelos de orden 4 y 6, la fracción serial disminuye constantemente, reflejando cada vez una mejor distribución de tareas conforme el tamaño de la ventana aumenta. Esto se debe a que el tiempo que se requiere para el cálculo de los parámetros, es cada vez mayor, llegando a igualar el tiempo requerido para la FFT y PSD en el NPH. Cuando se utiliza un modelo de orden 4 ó 6 y un segmento de datos mayor a 256, se tiene la mejor distribución del algoritmo además de que, el tiempo empleado para comunicación entre procesadores, no resulta significativo.

Para modelos de orden 8 y 10, el valor de fracción serial tiene la misma tendencia descendente, además de que los valores obtenidos, son muy similares. Esta variante del esquema heterogéneo, resulta casi ideal para estos dos casos, en particular para ventanas mayores a 64 datos, donde el valor resulta un poco más elevado, pero de manera poco significativa.

6.5 REFERENCIAS

[1] COCK, R., Parallel Programs for the Transputer, Prentice Hall, Englewood Cliffs, N. J., USA, 1991.

[2]GARCIA NOCETTI, D. F., SOLANO GONZÁLEZ, J., BENÍTEZ PÉREZ, H., Parallel Implementation of Parametric Spectral Estimation for Doppler Blood Flow Instrumentation, The Proceedings of the 6th International Conference on Signal Processing Applications & Technology, Vol I, pp. 613-617, Boston, Massachusetts, USA, 1995.

[3]GARCIA NOCETTI, D. F., SOLANO GONZÁLEZ J., MARTÍNEZ FLORES, J., RAMOS HERNÁNDEZ, D., Heterogeneous Parallel Architecture for Improving Signal Processing Performance in Doppler Blood Flow Instrumentation, 10th International Parallel Processing Symposium, Honolulu Hawaii, USA, 1996.

[4]KARP, A., FLATT, P., Measuring Parallel Processor Performance, Communications of the ACM, Vol. 33, Number 5, May 1990.

[5]RUANO, M. G., Investigation of Real-Time Spectral Analysis Techniques for Use with Pulsed Ultrasonic Doppler Bood-Flow Detectors, Thesis submited to the University College of North Wales, Bangor, U. K., 1992.

[6]RUANO, M. G., GARCÍA NOCETTI, D. F., FISH, P. J., FLEMING, P. J., Alternative parallel implementation of an AR-modified covariance spectral estimator for diagnostic ultrasonic blood flow studies, Parallel Computing 19, pp. 463-476, North-Holland, 1993.

CAPÍTULO 7

CAPÍTULO 7

CONCLUSIONES GENERALES

7.1 CONCLUSIONES

En este trabajo de tesis se presentó un nuevo esquema de procesamiento paralelo en estimación espectral de señales aleatorias, utilizando una arquitectura heterogénea. Con base en la implementación y evaluación del esquema propuesto, es posible afirmar que se cumplieron los objetivos planteados en el capítulo 1.

Se implementó el algoritmo de estimación espectral de covariancia modificada en una arquitectura heterogénea de procesamiento paralelo. Se utilizó una topología pipeline, lo que permitió que se redujeran los tiempos de procesamiento de las aproximaciones anteriores sin perder la resolución en frecuencia que se obtiene al utilizar los métodos paramétricos. Sin embargo, debido a que estos algoritmos requieren de un gran número de operaciones de punto flotante, el tiempo de procesamiento obtenido hasta ese momento no cumplía con los requerimientos propios de la implementación. Se decidió dividir el algoritmo distribuyendo el cálculo de la PSD en dos procesadores, calculando los parámetros del método en el primero y la FFT en el segundo. Aun cuando los tiempos de procesamiento se redujeron notablemente, no fue posible procesar la señal Doppler en tiempo real.

Se incorporó un procesador digital de señales (DSP) a la plataforma de procesamiento paralelo dando origen a una arquitectura heterogénea. Los resultados que se obtuvieron fueron satisfactorios ya que se logró una reducción en el tiempo de ejecución; sin embargo, la eficiencia del sistema se redujo de manera drástica. Se determinó que esta reducción se debía al tipo de comunicación serial entre el transputer y el DSP que conforman el nodo de procesamiento heterogéneo.

La arquitectura heterogénea demostró, a pesar de su baja eficiencia, ser una alternativa adecuada para el procesamiento en tiempo real de señales Doppler (y en general cualquier tipo de señal aleatoria); siendo la principal ventaja que ofrece este esquema, la reducción de elementos de procesamiento necesarios para lograr procesamiento en

tiempo real. En [1] se presenta una evaluación del esquema propuesto en este trabajo de tesis, resaltando la posibilidad de extender este tipo de arquitectura a cualquier tipo de aplicación de procesamiento de señales obteniendo un buen desempeño con un número reducido de elementos de procesamiento.

Los resultados obtenidos al implementar el estimador espectral de covariancia modificada utilizando un esquema heterogéneo de procesamiento paralelo, cumplen con el objetivo inicial de obtener la densidad de potencia espectral de la señal Doppler en tiempo real. Las ventajas han sido ya mencionadas, siendo la única desventaja que presenta, el hecho de que el NPH no permite un aprovechamiento óptimo de los recursos que ofrece este tipo de arquitectura, sin embargo, la reducción en el número de nodos de procesamiento es muy importante en este tipo de esquemas.

Por último, se debe mencionar que el utilizar el lenguaje C paralelo en esta aproximación, permitió realizar con relativa facilidad la partición del algoritmo de covariancia modificada y obtener de este modo, funciones y programas escalables.

7.2 REFERENCIAS

[1] GARCÍA N., F., SOLANO G., J., MARTÍNEZ F., J., RAMOS H., D., Heterogeneous Parallel Architecture for Improving Signal Processing Performance in Doppler Blood Flow Instrumentation, 10th International Parallel Processing Symposium, Honolulu, Hawaii, USA, 1996.

APÉNDICE A

APÉNDICE A

A.1 PROGRAMA 1

```
/******************************************************************************
* PROGRAMA:  MONITOR.C                                                        *
* VARIABLES: P = ORDEN DEL MODELO                                             *
*            M = TAMAÑO DE LA VENTANA DE DATOS                                *
*            X[M] = VECTOR CON LOS DATOS MUESTREADOS (VENTANA)                *
* DESCRIPCIÓN:                                                                *
*        LEE LOS DATOS DEL ARCHIVO SIGNAL.DAT, LOS MANDA                      *
*        A UN SEGUNDO PROCESADOR A TRAVÉS DE UN CANAL Y ESPERA                *
*        LOS DATOS DEL PROCESADOR BUFFER2.                                    *
*        DESPLIEGA EN PANTALLA O GUARDA EN UN ARCHIVO                         *
*        LOS VALORES DE LA PSD CALCULADOS EN EL DSP                           *
******************************************************************************/

/*LIBRERIAS*/

#include <stdio.h>
#include <channel.h>
#include <misc.h>
#include <stdlib.h>
#include <process.h>

int main()
 {
  /*SE DEFINEN LOS CANALES DE COMUNICACIÓN
        CON LOS PROCESOS BUFFER1 Y BUFFER2*/

   Channel * a_buffer1, * de_buffer2, * de_buffer1;

   /*DECLARACIÓN DE VARIABLES*/

   FILE *datos,*archivo;
   char d[512][15],arch[15];
   float  x[512];
   int M=512,real32=4,P=10,i,j;

   de_buffer2 = (Channel *) get_param (3);
   a_buffer1 = (Channel *) get_param (4);
   de_buffer1 =(Channel *) get_param (5);

   printf("\nORDEN DEL MODELO: ");
   scanf("%d",&P);
   printf("VENTANA DE DATOS: ");
   scanf("%d",&M);
   ChanOutInt(a_buffer1,P);
   ChanOutInt(a_buffer1,M);
```

```
if((datos=fopen("signal.dat","rb"))==NULL)
{
    printf("\n No fue posible abrir el archivo de datos");
}
for(i=0;i<M;i++)
{
  fscanf(datos,"%s",&d[i]);
  x[i]=(float)(atof(d[i]));
 }
fclose(datos);

i=real32*M;
ChanOut(a_buffer1,x,i);
i=real32*128;
   for(j=0;j<40;j++)
   {
       ChanIn(de_buffer2,x,i);
       /*SE GUARDAN LOS VALORES EN UN ARCHIVO
              SI NO SE MANDAN A PANTALLA*/
   }

 exit_terminate(0);
}
```

A.2 PROGRAMA 2

```
/*******************************************************************************
* PROGRAMA:  MATRIX1.C                                                         *
* VARIABLES: p = ORDEN DEL MODELO                                              *
*            M = TAMAÑO DE LA VENTANA DE DATOS                                 *
*            x[M] = VECTOR CON LOS DATOS MUESTREADOS (VENTANA)                 *
*            c[p+1] = VECTOR CON LOS DATOS DEL PRIMER RENGLÓN DE               *
*                     LA MATRIZ DE COVARIANCIA                                 *
*            d[p][p] =  MATRIZ DIAGONAL UTILIZADA DE MANERA AUXILIAR           *
*                     EN EL MÉTODO DE CHOLESKI                                 *
*            a[p][p] = MATRIZ DE COVARIANCIA                                   *
*            b[p] = VECTOR DEL LADO DERECHO                                     *
*            l[p][p] = MATRIZ AUXILIAR EN EL MÉTODO DE CHOLESKI                *
* DESCRIPCIÓN:                                                                 *
*      RECIBE EL VECTOR DE DATOS DEL CONVERTIDOR A/D                           *
*      CALCULA EL PRIMER RENGLÓN DE LA MATRIZ DE COVARIANCIA                   *
*      Y POR SIMETRIA ASIGNA LOS DEMÁS VALORES DE LA MATRIZ,                   *
*      RESUELVE EL SISTEMA DE ECUACIONES LINEALES POR EL MÉTODO DE             *
*      CHOLESKI.                                                               *
*      MANDA LOS VALORES DEL PRIMER RENGLÓN DE LA MATRIZ Y LOS                 *
*      PARÁMETROS CALCULADOS AL NODO DE PROCESAMIENTO HETEROGÉNEO*
*******************************************************************************/

/*LIBRERÍAS*/

#include <channel.h>
```

```
#include <process.h>
#include <misc.h>
#include <math.h>

int main()
 {
  /*DECLARACIÓN DE VARIABLES*/
  int n,p=10,M=512,real32=4,r;
  int jj=0;
  float A,B,C,c[11],x[512];
  int k,i,j,N=10;
  float d[10][10],a[10][10],b[10],l[10][10];

  /*SE DEFINEN LOS CANALES DE COMUNICACIÓN
    CON EL NODO Y CON EL CONVERTIDOR A/D */
   Channel * de_buffer1,* a_buffer2;

   de_buffer1 = (Channel *) get_param (1);
   a_buffer2  = (Channel *) get_param (2);

   p=ChanInInt(de_buffer1);
   N=p;
   M=ChanInInt(de_buffer1);
   ChanOutInt(a_buffer2,p);

/*SE REALIZA r VECES*/
for(r=0;r<20;r++)
{
      /*INICIA CÁLCULO DEL PRIMER RENGLÓN DE LA MATRIZ DE COVARIANCIA*/
       k=M*real32;
       ChanIn(de_buffer1,x,k);
       for(k=0;k<=p;k++)
       {
         A=B=0;
         for(n=p;n<=M-1;n++)
         {
           A=A+x[n-jj]*x[n-k];
           B=B+x[n+jj-p]*x[n+k-p];
         }
         C=(A+B)/(M-p);
         c[k]=C/2;
       }

       k=real32*(N+1);
       ChanOut(a_buffer2,c,k);

 /*INICIA CHOLESKI*/
       for(i=0;i<N;i++)
       {
        for(j=0;j<N;j++)
        {
          if(j>i)
                a[i][j]=c[j-i];
          else
```

```
            a[i][j]=c[i-j];
        }
       b[i]=-c[i+1];
      }
            d[0][0]=a[0][0];
            l[0][0]=1;
            for(i=1;i<N;i++)
            {
                    l[i][i]=1;
                    l[i][0]=a[i][0]/d[0][0];
                    l[0][i]=d[0][i]=d[i][0]=0;
                    for(j=1;j<=i-1;j++)
                    {
                                    l[j][i]=0;
                                    l[i][j]=a[i][j]/d[j][j];
                                    d[i][j]=d[j][i]=0;
                                    for(k=0;k<=j-1;k++)
                                            l[i][j]=l[i][j]-l[i][k]*d[k][k]*l[j][k]/d[j][j];
                    }
                    d[i][i]=a[i][i];
                    for(k=0;k<=i-1;k++)
                     d[i][i]=d[i][i]-d[k][k]*l[i][k]*l[i][k];
            }
      for(k=1;k<N;k++)
      {
        for(j=0;j<=k-1;j++)
           b[k]=b[k]-l[k][j]*b[j];
      }

      b[N-1]=b[N-1]/d[N-1][N-1]; /*N-1*/
      for(k=N-2;k>=0;k--) /*k=N-2*/
      {
        b[k]=b[k]/d[k][k];
        for(j=k+1;j<=N-1;j++)
           b[k]=b[k]-l[j][k]*b[j];
      }/*FIN DE CHOLESKI*/

       k=real32*N;
       ChanOut(a_buffer2,b,k);
}/*fin ciclo*/
   exit_terminate(0);
 }

/****************************************************************************************
* EL PROGRAMA MATRIX2.C REALIZA LAS MISMAS OPERACIONES QUE EL PROGRAMA *
* MATRIX1.C, SÓLO QUE SE ASIGNA A DIFERENTE PROCESADOR Y SE DEFINEN SUS     *
* CANALES DE COMUNICACIÓN INDEPENDIENTES                                     *
****************************************************************************************/
```

A.3 PROGRAMA 4

Para poder ejecutar en paralelo los programas monitor.c, matrix1.c, matrix2.c y buffer2.c se requiere del archivo de configuración covmod.cfs, en el cual se define a cada programa como proceso independiente.

```
/*****************************************************************************
*PROGRAMA: COVMOD.CFS                                                        *
*ARCHIVO DE CONFIGURACIÓN DE LA RED DE PROCESADORES                          *
*****************************************************************************/

/*DECLARACIÓN DE PROCESADORES*/
T805 (memory = 1M) pmonitor;
T805 (memory = 1M) pbuffer1;
T805 (memory = 1M) pmatrix1;
T805 (memory = 1M) pmatrix2;
T800 (memory = 1M) pbuffer2;

/*DECLARACIÓN DEL DSP*/
edge dsp;

/*INTERCONEXIÓN DE PROCESADORES*/
connect host to pmonitor.link[0];
connect pmonitor.link[2] to pbuffer1.link[1];
connect pbuffer1.link[2] to pmatrix1.link[1];
connect pbuffer1.link[3] to pmatrix2.link[3];
connect pmatrix1.link[3] to  pbuffer2.link[3];
connect pmatrix2.link[2] to pbuffer2.link[1];
connect pbuffer2.link[0] to pmonitor.link[3];

/*NODO HETEROGÉNEO*/
connect pbuffer2.link[2] to dsp;

/*PROCESO MONITOR (PROGRAMA MONITOR.C*/
process (stacksize = 20K, heapsize = 20K,
      interface (input host_in, output host_out, input de_buffer2,output a_buffer1,input de_buffer1)
      ) monitor;

monitor (priority = HIGH);

/*PROCESO BUFFER1 (PROGRAMA QUE SIMULA AL CONVERTIDOR A/D)*/
process (stacksize = 20K, heapsize = 20K,
      interface (input de_monitor, output a_matrix1, output a_matrix2, output a_monitor)
      ) buffer1;

buffer1 (priority = HIGH);

/*PROCESO MATRIX1 (PROGRAMA MATRIX1.C)*/
process (stacksize = 20K, heapsize = 20K,
      interface (input de_buffer1,output a_buffer2)
      ) matrix1;
```

```
matrix1 (priority = HIGH);

/*PROCESO MATRIX2 (PROGRAMA MATRIX2.C)*/
process (stacksize = 20K, heapsize = 20K,
       interface (input de_buffer1,output a_buffer2)
       ) matrix2;

matrix2 (priority = HIGH);

/*PROCESO BUFFER2 (PROGRAMA BUFFER2.C)*/
process (stacksize = 20K, heapsize = 20K,
      interface (input de_matrix1, input de_matrix2, output a_monitor,input de_dsp, output a_dsp)
      ) buffer2;

buffer2 (priority = HIGH);

/*DEFINICIÓN DE LA CONEXIÓN CON EL HOST */
input HostInput;
output HostOutput;

/*CONEXIÓN CON EL NODO*/
input dspinput;
output dspoutput;

/*CONEXIÓN LÓGICA DE PROCESOS*/
connect HostInput to monitor.host_in;
connect HostOutput to monitor.host_out;
connect monitor.a_buffer1 to buffer1.de_monitor;
connect monitor.de_buffer1 to buffer1.a_monitor;
connect buffer1.a_matrix1 to matrix1.de_buffer1;
connect buffer1.a_matrix2 to matrix2.de_buffer1;
connect matrix1.a_buffer2 to buffer2.de_matrix1;
connect matrix2.a_buffer2 to buffer2.de_matrix2;
connect buffer2.a_monitor to monitor.de_buffer2;
connect buffer2.a_dsp to dspoutput;
connect buffer2.de_dsp to dspinput;

/*MAPEO DE PROGRAMAS A PROCESOS*/
use "monitor.lku" for monitor;
use "matrix1.lku" for matrix1;
use "matrix2.lku" for matrix2;
use "buffer1.lku" for buffer1;
use "buffer2.lku" for buffer2;

/*MAPEO DE PROCESOS A PROCESADORES*/
place monitor on pmonitor;
place matrix1 on pmatrix1;
place matrix2 on pmatrix2;
place buffer1 on pbuffer1;
place buffer2 on pbuffer2;

/*CONEXIÓN LÓGICA CON EL HOST*/
place HostInput on host;
```

```
place HostOutput on host;

/*CONEXIÓN LÓGICA CON EL DSP*/
place dspinput on dsp;
place dspoutput on dsp;

/*MAPEO DE CANALES LÓGICOS A FÍSICOS*/
place monitor.host_in on pmonitor.link[0];
place monitor.host_out on pmonitor.link[0];
place monitor.a_buffer1 on pmonitor.link[2];
place monitor.de_buffer1 on pmonitor.link[2];
place monitor.de_buffer2 on pmonitor.link[3];
place buffer1.de_monitor on pbuffer1.link[1];
place buffer1.a_monitor on pbuffer1.link[1];
place buffer1.a_matrix1 on pbuffer1.link[2];
place matrix1.de_buffer1 on pmatrix1.link[1];
place matrix1.a_buffer2 on pmatrix1.link[3];
place buffer2.de_matrix1 on pbuffer2.link[3];
place buffer1.a_matrix2 on pbuffer1.link[3];
place matrix2.de_buffer1 on pmatrix2.link[3];
place matrix2.a_buffer2 on pmatrix2.link[2];
place buffer2.de_matrix2 on pbuffer2.link[1];
place buffer2.a_monitor on pbuffer2.link[0];
place buffer2.a_dsp on pbuffer2.link[2];
place buffer2.de_dsp on pbuffer2.link[2];
```

APÉNDICE B

APÉNDICE B

B.1 PROGRAMA 1

```
*******************************************************************************************
* PROGRAMA QUE CALCULA LA FFT REAL, OBTENIDO DEL LIBRO
* DIGITAL SIGNAL PROCESSING APPLICATIONS
* EDITADO POR LUIS AGUIRRE TORRES Y MARCO ANTONIO VIGUERAS VILLASEÑOR
* VER MODIFICACIONES EN LAS LINEAS 95 Y 195
*******************************************************************************************
*DESCRIPCIÓN ORIGINAL
*******************************************************************************************
* Name:
*   fft_rl --- radix-2 real FFT to be called as a C function.
*
* Synopsis:
*   int fft_rl(N, M, data)
*   int N         FFT size: N=2**M
*   int M         Number of stages = log2(N)
*   float *data   Array with input and output data
*
* Description:
*   Generic function to do a radix-2 FFT computation on the 320C30.
*   The data array is N-long, with only real data. The output is stored in
*   the same locations with real and imaginary points R and I as follows:
*       R(0), R(1),...., R(N/2), I(N/2-1),..., I(1)
*   The program is based on the FORTRAN program in the paper by Sorensen et
*   al., June 1987 issue of Trans. on ASSP.
*   The computation is done in place, and the original data is destroyed.
*   Bit reversal is implemented at the beginning of the function. If this
*   is not necessary, this part can be commented out.
*   The sine/cosine table for the twiddle factors is expected to be supplied
*   during link time, and it should have the following format:
*
*           .global  _sine
*           .data
*   _sine   .float  value1       = sin(0*2*pi/N)
*           .float  value2       = sin(1*2*pi/N)
*           ......
*           .float  value(N/2)   = cos((N/4)*2*pi/N)
*
*   The values value1 to value(N/4) are the first quarter of the sine
*   period, and value(N/4+1) to value(N/2) are the first quarter of the
*   cosine period.
*
* Stack structure upon the call:
*           +--------------+
*   -FP(4)  |  data        |
*   -FP(3)  |  M           |
```

```
*   -FP(2)   |   N        |
*   -FP(1)   | return addr |
*   -FP(0)   |  old FP    |
*            +--------------+
*
* Registers used: R0, R1, R2, R3, R4, R5, AR0, AR1, AR2, AR4, AR5
*   IR0, IR1, RS, RE, RC
*
* AUTHOR: PANOS E. PAPAMICHALIS
*      TEXAS INSTRUMENTS                OCTOBER 13, 1987
*
*****************************************************************************************
*
FP  .set    AR3

        .GLOBL  _fft_rl       ; ENTRY POINT FOR EXECUTION
        .GLOBL  _sine         ; ADDRESS OF SINE TABLE

        .BSS  FFTSIZ,1
        .BSS  LOGFFT,1
        .BSS  INPUT,1

        .TEXT

SINTAB  .word   _sine

;  INITIALIZE C FUNCTION

_fft_rl:  PUSH     FP          ; SAVE DEDICATED REGISTERS
          LDI      SP,FP
          PUSH     R4
          PUSH     R5
          PUSH     AR4
          PUSH     AR5

          LDI      *-FP(2),R0   ; MOVE ARGUMENTS TO LOCATIONS MATCHING
          STI      R0,@FFTSIZ   ;    THE NAMES IN THE PROGRAM
          LDI      *-FP(3),R0
          STI      R0,@LOGFFT
          LDI      *-FP(4),R0
          STI      R0,@INPUT

;*****************************************************************************
;MODIFICACIÓN DEL PROGRAMA DEL LIBRO DIGITAL SIGNAL PROCESSING APPLICATIONS
;LA SIGUIENTE LINEA DEBE CONTENER LA DIRECCION DE LA VARIABLE data_input
;UTILIZADA EN MAIN.C
;         ldi XXXXh, R0
;ESTA DIRECCIÓN SE ENCUENTRA EN EL ARCHIVO MAIN.MAP QUE SE GENERA AL
;MOMENTO DE COMPILAR.
;*****************************************************************************

          ldi     0A97h, R0
          sti     R0, @INPUT              ;DIRECCIÓN DADA DE ACUERDO A DONDE
                                          ;ALMACENE LOS ELEMENTOS data_input
```

```
        ldi  @FFTSIZ, IR0
        lsh  -1, IR0

; LENGTH-TWO BUTTERFLIES

        LDI   @INPUT,AR0     ; AR0 POINTS TO X(I)

        LDI   IR0,RC         ; REPEAT N/2 TIMES
        SUBI  1,RC           ; RC SHOULD BE ONE LESS THAN DESIRED #
        RPTB  BLK1
        ADDF  *+AR0,*AR0++,R0        ; R0=X(I)+X(I+1)
        SUBF  *AR0,*-AR0,R1          ; R1=X(I)-X(I+1)
BLK1    STF   R0,*-AR0               ; X(I)=X(I)+X(I+1)
||      STF   R1,*AR0++              ; X(I+1)=X(I)-X(I+1)

; FIRST PASS OF THE DO-20 LOOP (STAGE K=2 IN DO-10 LOOP)

        LDI   @INPUT,AR0     ; AR0 POINTS TO X(I)

        LDI   2,IR0          ; IR0=2=N2
        LDI   @FFTSIZ,RC
        LSH   -2,RC          ; REPEAT N/4 TIMES
        SUBI  1,RC           ; RC SHOULD BE ONE LESS THAN DESIRED #
        RPTB  BLK2
        ADDF  *+AR0(IR0),*AR0++(IR0),R0   ; R0=X(I)+X(I+2)
        SUBF  *AR0,*-AR0(IR0),R1          ; R1=X(I)-X(I+2)
        NEGF  *+AR0,R0                    ; R0=-X(I+3)
||      STF   R0,*-AR0(IR0)               ; X(I)=X(I)+X(I+2)
BLK2    STF   R1,*AR0++(IR0)              ; X(I+2)=X(I)-X(I+2)
||      STF   R0,*+AR0                    ; X(I+3)=-X(I+3)

; MAIN LOOP (FFT STAGES)
        LDI   @FFTSIZ,IR0
        LSH   -2,IR0         ; IR0=INDEX FOR E
        LDI   3,R5           ; R5 HOLDS THE CURRENT STAGE NUMBER
        LDI   1,R4           ; R4=N4
        LDI   2,R3           ; R3=N2
LOOP    LSH   -1,IR0         ; E=E/2
        LSH   1,R4           ; N4=2*N4
        LSH   1,R3           ; N2=2*N2

; INNER LOOP (DO-20 LOOP IN THE PROGRAM)

        LDI   @INPUT,AR5     ; AR5 POINTS TO X(I)

INLOP   LDI   IR0,AR0
        ADDI  @SINTAB,AR0    ; AR0 POINTS TO SIN/COS TABLE
```

```
        LDI    R4,IR1       ; IR1=N4
        LDI    AR5,AR1
        ADDI   1,AR1        ; AR1 POINTS TO X(I1)=X(I+J)
        LDI    AR1,AR3
        ADDI   R3,AR3       ; AR3 POINTS TO X(I3)=X(I+J+N2)
        LDI    AR3,AR2
        SUBI   2,AR2        ; AR2 POINTS TO X(I2)=X(I-J+N2)
        ADDI   R3,AR2,AR4   ; AR4 POINTS TO X(I4)=X(I-J+N1)
        LDF    *AR5++(IR1),R0    ; R0=X(I)
        ADDF   *+AR5(IR1),R0,R1  ; R1=X(I)+X(I+N2)
        SUBF   R0,*++AR5(IR1),R0 ; R0=-X(I)+X(I+N2)
||      STF    R1,*-AR5(IR1)     ; X(I)=X(I)+X(I+N2)
        NEGF   R0                ; R0=X(I)-X(I+N2)
        NEGF   *++AR5(IR1),R1    ; R1=-X(I+N4+N2)
||      STF    R0,*AR5           ; X(I+N2)=X(I)-X(I+N2)
        STF    R1,*AR5           ; X(I+N4+N2)=-X(I+N4+N2)


; INNERMOST LOOP
        LDI    @FFTSIZ,IR1
        LSH    -2,IR1         ; IR1=SEPARATION BETWEEN SIN/COS TBLS
        LDI    R4,RC
        SUBI   2,RC           ; REPEAT N4-1 TIMES
        RPTB   BLK3
        MPYF   *AR3,*+AR0(IR1),R0    ; R0=X(I3)*COS
        MPYF   *AR4,*AR0,R1          ; R1=X(I4)*SIN
        MPYF   *AR4,*+AR0(IR1),R1    ; R1=X(I4)*COS
||      ADDF   R0,R1,R2              ; R2=X(I3)*COS+X(I4)*SIN
        MPYF   *AR3,*AR0++(IR0),R0   ; R0=X(I3)*SIN
        SUBF   R0,R1,R0              ; R0=-X(I3)*SIN+X(I4)*COS !!!
        SUBF   *AR2,R0,R1            ; R1=-X(I2)+R0 !!!
        ADDF   *AR2,R0,R1            ; R1=X(I2)+R0 !!!
||      STF    R1,*AR3++             ; X(I3)=-X(I2)+R0 !!!
        ADDF   *AR1,R2,R1            ; R1=X(I1)+R2
||      STF    R1,*AR4--             ; X(I4)=X(I2)+R0 !!!
        SUBF   R2,*AR1,R1            ; R1=X(I1)-R2
||      STF    R1,*AR1++             ; X(I1)=X(I1)+R2
BLK3    STF    R1,*AR2--             ; X(I2)=X(I1)-R2


        SUBI   @INPUT,AR5


        ADDI   R4,AR5         ; AR5=I+N1
        CMPI   @FFTSIZ,AR5
        BLED   INLOP          ; LOOP BACK TO THE INNER LOOP


        ADDI   @INPUT,AR5


        NOP
        NOP
```

```
        ADDI  1,R5
        CMPI  @LOGFFT,R5
        BLE   LOOP

; RESTORE THE REGISTER VALUES AND RETURN

        POP   AR5
        POP   AR4
        POP   R5
        POP   R4
        POP   FP
        RETS
```

B.2 PROGRAMA 2

```
/***************************************************************************
PROGRAMA QUE RECIBE DATOS DEL TRANSPUTER, HACE LA FFT + PSD
 POR DECIMACIÓN EN FRECUENCIA Y MANDA LA MITAD DE REGRESO AL
 TRANSPUTER

 PARA COMPILAR ES NECESARIO COMPILAR EL GRUPO DE PROGRAMAS

 APPENDIX.ASM
 SENO.ASM
 INICIA.ASM
 RECIBE.ASM
 MANDA.ASM

 DE LA SIGUIENTE MANERA:

asm30 inicia
asm30 manda
asm30 recibe
asm30 seno
asm30 appendix -l
CL30 -kg -O2 -al main.c -z main.cmd

***************************************************************************/

#include "stdlib.h"

/*********************FUNCIÓN QUE REALIZA LA FFT**********************/
extern int  fft_rl(int N,int  M,float *data);


extern int begin(void);  /* FUNCIÓN QUE INICIALIZA EL PUERTO SERIE        */
extern int recibe(void); /* FUNCIÓN QUE RECIBE P NÚMEROS DE PUNTO FLOTANTE */
extern int manda(void);  /* FUNCIÓN QUE MANDA P NÚMEROS DE PUNTO FLOTANTE  */

float data_input[512];  /* VECTOR CON LOS DATOS */
int P;                  /* NÚMERO DE DATOS QUE SE RECIBEN Y MANDAN*/
```

```
float cxx[11];
float var;
int ven;
main()
{
int  N;
int  M;
int k,j,i;
float *direc;
float templ,temp2;
N=256; /*TAMAÑO DE LA VENTANA DE DATOS*/
M=8;  /*LOG BASE 2 DE N*/
data_input[0]=1.0;
for(i=1;i<(2*N);i++) /* COLOCA SÓLO CEROS EN EL VECTOR data_inoput */
 data_input[i]=0.0;

begin();/*LLAMADA A LA FUNCIÓN DE INICIALIZACIÓN

  for(ven=0;ven<40;ven++)
  {
     P=11;       /*EL VALOR DE P SE ASIGNA SEGÚN EL NÚMERO DE PARÁMETROS +1*/
     recibe();          /*RECIBE P DATOS EN data_input */
     for(i=1;i<=P;i++)
          cxx[i-1]=data_input[i];
     var=cxx[0];
     data_input[P]=0.0;
     P=10;       /*EL VALOR DE P SE ASIGNA SEGÚN EL NÚMERO DE PARÁMETROS*/
     recibe();          /* P DATOS EN data_input[1] EN ADELANTE    */
     for(i=1;i<=P;i++)   /* OBTENCIÓN DE LA VARIANZA DE RUIDO BLANCO */
          var=var+data_input[i]*cxx[i];

     /****BIT REVERSE ****/

     j=0;
     for(i=1;i<N-1;i++)
     {
          k=128;      /* N/2 */
          while(k<=j)
          {
               j=j-k;
               k=k/2;
          }
          j=j+k;
          if(i<j)
          {
               templ=data_input[j];
               data_input[j]=data_input[i];
               data_input[i]=templ;
          }
     } /***FIN DEL BIT REVERSE****/

     *direc=0;
     i = fft_rl(N,M,direc);
     j=N-1;
```

```
        temp1=data_input[0];
        data_input[0]=temp1*temp1;
        data_input[0]=var/data_input[0];
        for(i=1;i<=N/2;i++)
        {
            temp1=data_input[i];
            temp2=data_input[j];
            data_input[i]=temp1*temp1+temp2*temp2;
            data_input[i]=var/data_input[i];
            j--;
        }.

        P=N/2;
        manda();        /* REGRESA P DATOS AL TRANSPUTER */
        data_input[0]=1.0;
        for(i=1;i<(2*N);i++) /* COLOCA CEROS A LOS DATOS */
            data_input[i]=0.0;


    }

return 1;
}
/*FIN DEL PROGRAMA MAIN.C*/
```

B.3 PROGRAMA 3

```
;*****************************************************************************
;EL PROGRAMA INICIA.ASM INICIALIZA EL PUERTO SERIE Y EL TIMER PARA LA
;COMUNICACIÓN CON EL TRANSPUTER.
;EL PROGRAMA DEBE SER EJECUTADO POR MEDIO DE UN PROGRAMA EN C EN DONDE SE
;DEBE DECLARAR COMO
;extern int begin(void)
;Y SE UTILIZA COMO FUNCION
;*****************************************************************************

        .title  "BEGIN"
FP       .set AR3
        .global _inicia
        .global _regresa
        .global _main
        .global _data_input
        .global _R

;CONVERSION TABLE FROM IEEE TO TMS320 AND TMS320 TO IEEE
        .data
ctab     .word  0FF800000h
        .word  0FF000000h
        .word  07F000000h
        .word  080000000h
```

```
          .word  081000000h
taba        .word  ctab
          .sect  "vectors"
reset:      .word  _inicia
          .text

; BASE ADDRESS OF SERIAL PORTS 1:
serial_1     .word  808050h
timer_1      .word  808030h

; OFFSET FROM BASE ADDRESS OF SERIAL PORTS REGISTERS:
gcr         .set   00          ;Global Control Register
tx_cr        .set   02          ;FSX/DX/CLKX Port Control Register
rx_cr        .set   03          ;FSR/DR/CLKR Port Control Register
timer_cr      .set   04          ;R/X Timer Control Register
period       .set   06          ;R/X Timer Period Register
dxr         .set   08          ;Data Transmit Register
drr         .set   0Ch         ;Data Receive Register
timer_gc      .set   00          ;Timer Global Control

; PROGRAMMING VALUES TO BE SET ON SERIAL PORT REGISTERS:
                              ;XXXX BBBB BBBB BBBB BBBB BBBB BBBR RBRR
gcr_word      .word  0E800044h   ;0000 1110 1000 0000 0000 0000 0100 0100
serial_reset  .word  00800044h   ;0000 0000 1000 0000 0000 0000 0100 0100

                              ;XXXX XXXX XXXX XXXX XXXX RBBB RBBB RBBB
tx_cr_word    .word  0111h       ;0000 0000 0000 0000 0000 0001 0001 0001
rx_cr_word    .word  0111h       ;0000 0000 0000 0000 0000 0001 0001 0001

                              ;XXXX XXXX XXXX XXXX XXXX RXBB RBBX RBBB
timer_ctl_word .word  000Fh      ;0000 0000 0000 0000 0000 0000 0000 1111
timer_prd_word .word  0001h      ;0000 0000 0000 0000 0000 0000 0000 0010
timer_gc_word  .word  0002h      ;0000 0000 0000 0000 0000 0000 0000 0010
timer_gc_word2 .word  0006h      ;0000 0000 0000 0000 0000 0000 0000 0110


;* MAIN PROGRAM: *

_inicia:
     nop
     nop
     push  FP
     ldi   SP, FP
     ldi   80h, DP          ;point to internal registers
     ldi   0, R0
     sti   R0, @8064h        ;set zero wait states on primary bus
     or    0800h, ST         ;turn on cache
     ldi   0, DP            ;point to where this stuff is

; SET SERIAL 1 MAP ADDRESS:
     ldi   @serial_1, AR1

; SET TIMER 1:
     ldi   @timer_1, AR2
```

```
      ldi    @timer_gc_word2, R0
      sti    R0, *+AR2(timer_gc) ;set TCLK1

; SET TX PORT CONTROL REGISTER
      ldi    @tx_cr_word, R0
      sti    R0, *+AR1(tx_cr)    ;set port 1

; SET RX PORT CONTROL REGISTER:
      ldi    @rx_cr_word, R0
      sti    R0, *+AR1(rx_cr)    ;set port 1

; SET UP TIMER:
      ldi    @timer_ctl_word, R0
      sti    R0, *+AR1(timer_cr) ;set port 1

; SET UP TIMER PERIOD:
      ldi    @timer_prd_word, R0
      sti    R0, *+AR1(period)   ;set port 1

; RESET SERIAL PORT:
      ldi    @serial_reset, R0
      sti    R0, *+AR1(gcr)      ;reset port 1
      or     00h, IOF            ;XF1 configured as input pin
      nop
      nop
hang:  nop                       ;kill time

; REMOVE RESET FROM SERIAL PORTS:
      ldi    @gcr_word, R0
      sti    R0, *+AR1(gcr)      ;port 1
      or     0C0h, IE            ;set IE bits EXINT1 and ERINT1
      or     40h, IF             ;force TX ready flag
      ldi    00h, R1             ;initial value of R1 for
                                 ;transmit
      ldi    @_data_input, AR0   ;load the first word reserved
                                 ;to AR0

rx_1_init:     tstb   80h, IF    ;test bit 7 IF
      bz     rx_1_init           ;wait for RX buffer full
      ldi    0FF7Fh, R2
      and    R2, IF              ;clear interrupt flag
      ldi    *+AR1(drr), R1      ;load buffer RX into R1
      and    00FFh,R1
      ldi    0067h, R7
      cmpi   R1, R7              ;compares RX with 0067h
      bnz    rx_1_init           ;wait if RX not equal to 0067h
      ldi    @timer_gc_word, R0
      sti    R0, *+AR2(timer_gc) ;turn off TCLK1

main_loop:     nop
      ldi    0, IR0
      ldi    @timer_gc_word2, R0
      sti    R0, *+AR2(timer_gc) ;set TCLK1
```

```
inicia_rx:    ldi   00h, R3

rx_1_wait:    tstb   80h, IF    ;test bit 7 IF
      bz    rx_1_wait          ;wait for RX buffer full
      ldi   0FF7Fh, R2
      and   R2, IF             ;clear interrupt flag
      ldi   *+AR1(drr), R1     ;load buffer RX into R1
      and   00FFh,R1
      ldi   R1, R3             ;load R1 into R3

rx_2_wait:    tstb   80h, IF    ;test bit 7 IF
      bz    rx_2_wait          ;wait for RX buffer full
      ldi   0FF7Fh, R2
      and   R2, IF             ;clear interrupt flag
      ldi   *+AR1(drr), R1     ;load buffer RX into R1
      and   00FFh,R1
      ash   8, R1              ;left-shift 8 bits of R1
      or    R1, R3             ;load R1 into R3

rx_3_wait:    tstb   80h, IF    ;test bit 7 IF
      bz    rx_3_wait          ;wait for RX buffer full
      ldi   0FF7Fh, R2
      and   R2, IF             ;clear interrupt flag
      ldi   *+AR1(drr), R1     ;load buffer RX into R1
      and   00FFh,R1
      ash   16, R1             ;left-shift 16 bits of R1
      or    R1, R3             ;load R1 into R3

rx_4_wait:    tstb   80h, IF    ;test bit 7 IF
      bz    rx_4_wait          ;wait for RX buffer full
      ldi   0FF7Fh, R2
      and   R2, IF             ;clear interrupt flag
      ldi   *+AR1(drr), R1     ;load buffer RX into R1
      and   00FFh,R1
      ash   24, R1             ;left-shift 24 bits of R1
      or    R1, R3             ;load R1 into R3
      sti   R3, *+AR0(IR0)     ;load R3 into AR0
      addi  1, IR0             ;incr. index IR0
      ldi   100h, R6           ;data = 256
      cmpi  IR0, R6
      bnz   inicia_rx
      ldi   @timer_gc_word, R0
      sti   R0, *+AR2(timer_gc) ;turn off TCLK1 and finish the
                               ;receive


      pop FP
      .end
```

B.4 PROGRAMA 4

```
;*******************************************************************************************
;EL PROGRAMA MANDA.ASM ENVÍA DATOS ALOJADOS EN LA VARIABLE data_input DEL DSP
;AL TRANSPUTER. LA VARIABLE data_input SE DECLARA COMO VARIABLE GLOBAL
;DENTRO DE UN PROGRAMA EN C.
;DEBE SER EJECUTADO DENTRO DE UN PROGRAMA EN C COMO:
;extern int manda(void)
;EL NÚMERO DE DATOS ENVIADOS VIENE DETERMINADO POR LA VARIABLE P TAMBIÉN
;DECLARADA COMO GLOBAL EN EL PROGRAMA EN C
;LOS VALORES QUE MANDA SE ALOJARAN EN LA DIRECCIÓN DE MEMORIA QUE SE
;DETERMINE EN LAS LINEAS 85, 90 Y 113 EN DONDE SE TIENE:
;      LDI XXXXh, AR0
;PUEDE SER LA DIRECCIÓN DE data_input[0] O CUALQUIER OTRA DIRECCIÓN DENTRO
;DE ESTE VECTOR TENIENDO MUCHO CUIDADO EN EL MANEJO DE LA MEMORIA,
;SI SE TIENE ALGUNA DUDA DE QUE DIRECCION UTILIZAR ES POSIBLE CONOCER LA
;DIRECCIÓN DE MEMORIA DE data_input[0] EN EL ARCHIVO DE C CON TERMINACIÓN
;ARCHIVO.MAP DESPUES DE COMPILAR EL PAQUETE DE PROGRAMAS NECESARIOS
;EN ESTE CASO SE ALOJAN A PARTIR DE data_input[0]
;*******************************************************************************************


                .title  "MANDA"
FP .set AR3
                .global _manda
                .global _main
                .global _data_input
                .global _P

M          .set   128        ;M = WINDOW SIZE / 2

;CONVERSION TABLE FROM IEEE TO TMS320 AND TMS320 TO IEEE
                .data
ctab       .word  0FF800000h
                .word  0FF000000h
                .word  07F000000h
                .word  080000000h
                .word  081000000h
taba       .word  ctab

                .sect  "vectors"

reset:     .word  _manda

                .text

; BASE ADDRESS OF SERIAL PORTS I:
serial_I     .word  808050h
timer_I      .word  808030h

; OFFSET FROM BASE ADDRESS OF SERIAL PORTS REGISTERS:
```

```
gcr          .set   00         ;Global Control Register
tx_cr        .set   02         ;FSX/DX/CLKX Port Control Register
rx_cr        .set   03         ;FSR/DR/CLKR Port Control Register
timer_cr     .set   04         ;R/X Timer Control Register
period       .set   06         ;R/X Timer Period Register
dxr          .set   08         ;Data Transmit Register
drr          .set   0Ch        ;Data Receive Register
timer_gc     .set   00         ;Timer Global Control

; PROGRAMMING VALUES TO BE SET ON SERIAL PORT REGISTERS:
                                  ;XXXX BBBB BBBB BBBB BBBB BBBB BBBR RBRR
gcr_word      .word  0E800044h   ;0000 1110 1000 0000 0000 0000 0100 0100
serial_reset  .word  00800044h   ;0000 0000 1000 0000 0000 0000 0100 0100
                                  ;XXXX XXXX XXXX XXXX XXXX RBBB RBBB RBBB
tx_cr_word    .word  0111h       ;0000 0000 0000 0000 0000 0001 0001 0001
rx_cr_word    .word  0111h       ;0000 0000 0000 0000 0000 0001 0001 0001
                                  ;XXXX XXXX XXXX XXXX XXXX RXBB RBBX RBBB
timer_ctl_word .word  000Fh      ;0000 0000 0000 0000 0000 0000 0000 1111
timer_prd_word .word  0001h      ;0000 0000 0000 0000 0000 0000 0000 0010
timer_gc_word  .word  0002h      ;0000 0000 0000 0000 0000 0000 0000 0010
timer_gc_word2 .word  0006h      ;0000 0000 0000 0000 0000 0000 0000 0110

regresa      .word  M          ;REGRESA IS THE ARRAY THAT RETURNS TO THE TRANSPUTER


;*MAIN PROGRAM:*

_manda:
        push   FP
        ldi    SP, FP
        ldi    0A97h, AR0        ; DATAINPUT MEMORY ADRESS
        ldi    @serial_1, AR1    ;STORES THE SERIAL PORT MEMORY ADDRESS
        ldi    @timer_1, AR2     ;STORES THE TIMER MEMORY ADRESS

;TMS320C30 TO IEEE FLOATING-POINT FORMAT COVERSION
        ldi    0A97h, AR0        ; DATAINPUT MEMORY ADDRESS
        ldi    @_P, RC

;REGISTER BASED PARAMETER ENTRY
        subi   1, RC             ;RC <= N-1
        ldi    @taba, AR3        ;AR3 -> constant table

;'C30 -> IEEE CONVERSION LOOP
        rptb   loop5             ;repeat loop N times
        absf   *AR0, R0          ;test abs(number)
        ldfz   *+AR3(4), R0      ;if == zero, load fake 0.0
        lsh    1, R0             ;shift off sign bit
        pushf  R0                ;save as a flt. pt.
        ldf    *AR0, R1          ;test original number
        bged   loop5             ;if >= 0, store number(delayed)
        pop    R0                ;unsave as an interger number
        addi   *+AR3(2), R0      ;add exponent bias (127)
        lsh    -1, R0            ;adjust for sign bit
        or     *+AR3(3), R0      ;negate ieee number
```

```
loop5: sti    R0, *AR0++          ;store ieee number, incr. AR0

;TRANSMITION OF DATA
        ldi   0A97h, AR0          ;direccion de la variable fitness
        ldi   0, IR0

inicia_tx:
        nop

tx_1_loop:
        ldi   *+AR0(IR0), R1
        sti   R1, *+AR1(dxr)      ;send R1 value through TX1

tx_1_wait:
        tstb  40h, IF
        bz    tx_1_wait           ;wait for TX buffer empty
        ldi   0FFBFh, R2
        and   R2, IF              ;clear interrupt flag

iack_11_wait:  tstb   80h, IOF
        bz    iack_11_wait        ;wait until IACK = 1
        ldi   @timer_gc_word2, R0
        sti   R0, *+AR2(timer_gc) ;set TCLK1

iack_12_wait:  tstb   80h, IOF
        bnz   iack_12_wait        ;wait until IACK = 0
        ldi   @timer_gc_word, R0
        sti   R0, *+AR2(timer_gc) ;turn off TCLK1
        ldi   *+AR0(IR0), R1
        ash   -8, R1              ;right-shift 8 bits of R1
        sti   R1, *+AR1(dxr)      ;send R1 value through TX1

tx_2_wait:
        tstb  40h, IF
        bz    tx_2_wait           ;wait for TX buffer empty
        ldi   0FFBFh, R2
        and   R2, IF              ;clear interrupt flag

iack_21_wait:
        tstb  80h, IOF
        bz    iack_21_wait        ;wait until IACK = 1
        ldi   @timer_gc_word2, R0
        sti   R0, *+AR2(timer_gc) ;set TCLK1

iack_22_wait:
        tstb  80h, IOF
        bnz   iack_22_wait        ;wait until IACK = 0
        ldi   @timer_gc_word, R0
        sti   R0, *+AR2(timer_gc) ;turn off TCLK1
        ldi   *+AR0(IR0), R1
        ash   -16, R1             ;right-shift 16 bits of R1
        sti   R1, *+AR1(dxr)      ;send R1 value through TX1
```

```
tx_3_wait:
        tstb   40h, IF
        bz     tx_3_wait          ;wait for TX buffer empty
        ldi    0FFBFh, R2
        and    R2, IF             ;clear interrupt flag

iack_31_wait:
        tstb   80h, IOF           ;until IACK = 1
        bz     iack_31_wait
        ldi    @timer_gc_word2, R0
        sti    R0, *+AR2(timer_gc)  ;set TCLK1

iack_32_wait:  tstb   80h, IOF
        bnz    iack_32_wait       ;until IACK = 0
        ldi    @timer_gc_word, R0
        sti    R0, *+AR2(timer_gc)  ;turn off TCLK1
        ldi    *+AR0(IR0), R1
        ash    -24, R1            ;right-shift 24 bits of R1
        sti    R1, *+AR1(dxr)     ;send R1 value through TX1


tx_4_wait:
        tstb   40h, IF
        bz     tx_4_wait          ;wait for TX buffer empty
        ldi    0FFBFh, R2
        and    R2, IF             ;clear interrupt flag

iack_41_wait:
        tstb   80h, IOF
        bz     iack_41_wait       ;wait until IACK = 1
        ldi    @timer_gc_word2, R0
        sti    R0, *+AR2(timer_gc)  ;set TCLK1

iack_42_wait:
        tstb   80h, IOF
        bnz    iack_42_wait       ;wait until IACK = 0
        ldi    @timer_gc_word, R0
        sti    R0, *+AR2(timer_gc)  ;turn off TCLK1
        addi   1, IR0             ;incr. index IR0
        ldi    @_P, R6

        cmpi   IR0, R6
        bnz    inicia_tx

        pop    FP
        RETS
```

B.5 PROGRAMA 5

```
;*************************************************************************************
;EL PROGRAMA RECIBE.ASM RECIBE DATOS DEL TRANSPUTER EN EL VECTOR data_input
;QUE SE DECLARA COMO VARIABLE GLOBAL DENTRO DE UN PROGRAMA EN C.
;DEBE SER EJECUTADO DENTRO DE UN PROGRAMA EN C COMO:
;extern int recibe(void)
;EL NÚMERO DE DATOS VIENE DETERMINADO POR LA VARIABLE P TAMBIÉN DECLARADA
;COMO GLOBAL EN EL PROGRAMA EN C
;LOS VALORES QUE RECIBE SE ALOJARÁN EN LA DIRECCIÓN DE MEMORIA QUE SE
;DETERMINE EN LAS LINEAS 78, 138 Y 158 EN DONDE SE TIENE:
;       LDI XXXXH, AR0
;PUEDE SER LA DIRECCIÓN DE data_input[0] O CUALQUIER OTRA DIRECCIÓN DENTRO
;DE ESTE VECTOR TENIENDO MUCHO CUIDADO EN EL MANEJO DE LA MEMORIA,
;SI SE TIENE ALGUNA DUDA DE QUE DIRECCIÓN UTILIZAR ES POSIBLE CONOCER LA
;DIRECCION DE MEMORIA DE data_input[0] EN EL ARCHIVO DE C CON TERMINACIÓN
;ARCHIVO.MAP DESPUES DE COMPILAR EL PAQUETE DE PROGRAMAS NECESARIOS
;EN ESTE CASO SE ALOJAN A PARTIR DE data_input[1] POR FINES PROPIOS DEL
;ALGORITMO
;*************************************************************************

            .title  "RECIBE"
FP .set AR3
            .global _recibe
            .global _main
            .global _data_input
            .global _P

;TABLE WITH CONSTANTS FOR CONVERSION IEEE to TMS320C30
;AND TMS320C30 TO IEEE
            .data
ctab        .word  0FF800000h
            .word  0FF000000h
            .word  07F000000h
            .word  080000000h
            .word  081000000h
taba        .word  ctab
            .sect  "vectors"
reset:      .word  _recibe
            .text
; Base address of serial ports 1:
serial_1    .word  808050h
timer_1     .word  808030h

; OFFSET FROM BASE ADDRESS OF SERIAL PORTS REGISTERS:
gcr         .set  00        ;Global Control Register
tx_cr       .set  02        ;FSX/DX/CLKX Port Control Register
rx_cr       .set  03        ;FSR/DR/CLKR Port Control Register
timer_cr    .set  04        ;R/X Timer Control Register
period      .set  06        ;R/X Timer Period Register
```

```
dxr          .set   08           ;Data Transmit Register
drr          .set   0Ch          ;Data Receive Register
timer_gc     .set   00           ;Timer Global Control

; PROGRAMMING VALUES TO BE SET ON SERIAL PORT REGISTERS:
                                  ;XXXX BBBB BBBB BBBB BBBB BBBB BBBR RBRR
gcr_word      .word  0E800044h   ;0000 1110 1000 0000 0000 0000 0100 0100
serial_reset  .word  00800044h   ;0000 0000 1000 0000 0000 0000 0100 0100

                                  ;XXXX XXXX XXXX XXXX XXXX RBBB RBBB RBBB
tx_cr_word    .word  0111h       ;0000 0000 0000 0000 0000 0001 0001 0001
rx_cr_word    .word  0111h       ;0000 0000 0000 0000 0000 0001 0001 0001

                                  ;XXXX XXXX XXXX XXXX XXXX RXBB RBBX RBBB
timer_ctl_word .word  000Fh      ;0000 0000 0000 0000 0000 0000 0000 1111
timer_prd_word .word  0001h      ;0000 0000 0000 0000 0000 0000 0000 0010
timer_gc_word  .word  0002h      ;0000 0000 0000 0000 0000 0000 0000 0010
timer_gc_word2 .word  0006h      ;0000 0000 0000 0000 0000 0000 0000 0110


;* MAIN PROGRAM: *

_recibe:
        push  FP
        ldi   SP, FP
        ldi   0A98h, AR0         ;THIS IS THE MEMORY ADDRESS OF THE ARRAY a
        ldi   @serial_1, AR1     ;STORES SERIAL PORT MEMORY ADDRESS
        ldi   @timer_1, AR2      ;STORES TIEMER MEMORY ADDRESS

main_loop:
        nop
        ldi   0, IR0
        ldi   @timer_gc_word2, R0
        sti   R0, *+AR2(timer_gc)  ;turn off TCLK1

inicia_rx:
        ldi   00h, R3

rx_1_wait:
        tstb  80h, IF            ;test bit 7 IF
        bz    rx_1_wait          ;wait for RX buffer full
        ldi   0FF7Fh, R2
        and   R2, IF             ;clear interrupt flag
        ldi   *+AR1(drr), R1     ;load buffer RX into R1
        and   00FFh,R1
        ldi   R1, R3             ;load R1 into R3

rx_2_wait:
        tstb  80h, IF            ;test bit 7 IF
        bz    rx_2_wait          ;wait for RX buffer full
        ldi   0FF7Fh, R2
        and   R2, IF             ;clear interrupt flag
        ldi   *+AR1(drr), R1     ;load buffer RX into R1
        and   00FFh,R1
```

```
        ash   8, R1          ;left-shift 8 bits of R1
        or    R1, R3         ;load R1 into R3

rx_3_wait:
        tstb  80h, IF        ;test bit 7 IF
        bz    rx_3_wait       ;wait for RX buffer full
        ldi   0FF7Fh, R2
        and    R2, IF        ;clear interrupt flag
        ldi   *+AR1(drr), R1      ;load buffer RX into R1
        and    00FFh,R1
        ash    16, R1        ;left-shift 16 bits of R1
        or    R1, R3         ;load R1 into R3

rx_4_wait:
        tstb  80h, IF        ;test bit 7 IF
        bz    rx_4_wait       ;wait for RX buffer full
        ldi   0FF7Fh, R2
        and    R2, IF        ;clear interrupt flag
        ldi   *+AR1(drr), R1      ;load buffer RX into R1
        and    00FFh,R1
        ash    24, R1        ;left-shift 24 bits of R1
        or    R1, R3         ;load R1 into R3
        sti   R3, *+AR0(IR0)      ;load R3 into AR0
        addi   1, IR0        ;incr. index IR0
        ldi   @_P, R6          ;Number of parameters
        cmpi  IR0, R6
        bnz   inicia_rx
        ldi   @timer_gc_word, R0
        sti   R0, *+AR2(timer_gc)  ;turn off TCLK1 and finish the

;IEEE TO TMS320C30 FLOATING-POINT FORMAT COVERSION
        ldi   0A98h, AR0         ;memory address for the array a
        ldi   @_P, RC

;REGISTER BASED PARAMETER ENTRY
        subi   1, RC         ;RC <= N-1
        ldi   @taba, AR3        ;AR3 -> constant table

;IEEE -> 'C30 CONVERSION LOOP
        rptb  loop4          ;repeat loop N times
        and    *AR0, *AR3, R0      ;replace fraction with 0
        addi   *AR0, R0          ;shift sign and exponent inserting 0
        ldiz   *+AR3(1), R0        ;if all zero, load 'C30 0.0
        ldi   *AR0, R1          ;test original number
        bged   loop4            ;if >= 0, store number(delayed)
        subi   *+AR3(2), R0        ;remove exponent bias (127)
        push   R0            ;save as an integer
        popf   R0            ;unsave as a flt. pt. number
        negf   R0            ;negate 'C30 number

loop4:  stf   R0, *AR0++         ;store 'C30 number, incr. AR0
        ldi   0A98h, AR0         ;direccion inicial del vector a

        pop   FP
```

```
RETS
```

B.6 PROGRAMA 6

```
;***********************************************************************
;TABLA DE SENOS Y COSENOS PARA EL CÁLCULO DE LOS FACTORES TWIDLE
;SE GENERA CON UN PROGRAMA (EN CUALQUIER LENGUAJE) QUE CALCULE LOS VALORES
;CON LA SIGUIENTE FUNCIÓN:
;  VALOR(0)       = SIN (0*PI*2/N)
;     .
;     .
;  VALOR(N/4 - 1) = SIN ([N/4 - 1]*PI*2/N)
;  VALOR(N/4)     = COS (0*PI*2/N)
;     .
;     .
;  VALOR(N/2 - 1) = COS ([N/4 - 1]*PI*2/N)
;SI LA VENTANA ES DE TAMAÑO 'N' SE DEBEN GENERAR 'N/2' VALORES
;
;REFERENCIA:
;DIGITAL SIGNAL PROCESSING APPLICATIONS
;PANOS E. PAPAMICHALIS
;APPENDIX C2. FFT_RL-RADIX-2 REAL FFT TO BE CALL AS A C FUNCTION
;***********************************************************************

.global _main
 .global _fft_rl
 .global _sine
 .data
_sine   .float 0.000000
   .float 0.024541
   .float 0.049068
   .float 0.073565
   .float 0.098017
   .float 0.122411
   .float 0.146731
   .float 0.170962
   .float 0.195091
   .float 0.219101
   .float 0.242980
   .float 0.266713
   .float 0.290285
   .float 0.313682
   .float 0.336890
   .float 0.359895
   .float 0.382684
   .float 0.405242
   .float 0.427556
   .float 0.449612
   .float 0.471397
   .float 0.492899
   .float 0.514103
   .float 0.534998
   .float 0.555571
   .float 0.575809
   .float 0.595700
   .float 0.615232
   .float 0.634394
   .float 0.653173
   .float 0.671560
   .float 0.689541
   .float 0.707107
   .float 0.724248
   .float 0.740952
   .float 0.757209
   .float 0.773011
   .float 0.788347
   .float 0.803208
   .float 0.817585
   .float 0.831470
   .float 0.844854
   .float 0.857729
   .float 0.870088
   .float 0.881922
   .float 0.893225
   .float 0.903990
   .float 0.914210
   .float 0.923880
   .float 0.932993
```

```
.float 0.941545
.float 0.949529
.float 0.956941
.float 0.963776
float 0.970032
float 0.975702
.float 0.980786
.float 0.985278
.float 0.989177
.float 0.992480
.float 0.995185
.float 0.997291
.float 0.998796
.float 0.999699
.float 1.000000
.float 0.999699
.float 0.998795
.float 0.997290
.float 0.995185
.float 0.992480
.float 0.989176
.float 0.985278
.float 0.980785
.float 0.975702
.float 0.970031
.float 0.963776
.float 0.956940
.float 0.949528
.float 0.941544
.float 0.932993
.float 0.923879
.float 0.914210
.float 0.903989
.float 0.893224
.float 0.881921
.float 0.870087
.float 0.857728
.float 0.844853
.float 0.831469
.float 0.817584
.float 0.803207
.float 0.788346
.float 0.773010
.float 0.757208
.float 0.740951
.float 0.724247
.float 0.707106
.float 0.689540
.float 0.671558
.float 0.653172
.float 0.634393
.float 0.615231
.float 0.595699
.float 0.575807
.float 0.555569
.float 0.534997
.float 0.514102
.float 0.492897
.float 0.471396
.float 0.449610
.float 0.427554
.float 0.405240
.float 0.382682
.float 0.359894
.float 0.336889
.float 0.313680
.float 0.290283
.float 0.266711
.float 0.242979
.float 0.219100
.float 0.195089
.float 0.170960
.float 0.146729
.float 0.122409
.float 0.098016
.float 0.073563
.float 0.049066
.float 0.024540
```

B.7 PROGRAMA 7

Programa main.map

```
******************************************************
TMS320C3x/4x COFF Linker        Version 4.40
******************************************************
Mon Feb 19 17:32:01 1996

OUTPUT FILE NAME:  <main.out>
```

```
ENTRY POINT SYMBOL: "_c_int00"  address: 000009dc


MEMORY CONFIGURATION

         name     origin     length     attributes    fill
       --------  --------   ---------   ----------  --------
       VECS      00000000   000000800     RWIX
       SRAM      00000801   000005000     RWIX
       RAM0      00809800   000000400     RWIX
       RAM1      00809c00   000000400     RWIX


SECTION ALLOCATION MAP

 output                            attributes/
section   page    origin       length       input sections
--------  ----  ----------  ----------   ----------------
.text      0    00000801    00000295
                00000801    00000098     main.obj (.text)
                00000899    00000000     seno.obj (.text)
                00000899    0000005a     appendix.obj (.text)
                000008f3    00000037     begin.obj (.text)
                0000092a    0000004c     recibe.obj (.text)
                00000976    00000064     manda.obj (.text)
                000009da    0000001b     rts30.lib : boot.obj (.text)
                000009f5    00000023               : divi.obj (.text)
                00000a18    00000031               : divu.obj (.text)
                00000a49    00000024               : invf.obj (.text)
                00000a6d    00000029               : exit.obj (.text)

.cinit     0    00809800    0000000b
                00809800    00000004     main.obj (.cinit)
                00809804    00000006     rts30.lib : exit.obj (.cinit)
                0080980a    00000001     --HOLE-- [fill = 00000000]

.stack     0    00809c00    00000400     UNINITIALIZED
                00809c00    00000000     rts30.lib : boot.obj (.stack)

.bss       0    00000a96    00000235     UNINITIALIZED
                00000a96    00000210     main.obj (.bss)
                00000ca6    00000000     rts30.lib : invf.obj (.bss)
                00000ca6    00000000               : divu.obj (.bss)
                00000ca6    00000000               : divi.obj (.bss)
                00000ca6    00000000               : boot.obj (.bss)
                00000ca6    00000000     manda.obj (.bss)
                00000ca6    00000000     recibe.obj (.bss)
                00000ca6    00000000     begin.obj (.bss)
                00000ca6    00000000     seno.obj (.bss)
                00000ca6    00000003     appendix.obj (.bss)
                00000ca9    00000022     rts30.lib : exit.obj (.bss)

.data      0    00000ccb    00000092
                00000ccb    00000000     main.obj (.data)
```

```
         00000ccb  00000000   rts30.lib : exit.obj (.data)
         00000ccb  00000000             : invf.obj (.data)
         00000ccb  00000000             : divu.obj (.data)
         00000ccb  00000000             : divi.obj (.data)
         00000ccb  00000000             : boot.obj (.data)
         00000ccb  00000000   appendix.obj (.data)
         00000ccb  00000080   seno.obj (.data)
         00000d4b  00000006   begin.obj (.data)
         00000d51  00000006   recibe.obj (.data)
         00000d57  00000006   manda.obj (.data)

vecs     0  00000000  00000000   UNINITIALIZED

vectors  0  00000000  00000003
         00000000  00000001   begin.obj (vectors)
         00000001  00000001   recibe.obj (vectors)
         00000002  00000001   manda.obj (vectors)


GLOBAL SYMBOLS

address  name                 address  name
-------- ----                 -------- ----
00000a96 .bss                 00000400 __STACK_SIZE
00000ccb .data                00000801 .text
00000801 .text                00000801 _main
000009f5 DIV_I30              0000089a _fft_rl
00000a18 DIV_U30              000008ff _begin
00000a49 INV_F30              00000934 _recibe
00000a96 _P                   00000981 _manda
00000400 __STACK_SIZE         000009dc _c_int00
00809c00 __stack              000009f5 DIV_I30
00000a91 _abort               00000a18 DIV_U30
00000a7e _atexit              00000a49 INV_F30
000008ff _begin               00000a6d _exit
000009dc _c_int00             00000a7e _atexit
00000c98 _cxx                 00000a91 _abort
00000a97 _data_input          00000a96 .bss
00000a6d _exit                00000a96 etext
0000089a _fft_rl              00000a96 _P
00000801 _main                00000a97 _data_input
00000981 _manda               00000c97 _var
00000934 _recibe              00000c98 _cxx
00000ccb _sine                00000ca3 _ven
00000c97 _var                 00000ccb end
00000ca3 _ven                 00000ccb .data
00809800 cinit                00000ccb _sine
00000d5d edata                00000d5d edata
00000ccb end                  00809800 cinit
00000a96 etext                00809c00 __stack
```

APÉNDICE C

APÉNDICE C

VENTANAS DE DATOS MÁS COMUNES*

NOMBRE	DEFINICIÓN	TRANSFORMADA DE FOURIER
RECTANGULAR	$w[k]=\begin{cases}1 & \lvert k\rvert\le M\\ 0 & \lvert k\rvert>M\end{cases}$	$W(f)=W_R(f)=\dfrac{\sin\pi f(2M+1)}{\sin\pi f}$
BARTLETT	$w[k]=\begin{cases}1-\dfrac{\lvert k\rvert}{M} & \lvert k\rvert\le M\\ 0 & \lvert k\rvert>M\end{cases}$	$W(f)=\dfrac{1}{M}\left(\dfrac{\sin\pi fM}{\sin\pi f}\right)^2$
HANNING	$w[k]=\begin{cases}\dfrac{1}{2}+\dfrac{1}{2}\cos\dfrac{\pi k}{M} & \lvert k\rvert\le M\\ 0 & \lvert k\rvert>M\end{cases}$	$W(f)=\dfrac{1}{4}W_R\left(f-\dfrac{1}{2M}\right)+\dfrac{1}{2}W_R(f)+\dfrac{1}{4}W_R\left(f+\dfrac{1}{2M}\right)$

NOMBRE	DEFINICIÓN	TRANSFORMADA DE FOURIER
HAMMING	$w[k]=\begin{cases} 0.54+0.46\cos\frac{\pi k}{M} & \lvert k\rvert \le M \\ 0 & \lvert k\rvert > M \end{cases}$	$W(f)=0.23W_R\left(f-\frac{1}{2M}\right)+0.54W_R(f)+0.23W_R\left(f+\frac{1}{2M}\right)$
PARZEN (M impar)	$w[k]=\begin{cases} 2\left(1-\frac{\lvert k\rvert}{M}\right)^3-\left(1-2\frac{\lvert k\rvert}{M}\right)^3 & \lvert k\rvert \le \frac{M}{2} \\ 2\left(1-\frac{\lvert k\rvert}{M}\right)^3 & \frac{M}{2}<\lvert k\rvert \le M \\ 0 & \lvert k\rvert > M \end{cases}$	$W(f)=\frac{8}{M^3}\left(\frac{3}{2}\frac{\sin^4 \pi f M/2}{\sin^4 \pi f}-\frac{\sin^4 \pi f M/2}{\sin^2 \pi f}\right)$

*KAY, S. M., Modern Spectral Estimation, .Prentice Hall Signal Processing Series, Alan V. Oppenheim, Series Editor, 1988.