UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

FACULTAD DE INGENIERIA

ALGORITMOS GENETICOS PARALELOS EN ESTIMACION ESPECTRAL UTILIZANDO UNA ARQUITECTURA HETEROGENEA

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A

MARCO ANTONIO VIGUERAS VILLASEÑOR

DIRECTOR DE TESIS: DR. JULIO SOLANO GONZALEZ

MEXICO, D. F. 1996

El primero de la estirpe está amarrado en un árbol y al último se lo están comiendo las hormigas.

Gabriel García Márquez.
(Cien años de soledad)

La invención del alma por el hombre se insinúa cada vez que surge el sentimiento del cuerpo como parásito, como gusano adherido al yo.

Julio Cortázar.
(Rayuela)

Dedicatorias.

...y por esto y muchas cosas más...a mis padres, con cariño, Rogelio y Martha.

Por el orgullo de decir que soy su hermano...a mis hermanos, pues. Gaby, Dany, Rosy, Mony y Adry.

A esa mujer que me hace inquietar...con locura, pasión y frenesí...Karina.

Por esas grandes aventuras de pequeños...José Luis y Julio César.

...y a esas malas compañías...mis amigos universitarios.

Agradecimientos.

Al Dr. Julio Solano por su valioso apoyo en la realización de esta tesis.

A Luis Aguirre, con quien compartí el Nodo Heterogéneo.

Al Departamento de Electrónica y Automatización del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas por las instalaciones en las cuales se elaboró este trabajo.

A DGAPA (Proyecto PAPIID-D0303593) y CONACYT (Proyecto 2146P-A9507 y PACIME-F284-A9209) por haber proporcionado los recursos para la elaboración de este trabajo.

A la UNAM...por existir.

PUBLICACIONES

El trabajo presentado en esta tesis ha generado información publicada en los siguientes artículos:

García Nocetti, D. F., Solano, J., Martínez, J., Ramos, D. "*Heterogeneous Parallel Architecture for Improving Signal Performance in Doppler Blood Flow Instrumentation*". 10th. International Parallel Processing Symposioum. April, 1996. Honolulu, Hawai.

Solano, J., García Nocetti, D.F., Rodríguez, K. Ramos, D. "*Parallel Genetic Algorithms in Autorregresive Modelling Using a Heterogeneous Architecture*". International Federation of Automatic Control. June, 1996. San Francisco, USA.

CONTENIDO

CAPITULO I. INTRODUCCION. 1
I.1. MOTIVACION 1
I.2. OBJETIVOS 4
I.3. CONTENIDO 5

CAPITULO II. PROCESAMIENTO PARALELO. 8
II.1. INTRODUCCION 8
II.2. PROCESAMIENTO PARALELO 8
II.2.1. Definición 9
II.2.2. Clasificación 11
II.2.3. El Transputer y su lenguaje OCCAM 12
II.3. EL PROCESADOR TMS320 24
II.3.1. Microprocesadores para Procesamiento de Señales 24
II.3.2. La familia TMS320 26
REFERENCIAS 29

CAPITULO III. ALGORITMOS GENETICOS. 31
III.1. DEFINICION 31
III.2. TEORIA DE LA EVOLUCION 32
III.3. HISTORIA DE LOS ALGORITMOS GENETICOS 35
III.4. TERMINOLOGIA 36
III.5. DIFERENCIA CON OTROS METODOS 37
III.6. ALGORITMO GENETICO BASICO 38

CONTENIDO

III.7. ESQUEMAS DE REPRODUCCION 39
III.7.1. Selección 40
III.7.2. Reproducción 42
III.8. OPERADORES GENETICOS 43
III.8.1. Cruzamiento 43
III.8.2. Mutación 45
III.9. TEOREMA DEL ESQUEMA 46
III.10. ALGORITMOS GENETICOS PARALELOS 50
REFERENCIAS 55

CAPITULO IV. ESTIMACION ESPECTRAL PARAMETRICA. 59
IV.1. INTRODUCCION 59
IV.2. MODELOS PARAMETRICOS AR, MA Y ARMA 62
IV.3. METODO DE COVARIANCIA MODIFICADA 71
REFERENCIAS 77

CAPITULO V. ESTIMACION ESPECTRAL EN FLUJOMETRIA DOPPLER UTILIZANDO ALGORITMOS GENETICOS.... 79
V.I. INTRODUCCION 79
V.2. ANALISIS ESPECTRAL DE SEÑALES DOPPLER 80
V.3. ESTIMACION ESPECTRAL UTILIZANDO AG's 82
V.3.1. Medidor Doppler de Flujo Sanguíneo 82
V.3.2. Modelo Paramétrico..... 82
V.3.3. Implementación del Modelo Paramétrico Autorregresivo usando Algoritmos Genéticos 83
REFERENCIAS 87

CONTENIDO

CAPITULO VI. ESTIMACION ESPECTRAL PARAMETRICA EN TIEMPO REAL. 88

VI.1. INTRODUCCION 88

VI.2. REDUCCION DE LA FUNCION OBJETIVO UTILIZANDO TERMINOS DE LA MATRIZ DE COVARIANCIA MODIFICADA . 89

VI.3. ALGORITMOS GENETICOS, REPRESENTACION REAL 91

VI.4. IMPLEMENTACION UTILIZANDO ALGORITMOS GENETICOS PARALELOS..... 92

VI.5. ESTIMACION ESPECTRAL PARAMETRICA, ESQUEMA HOMOGENEO..... 94

REFERENCIAS 96

CAPITULO VII. ESTIMACION ESPECTRAL PARAMETRICA, ESQUEMA HETEROGENEO. 97

VII.1. INTRODUCCION 97

VII.2. ESQUEMA HETEROGENEO 98

VII.2.1. Nodo de Procesamiento Heterogéneo (NPH) 99

VII.2.2. Interfaz de Comunicación 100

VII.3. ANALISIS DE DESEMPEÑO DEL ESQUEMA HETEROGENEO . 103

VII.3.1. Esquema Heterogéneo con NPH ejecutando la PSD . 103

VII.3.2. Esquema Heterogéneo con NPH en el Cálculo de los Parámetros 106

VII.3.3. Esquema Heterogéneo con NPH por mapeo de memoria 108

REFERENCIAS 112

CONTENIDO

CAPITULO VIII. CONCLUSIONES. 113

VIII.1. CONCLUSIONES GENERALES 113

VIII.2. TRABAJO FUTURO 115

APENDICE A. PROGRAMAS, ESQUEMA HETEROGENEO. 117

A.1. ESQUEMA HETEROGENEO CON EL NPH EJECUTANDO LA PSD 117

A.2. ESQUEMA HETEROGENEO CON EL NPH EN EL CALCULO DE LOS PARAMETROS 125

APENDICE B. PROGRAMAS DE COMUNICACION T805-TMS320C30 .. 131

CAPITULO I.
INTRODUCCION

I.1 MOTIVACION.

En la actualidad, las enfermedades cardiovasculares son la causa de un elevado número de decesos en la población que son víctimas de este tipo de padecimientos. Dispositivos como el Detector de Flujo Sanguíneo por efecto Doppler son comúnmente utilizados para diagnosticar y monitorear la presencia de este tipo de problemas en una etapa temprana.

La disminución en el diámetro de arterias principales del cuerpo humano a causa de acumulación de placa en sus paredes, causa que la distribución de velocidad del Flujo Sanguíneo se perturbe. Así, es importante que un sistema médico de medición sea capaz de medir tales perturbaciones de tal forma que se pueda determinar el grado de estenosis que un paciente presenta. En consecuencia, es fundamental el utilizar esquemas apropiados de procesamiento para extraer información cuantitativa para su diagnóstico.

La gran mayoría de los sistemas de Ultrasonido Doppler disponibles, hacen uso de la Transformada Rápida de Fourier en segmentos de señal consecutivos o traslapados activamente. Sin embargo, los métodos basados en la FFT presentan problemas inherentes al proceso, tales como, la limitación en cuanto a resolución en frecuencia, es decir, la habilidad para distinguir respuestas espectrales entre dos o más señales; y la filtración en el dominio espectral debido al ventaneo de datos.

En un intento para solucionar las limitantes de la FFT, diversas investigaciones realizan la Estimación Espectral basada en Métodos Paramétricos, los cuales, presentan mejoras significativas en su resolución espectral en tiempo-frecuencia. Dichos métodos se basan principalmente en la estimación de la señal por medio del conocimiento inicial de una ventana de datos en un tiempo dado. Dentro de estos métodos se identifica al Método de Covariancia Modificada (un método Paramétrico Autorregresivo) que presenta la ventaja de ser computacionalmente menos complejo comparado con otros métodos paramétricos en donde su solución depende de resolver sistemas de ecuaciones no-lineales.

El método de Covariancia Modificada se basa en la minimización de una función de error, dicha solución se puede obtener por medio del método de Cholesky, desarrollando y resolviendo un conjunto de ecuaciones lineales. No obstante que existe una gran variedad de técnicas como las descritas, es necesario llevar a cabo un esfuerzo para mejorar diversos aspectos de la Estimación Espectral, ya sea en resolución o velocidad de procesamiento, tal es el caso del método alternativo presentado en esta tesis.

Trabajos previos han explorado nuevos métodos alternativos que, por su sencillez, presentan ventajas sobre los métodos tradicionales. Un método basado en Algoritmos Genéticos ha sido utilizado para encontrar el conjunto óptimo de parámetros asociados al filtro del modelo paramétrico empleado, mostrando la factibilidad de reducir la complejidad computacional de la solución, aprovechando las características inherentes a los Algoritmos Genéticos.

Sin embargo, y a pesar de la buena respuesta espectral mostrada por el método, los diferentes procesos involucrados en el AG muestran una respuesta alejada a los requerimientos del sistema médico en estudio (2-20 ms.).

De acuerdo a lo anterior y utilizando computadoras bajo la arquitectura Von Neumann, sería prácticamente imposible satisfacer tales requerimientos.

La alternativa más viable es por tanto, el uso de Procesamiento Paralelo. Notables desarrollos como el Transputer de INMOS y su lenguaje asociado OCCAM, pueden ser considerados como un soporte importante en la aplicación efectiva y a bajo costo de dicha alternativa.

El trabajo descrito en esta tesis extiende el método genélico, explotando su naturaleza paralela. El poder operar diferentes subpoblaciones en forma independiente, no sólo reduce el tiempo de ejecución del problema, sino que además, mejora la calidad de la solución.

Se presentan dos esquemas de solución paralela para obtener una respuesta del sistema en tiempo-real: el Esquema Homogéneo y el Esquema Heterogéneo.

El Esquema Homogéneo, se basa en una estructura de pipeline con tres procesadores Transputer en anillo para el cálculo de los parámetros del filtro y, un procesador Transputer en la última etapa para el cálculo de la PSD.

El Esquema Heterogéneo, aunque similar en estructura al Esquema Homogéneo, incorpora un Nodo de Procesamiento Heterogéneo (Transputer-DSP) que puede ser fácilmente integrado a un sistema basado en Transputers e incluye bibliotecas para aceptar y ejecutar código específico para el DSP en forma transparente desde el Transputer. Con el uso de este nuevo esquema se pretende disminuir aun más el tiempo de procesamiento del sistema obtenido, reduciendo también, el costo que involucraría lograr los mismos resultados aumentando procesadores en el Esquema Homogéneo.

Esta implementación ha sido propuesta para ser usada por un Analizador Espectral Doppler Ultrasónico, el cual calculará y desplegará, en tiempo-real, el contenido de la frecuencia de Señales Doppler Ultrasónicas para mediciones médicas.

I.2. OBJETIVOS.

El objetivo principal en el desarrollo de esta investigación es:

Realizar la Estimación Espectral Paramétrica de señales utilizadas en Flujometría Doppler, por medio de Algoritmos Genéticos Paralelos en tiempo-real, utilizando para ello un Esquema Heterogéneo.

Para poder llevar a cabo este objetivo se plantean los siguientes objetivos secundarios:

Estudio y Evaluación del TMS320C30 para procesamiento de señales en tiempo-real respecto al procesamiento en el transputer T805.

Estudio y Evaluación del Nodo Heterogéneo para procesamiento de señales en tiempo-real respecto al procesamiento en el transputer T805.

Implementación de Algoritmos Genéticos Paralelos en tiempo-real sobre un Esquema Heterogéneo (TMS320C30-Transputers).

I.3. CONTENIDO.

El trabajo presentado en esta tesis se encuentra organizada de la siguiente manera:

En el capítulo uno se presenta una introducción general, enfatizando las razones que motivaron este trabajo. Incluye también los objetivos principales, así como el contenido de la misma.

Para el desarrollo del trabajo, se presentan primeramente los antecedentes necesarios para poder entender los diferentes conceptos a utilizar, como es el caso de Procesamiento Paralelo, Transputer y OCCAM, Procesadores Digitales de Señales, Algoritmos Genéticos y Estimación Espectral contenido en los capítulos dos, tres y cuatro.

En el capítulo cinco se describe el contexto en el cual se desarrollará esta tesis, así como su implementación por medio de los Algoritmos Genéticos.

El capítulo seis presenta la implementación de un Estimador Espectral Paramétrico en tiempo-real. Este capítulo describe el Esquema Homogéneo utilizado, así como, el desarrollo de las diferentes consideraciones necesarias para lograr la respuesta en tiempo-real.

El capítulo siete presenta la implementación del Esquema Heterogéneo propuesto en los objetivos de esta tesis. Asimismo, describe el Nodo de Procesamiento Heterogéneo utilizado, describiendo las diferentes interfaces utilizadas para el logro de una comunicación eficiente con el Transputer. Se presenta también, un análisis de balanceo de cargas, fundamental en la implementación eficiente del esquema.

Finalmente, en el capítulo ocho se muestran las conclusiones generales de este trabajo, así como algunas líneas de investigación a seguir en un futuro.

CAPITULO II.
PROCESAMIENTO PARALELO

II.1. INTRODUCCION.

El presente capítulo proporciona al lector los antecedentes esenciales de las áreas de procesamiento paralelo utilizadas en esta tesis de tal manera que, los capítulos subsecuentes puedan ser mejor apreciados.

Los temas que se incluyen son: Procesamiento Paralelo, Transputer y OCCAM y El procesador TMS320.

II.2. PROCESAMIENTO PARALELO.

La investigación llevada a cabo dentro del área de Procesamiento Paralelo presenta en la actualidad una gran importancia. Gracias a los avances tecnológicos que se han venido desarrollando, es posible resolver problemas que en el pasado no eran considerados por su gran demanda computacional.

Un ejemplo claro de este tipo de aplicaciones, lo constituye el Procesamiento Digital de Señales Doppler Ultrasónicas, tema tratado en esta tesis. Aquí, los requerimientos tanto de resolución como de respuesta en tiempo real hacen que el procesamiento paralelo juegue un papel relevante para su implementación.

II.2.1. Definición.

El Procesamiento Paralelo puede definirse como una colección simple de procesadores, conectados de una forma confiable para coordinar sus actividades y el cambio de datos [6], trabajando juntos para resolver un problema en común [1].

Una de las diferencias más importantes entre un procesador capaz de trabajar en paralelo y un procesador secuencial es que el segundo se basa en la arquitectura Von Neumann, modelo general de las computadoras secuenciales actuales, en donde la medida típica de desempeño es el tiempo tomado en ejecutar una operación aritmética o el número de operaciones aritméticas ejecutadas en un segundo [10], además de que, en una arquitectura con estas características se presenta un solo procesador, memoria y dispositivos de E/S ligada a un sólo bus de datos. En un procesador paralelo, esto se puede resolver ya que su forma de trabajar permite ejecutar diferentes partes de un problema al mismo tiempo. Pensemos en la suma de dos números A y B.

$$A=P \times 2^{Q} \qquad y \qquad B=R \times 2^{S}.$$

Si estamos trabajando en una computadora secuencial, primeramente se tendrá que determinar la diferencia de (Q-S) para determinar el número de lugares que está desplazado el bit más significativo de A con respecto a B.
Se suman las mantisas P y R.
Se normaliza el resultado por si existe un overflow.

Si el tiempo que tarda en realizar cada uno de los pasos es T, el tiempo del proceso será 3 x T y si se quiere realizar la suma N veces, el tiempo total de procesamiento será de (3xT) x N. Si el número de pasos es D, entonces el tiempo será de DxTxN.

En una arquitectura paralela, el sistema trabaja de la siguiente forma: cuando es completado el primer paso de la primera suma, se puede ejecutar el segundo paso de la primera suma y el primer paso de la segundo suma, así pues, cuando el procesador ejecuta el tercer paso de la primera suma, es capaz de ejecutar el segundo paso de la segunda suma y el primer paso de la tercer suma. Si se hacen las consideraciones anteriores, el tiempo será de DxT + (N-1)xT, con lo cual, el tiempo se reduce. Esto es conocido como procesamiento en Pipeline (procesamiento en línea). En la figura 2.1 se muestra como en una arquitectura de procesamiento paralelo se puede realizar el bloque de operaciones en un tiempo menor que en una arquitectura secuencial, ya que mientras en una arquitectura secuencial se ejecutan dos eventos, en una arquitectura paralela se ejecutan cuatro eventos (tomando en cuenta que cada evento se realiza en tres pasos) [10].

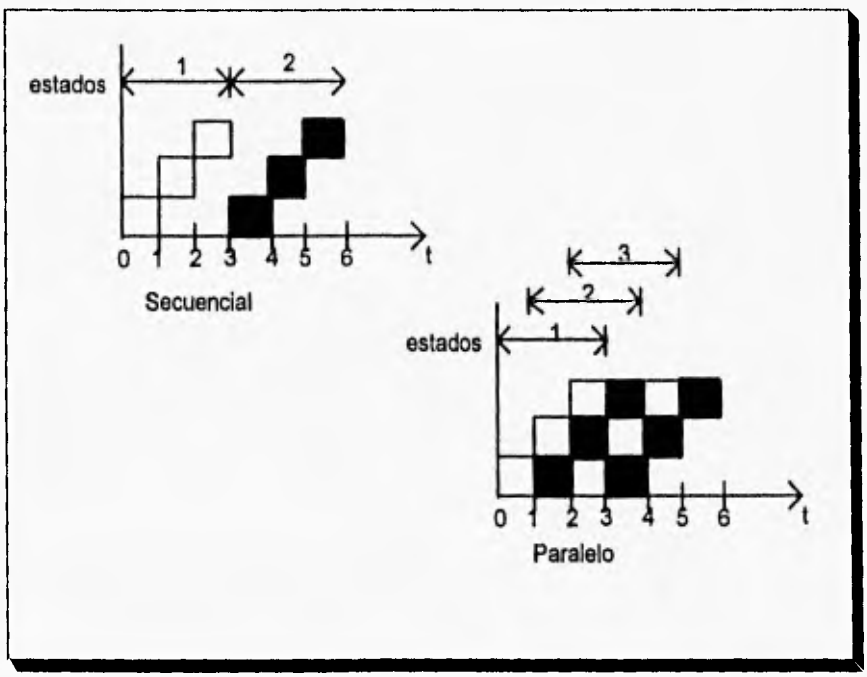


Figura 2.1. Procesamiento secuencial vs. Procesamiento Paralelo.

Se dice que existen dos tipos de procesamiento paralelo, el procesamiento paralelo macroscópico y el procesamiento paralelo microscópico [8].

El paralelismo macroscópico se basa en la partición explícita del trabajo que hay que efectuar en módulos independientes llamados *tareas o procesos*. En el enfoque microscópico no se plantea la hipótesis de una descomposición explícita, el programa se considera como un todo, pero durante la ejecución, mecanismos complejos determinan las relaciones de dependencia que existen entre las acciones elementales que hay que efectuar y, por lo tanto, las que pueden efectuarse al mismo tiempo. Generalmente las operaciones que se pueden ejecutar en paralelismo microscópico son el ciclo de fetch, decode y execute, con lo cual, mientras se ejecuta el ciclo de execute de una instrucción, se está ejecutando el ciclo de decode y el ciclo de fetch de dos instrucciones siguientes

II.2.2. Clasificación.

Dentro del llamado procesamiento paralelo macroscópico, existe una clasificación de arquitecturas de procesadores paralelos desarrollada por Flynn [2]. En ella, hace una diferencia de arquitecturas de acuerdo al manejo de instrucciones y datos:

SISD (Single Instruction, Single Data). Las instrucciones se ejecutan secuencialmente sobre un mismo conjunto de datos

SIMD (Single Instruction, Multiple Data). En esta arquitectura existe una sola secuencia de instrucción, trabajando sobre diferentes datos.

MISD (Multiple Data, Single Data). En esta arquitectura se tienen varios procesadores ejecutando diferentes instrucciones sobre una misma secuencia de datos.

MIMD (Multiple Instruction, Multiple Data). Consiste en varios procesadores independientes capaces de ejecutar instrucciones en forma independiente sobre diferentes datos.

Como ejemplo de las arquitecturas MIMD se tiene el transputer utilizado en el desarrollo del trabajo presentado en esta tesis.

II.2.3. El Transputer y su lenguaje OCCAM.

El Transputer

El Transputer es un microprocesador desarrollado por la firma INMOS en 1984 [13], su nombre viene de la unión de TRANSistor y comPUTER [4]. El Transputer es una nueva generación de arquitecturas VLSI de 16 a 32 bits de palabra que soporta la concurrencia y la sincronización. La sincronización se requiere cuando diferentes partes de un sistema, cada uno trabajando en diferentes procesadores, necesitan ser conectados para intercambiar información. Cuenta con links de comunicación que trabajan en full duplex utilizados para realizar el intercambio de datos entre varios transputers, para así, poderse distribuir tareas logrando un eficiente paralelismo macroscópico, mientras que, por otro lado, el transputer puede ejecutar un paralelismo microscópico entre el ciclo de fetch, decode y execute interno. El Transputer cuenta con memoria propia, con lo cual, cada uno de ellos contiene un código de programa a ejecutar y datos independientes que pueden ser compartidos únicamente por medio de la comunicación por los canales.

La arquitectura general del transputer se observa en la figura 2.2 [1].

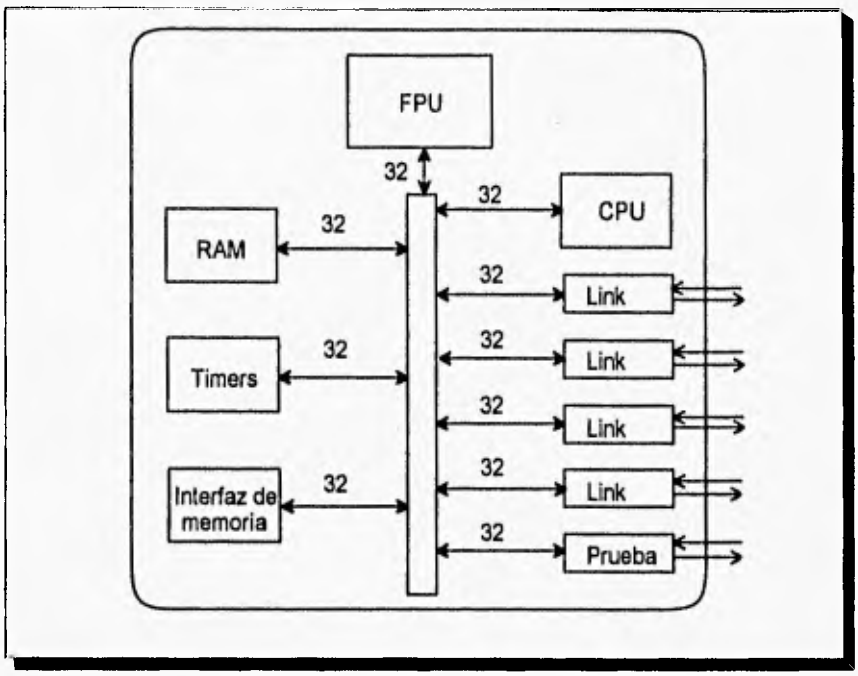


Figura 2.2. Arquitectura del Transputer.

El set de instrucciones es independiente de la longitud de palabra del procesador; todas las instrucciones son de un byte de longitud, cuatro bits de código y cuatro bits de datos.

Soporta el procesamiento concurrente, ya que cuenta con dos prioridades de procesamiento (prioridad alta como 0 y prioridad baja como 1). Es posible ejecutar dos procesos concurrentemente, el proceso que corre en prioridad baja sólo podrá ser ejecutado si el proceso que corre en prioridad alta está desocupado, si son más de dos procesos los que se quieren correr en forma concurrente, se hace uso de una cola de espera para que el procesador se haga cargo de cada proceso en diferentes tiempos.

Los links de comunicación son los canales por los cuales un transputer puede intercambiar información, ya sea con la memoria del mismo (canal interno) o con otro transputer (canal externo). Los canales de comunicación envían la información en palabras de 8 bits (un byte), así, si se requiere mandar un datos de 32 bits, es necesario mandarlo en cuatro bytes; dichos canales de comunicación envían la información en forma serial y trabajan en forma síncrona. Como cada canal cuenta con memoria reservada, un canal es identificado por su dirección de memoria, con lo cual, el proceso discrimina entre los dos tipos de canales. Esto indica que el proceso puede ser implementado con la misma instrucción tanto para canales internos como externos.

Algunos modelos de transputers (T800, T805 y T801) cuentan con una Unidad de Punto Flotante (FPU). La FPU está formada por un microcódigo que puede operar concurrentemente bajo el control del CPU. Tanto el CPU como la FPU pueden ejecutar procesos en forma paralela de datos en punto flotante haciendo más rápido el proceso. La FPU funciona bajo el estándar 754-1985 de la IEEE. Hay que aclarar que el proceso en la FPU no es visible al programador, es el transputer el encargado de mandar los procesos a la FPU o ejecutarlos en el propio CPU.

El primer transputer de INMOS fue el T424 (1984) que contenía 4 Kbyte de memoria RAM, dicha memoria fue reducida a 2 Kbyte para dar paso al T414 (1985), el cual es considerado el primer transputer como tal, con 32 bits de palabra, 2K de RAM, cuatro links de comunicación a 10 Mbit/s pero sin FPU. En 1987 aparece el T800, el cual, es compatible con el T414, con 32 bits de palabra, cuenta con su FPU, capacidad de RAM de 4K y un protocolo de comunicación de canal que se transmite antes de mandar los datos, al igual que el T414, su velocidad de transmisión por los links es de 5/10/20 Mbit/s, pero con el protocolo, evita el traslape de información, con lo cual, hace posible que la información viaje a 2.4 Mbytes/s a diferencia de 1.2 Mbytes/s en que trabajaba el T414. después del T800, vinieron los T805, T425 y el T225 que cuenta con una extensión de eventos e interfaz de memoria, un debugger y utiliza

necesariamente el protocolo de información. Después de estos tipos de transputers, han seguido otros como el M212 que son considerados como de propósito especial y que cuenta por ejemplo con posibilidades de manejo de disco.

El T805, que es el transputer que se utilizará en este trabajo, es un procesador de 32 bits de palabra, cuenta con un ciclo de reloj de 33 ns. Es capaz de realizar 30 MIPS (millones de instrucciones por segundo) y 4.3 MFLOPS (millones de operaciones en punto flotante por segundo). Es un transputer compatible con el T800, T425, T400 y T414. Contiene 4Kbytes de memoria RAM estática y cuatro links de comunicación a 5, 10 ó 20 Mbit/s.

El transputer como tal, es un procesador utilizado en diversos campos de investigación como es el procesamiento digital de señales e imágenes, simulación de sistemas, telecomunicaciones, robótica, reconocimiento de patrones e inteligencia artificial por citar algunos ejemplos.

Debido a que el transputer es un procesador diseñado para trabajar en forma paralela junto con otros transputers (generalmente) para la solución de un problema común, es necesario conocer diferentes esquemas de paralelismo que pueden ser implementados de acuerdo al problema que se esté tratando. A continuación presentamos los más usuales.

Esquemas de Paralelismo

Básicamente podemos hablar de tres esquemas de paralelismo que son los más utilizados en el procesamiento paralelo con Transputers, el esquema Farming, el esquema Pipeline y el esquema de Anillo.

El esquema Farming está formado por un nodo maestro que se encarga de distribuir el trabajo a los nodos esclavos. El nodo maestro se encarga de mandar tareas a través de un canal; el nodo esclavo que esté libre recibe la tarea del canal, la ejecuta, manda los resultados al maestro por otro canal y se pone en la espera de ejecutar más tareas. Todos los esclavos realizan el mismo proceso, con lo cual, no existe una comunicación directa entre el maestro y un esclavo, la única comunicación es por medio de los canales [14].

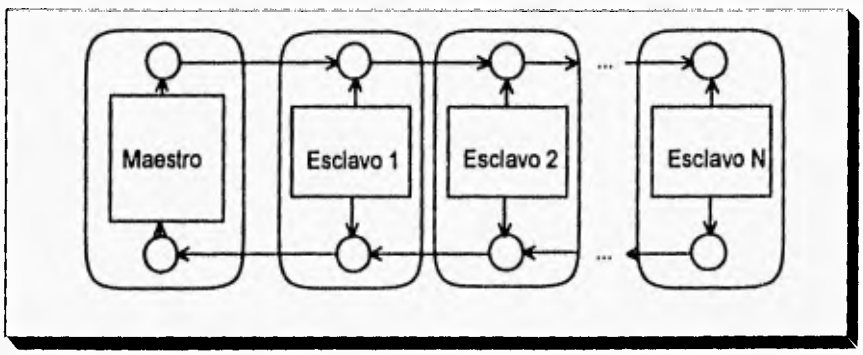


Figura 2.3. Esquema Farming.

En el esquema Pipeline [7], se trata de simular el procesamiento paralelo microscópico, con la diferencia de que ahora los pasos a seguir no estarán formados por una microinstrucción sino que estarán formados por procesos completos. El esquema Pipeline trabaja de igual forma que una línea de producción de cualquier producto. La materia prima (los datos), pasan por una serie de procesos (alojados en los Transputers) por medio de una banda (canales de comunicación) para finalmente obtener el producto terminado en forma secuencial. El esquema Pipeline se muestra en la figura 2.4.

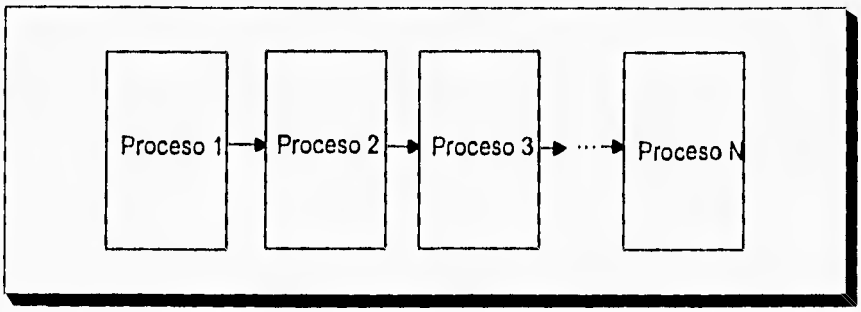


Figura 2.4. Esquema Pipeline.

En el esquema de Anillo, los Procesos se encuentran unidos unos con otros formando un anillo. Se puede hacer la semejanza con una mesa circular, en donde se sientan personas alrededor y se puede intercambia la información con el que se encuentra a la derecha o a la izquierda de uno (como se verá más adelante, este esquema será muy útil en la implementación del esquema de Algoritmos Genéticos Paralelos). La figura 2.5 muestra este esquema.

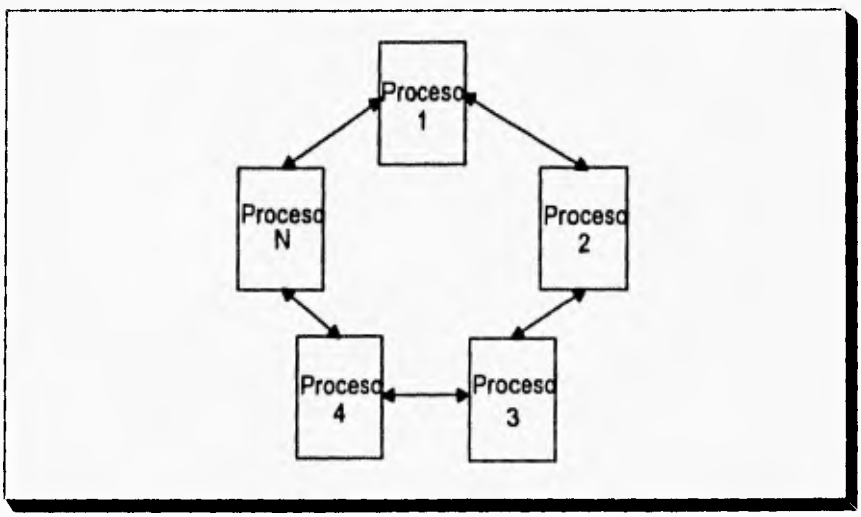


Figura 2.5. Esquema de Anillo.

Lenguaje OCCAM.

Hasta ahora se ha descrito en términos generales lo que es el procesamiento paralelo y el procesador Transputer, pero es necesario conocer la forma en que se debe ejecutar un programa ya sea en forma secuencial, concurrente o paralela en uno o en varios procesadores. Para ello tendremos que conocer el lenguaje OCCAM, lenguaje realizado para el Transputer. Cabe aclarar que aunque el lenguaje OCCAM fue diseñado para el transputer, actualmente existen otros lenguajes como el C paralelo o el C++ paralelo.

El lenguaje OCCAM fue diseñado para soportar procesamiento concurrente. El nombre de OCCAM y sus principios para su desarrollo son tomados de William Occam, el cual se hizo famoso por su razonamiento conocido como Occam's Razor: *"Entia non sunt multiplicanda preater necessitatem"*, que traducido del latín quiere decir "todos los fenómenos no deben ser explicados más allá de lo necesario" [4]. Es decir, en nuestro contexto, es posible resolver un problema complicado dividiéndolo en subproblemas sencillos. Es así como funciona el lenguaje OCCAM. La primera versión de OCCAM (OCCAM 1) fue introducida en 1982. En 1986 se introdujo la segunda versión (OCCAM 2) que tenía las siguientes ventajas con respecto al anterior:

Un incremento en el tipo de datos que puede manejar.

Un cambio completo en la sintaxis para declarar constantes y variables dentro de un programa.

La introducción de protocolos para describir la transferencia de datos por los canales (un link del transputer corresponde a un canal en OCCAM).

El lenguaje OCCAM funciona por procesos y se basa en tres primitivas primordiales [4] [10] [11]:

-Asignación de un valor o expresión a una variable:

variable :=expresión

-Declaraciones de entrada, que espera hasta que un valor es recibido desde un canal para asignárselo a una variable.

canal ? variable

-Declaraciones de salida, que manda un valor a través de un canal para ser recibido en otro proceso.

canal ! variable

Algunos autores como Carlini [1] señalan también como primitivas, agregando a las anteriores:

-El proceso SKIP, que es un proceso que comienza y termina, sin ejecutar alguna acción.

-El proceso STOP, que comienza, no ejecuta acción alguna y nunca termina.

Para poder utilizar estas primitivas OCCAM se basa en tres constructores disponibles:

-El constructor Secuencial indicado por **SEQ**, que indica que los componentes del proceso son ejecutados uno después del otro, comenzando cuando el componente anterior termina y terminando cuando el último componente se haya ejecutado.

Un ejemplo de la forma como trabaja el constructor SEQ es:

```
SEQ
  ch1 ? x
  x := x + 1
  ch2 ! x
```

En este sencillo ejemplo, lo que hace el procesador es recibir por el canal ch1 el valor de x, incrementa el valor de x y manda por el canal ch2 el nuevo valor de x. Todo lo hará en forma secuencial.

-El constructor Paralelo indicado por **PAR**, que causa que sus componentes puedan ser ejecutados concurrentemente, terminando cuando todos los componentes hayan terminado.

Un ejemplo del constructor PAR es:

```
PAR
  x:=x+1
  y:=y+2
```

En donde ejecutará concurrentemente las dos operaciones, es decir, la suma de x más uno y la suma de y más dos.

Una alternativa del constructor PAR es el PRI PAR, en donde se podrán ejecutar dos procesos con diferente prioridad. Anteriormente se había explicado que el transputer puede ejecutar procesos con prioridad alta y prioridad baja. Con PRI PAR puede ser posible esto de la siguiente manera:

```
PRI PAR
  p.alta()
  p.baja()
```

En donde el procesador ejecutará los dos procesos con una prioridad alta para p.alta() y una prioridad baja para p.baja(), así, el proceso p.alta() será siempre ejecutado con preferencia sobre p.baja().

-El constructor Alternar indicado por **ALT**, que selecciona uno de sus procesos componentes para ser ejecutado terminando cuando el proceso seleccionado haya terminado.

Para poder utilizar el constructor ALT es necesario preguntar si el proceso a realizar puede ser ejecutado, veamos el ejemplo:

```
ALT
   ch1 ? x
     x := x + 1
   ch2 ? y
     y := y + 1
```

En donde, uno de los dos procesos (x:=x+1 o y:=y+1) será ejecutado cuando el canal asociado a cada uno de ellos esté listo, es decir, si el canal ch2 trae información que se le asocia a y antes que ch1 tenga la información, entonces se ejecutará el proceso y:=y+1 y terminará el constructor ALT.

Estos tres constructores se pueden intercalar para formar procesos más elaborados. Por ejemplo:

```
SEQ
   PAR
      SEQ
         x:=1
         ch1 ! x
```

```
SEQ
  ALT
    ch2 ? y
      ch4 ! y
    ch3 ? z
      ch4 ! z
```

En donde todo el proceso ejecutará en forma secuencial dos procesos paralelos, el primero ejecutará en forma secuencial una asignación de variable y mandará la información por ch1. El segundo proceso mandará ya sea y o z por el canal ch4 dependiendo de si llega la información por ch2 o por ch3.

Cuando varios procesos compiten por recursos es posible que se dé una situación en la que ninguno de ellos pueda proseguir debido a que los recursos que cada uno de ellos necesita estén ocupados por otros. Esta situación se conoce con el nombre de *deadlock* [8]. El evitar un deadlock es tarea del programador y es por eso que hay que tener mucho cuidado en no caer en este tipo de problemas. Un ejemplo claro es el siguiente:

```
--PROCESO 1          --PROCESO 2
SEQ                  SEQ
  ch1 ? x              ch2 ? y
  ch2 ! x              ch1 ! y
```

Los dos procesos se ejecutan secuencialmente: el primero esperará un valor por el canal ch1, el cual será mandado por el proceso 2 después de que reciba un valor por el canal ch2, el cual será mandado por el proceso 1 después de que reciba.... Como se observa, ninguno de los dos procesos se podrá ejecutar porque surgió un deadlock. Gráficamente se puede representar como lo muestra la figura 2.6.

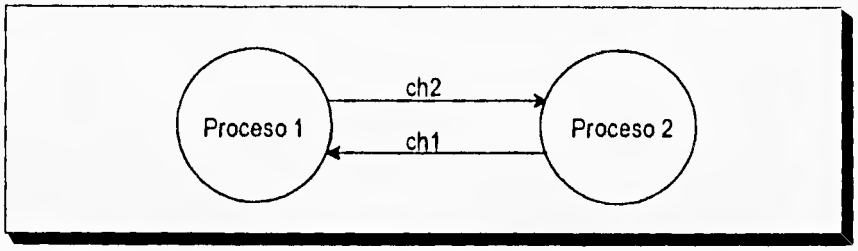


Figura 2.6. Representación de un Deadlock.

Las variables utilizadas en un programa en OCCAM pueden tener la longitud que uno desee, el primer carácter debe ser una letra y los siguientes caracteres pueden ser letras o números. OCCAM hace diferencia entre mayúsculas o minúsculas.

Los tipos de datos con que cuenta OCCAM son [5]:

INT, INT16, INT32, INT64 para diferentes tamaños de enteros

REAL32, REAL64 para diferentes tamaños de números reales.

BYTE para un entero entre 0 y 255, usado generalmente para representar caracteres en su valor ASCII

BOOL para un valor de verdad lógico, TRUE o FALSE.

OCCAM cuenta con un TIMER que avanza cada microsegundo, de gran utilidad para la medición de procesos.

Al momento de definir canales de comunicación, estos deben de llevar un protocolo en los procesos en los que se utilizan los canales para que así no surjan incompatibilidad de datos:

```
CHAN OF INT ch1:        --Define un canal de enteros
CHAN OF REAL32 ch2:     --Define un canal de reales tipo REAL32
CHAN OF ANY ch3:        --Define un canal que puede mandar cualquier tipo de datos
```

Como otros lenguajes de programación, OCCAM cuenta con condicionales (IF, ELSE), con ciclos (FOR, WHILE), y una gran cantidad de instrucciones que ayudan a programar en una forma sencilla.

II.3. EL PROCESADOR TMS320.

II.3.1. Microprocesadores para Procesamiento de Señales.

A medida que surgen nuevos problemas, se van desarrollando nuevos procesadores capaces de resolverlos. Dentro del procesamiento digital de señales, las herramientas matemáticas utilizadas son complejas y su tiempo de procesamiento muy costoso. Varias de las operaciones empleadas como la Transformada Rápida de Fourier, la Convolución, etc. consumen un tiempo considerablemente grande, lo cual, limita su implementación en procesadores convencionales.

Es así como surgen los Procesadores Digitales de Señales (DSP's), procesadores capaces de realizar operaciones matemáticas en un tiempo muy reducido en relación con los procesadores de propósito general.

La familia TMS320 surge desde la década de los ochentas, seguido por el INTEL 2920 y el NEC UPD7720 [3].

Los DSP's son circuitos integrados capaces de realizar operaciones como la multiplicación en tiempos muy cortos, cuentan con chips de conversión A/D y D/A, confiables para desarrollar diversas aplicaciones.

Típicamente cuentan con una Unidad Aritmética Lógica (ALU), un Acumulador (A), Un apuntador a pila (SP), un Registro Indice (IR), Registros de Direccionamiento de Memoria (MAR) y Contador de Programa (PC). Estos microprocesadores difieren entre sí por el número de registros, su arquitectura, la longitud de palabra (4, 8, 16 y 32 bits) y la forma de accesar a memoria.

Algunas familias de DSP's que existen en el mercado son presentadas en la tabla 2.1 [12]. En ella se muestra diferencias entre diferentes familias de DSP's;

Característica	TMS32010	µPD7720	S2811	INTEL2920
Palabra (bits)	16	16	16	25
Saturación Aritmética	Hardware	Software	Hardware	Hardware
Operaciones Booleanas?	Sí	Sí	No	Sí
Implementación Multiplicación	Hardware	Hardware	Hardware	Software
Precisión Multiplicación	16x16=32	16x16=32	12x12=16	12x25=28
Tiempo de Multiplicación (ns)	200	250	300	4.800
Palabra de Instrucción	16	23	17	24
Ciclo de Instrucción (ns)	200	250	300	400
Saltos Condicionales?	Sí	Sí	Sí	No
Instrucciones en ROM (bits)	1536x16	512x23	256x17	192x24

Tabla 2.1. Diferentes familias de DSP's

Como se observa, cuando la multiplicación se realiza por medio de hardware, el tiempo de proceso se reduce considerablemente. Respecto al ciclo de instrucción, el tiempo es menor en la familia TMS320 con respecto a los otros procesadores.

II.3.2. La familia TMS320

La familia de los microprocesadores TMS320 ha ido evolucionando a lo largo de más de una década, está basado en una arquitectura Harvard a diferencia de la arquitectura Von Neumann utilizada en los procesadores de propósito general, lo cual le da una gran flexibilidad y velocidad ya que separa en dos diferentes espacios de memoria el código a ejecutar de un programa y los datos a utilizar en el mismo. Permite trabajar en Pipeline en dos, tres o cuatro niveles . Esto significa que puede procesar dos, tres o cuatro instrucciones en paralelo, como son el ciclo de fetch, decode y execute.

Mientras que los procesadores de propósito general realizan la multiplicación en varios ciclos de reloj, ya que se basan en una serie de sumas para obtener el resultado, el TMS320 ejecuta una multiplicación en un solo ciclo de reloj, ésto gracias a que cuenta con multiplicador a nivel de hardware que se encarga únicamente de dicha operación, asimismo cuenta con un rápido ciclo de instrucción de menos de 200ns muy adecuado para la solución de problemas que requieren una respuesta en tiempo real.

El primer procesador de esta familia fue el TMS32010, cuyas características fueron mostradas en la tabla 2.1. Su desarrollo se llevó a cabo en 1982 y es un microprocesador capaz de realizar cinco millones de operaciones por segundo incluyendo la suma y la multiplicación, es decir, es capaz de realizar la multiplicación en un sólo ciclo de reloj, siendo su ciclo de instrucción de 280 ns, trabaja con una unidad de punto fijo de 16 bits por palabra. También dentro de esta primera generación se puede contar al TMS320C15/E15 y al TMS320C17/E17.

La segunda generación [9], incluye los dispositivos TMS32020 que aparecieron en 1985 y los TMS320C25 en 1986, capaces de desarrollar diez millones de operaciones por segundo, es decir, lo doble que el TMS32010, siendo su ciclo de instrucción de 200 ns., además de expandir el ciclo de memoria y expandir las funciones de entrada/salida. Cuenta con direccionamiento en bit reverse, propio para el cálculo de la FFT, 4K de palabras en ROM y 128K de espacio en memoria para programas, trabajando con una unidad de punto fijo de 16 bits por palabra.

La tercera generación [9], TMS320C30, desarrolla treinta y tres millones de operaciones por segundo, siendo su ciclo de instrucción de 60 ns., trabaja con punto flotante de 32 bits; su desarrollo, velocidad y precisión hace a este procesador tener una alta capacidad de procesamiento, contiene ocho registros auxiliares (AR0-AR7) de 32 bits para propósito general, ocho registros (R0-R7) de 40 bits que son usados para precisión en punto flotante, registros índices (IR0 e IR1), un Stack Pointer (SP), registro de Status del CPU (ST), banderas de interrupción (IE e IF), bandera de entrada/salida (IOF), registro DP que apunta a una de las 256 páginas de datos, RC que especifica el número de veces que se ejecutará un ciclo y, el Contador de Programa (PC).

El TMS320C30 está formado por un CPU que contiene un multiplicador en punto flotante, una Unidad Aritmética Lógica (ALU) para punto flotante, enteros y operaciones lógicas, registros auxiliares de unidades aritméticas y buses asociados. Su espacio de memoria es de 16Mx32 bits, contiene un bloque de memoria ROM de 4Kx32 y dos bloques RAM de 1Kx32.

Cuenta con un chip denominado de Acceso Directo a Memoria (DMA) que es capaz de accesar a memoria para leer o escribir sin necesidad de interferencia por parte del CPU, es decir, puede trabajar en pipeline con el ciclo de fetch, decode y execute. Cuenta con dos módulos de reloj de propósito general interno y externo y con dos puertos seriales totalmente independientes configurados a 8, 16, 24 ó 32 bits de datos.

No se basa en el formato IEEE para representar a los datos sino que utiliza una representación propia. Los datos pueden ser de tres formas diferentes: número en punto flotante de 16 bits, en donde se tienen 4 bits para representar al exponente, 1 bit de signo y 11 bits de mantisa; número en punto flotante de precisión sencilla de 32 bits, con 8 bits de exponente, 1 bit de signo y 23 bits de mantisa y, número en punto flotante de gran precisión con 40 bits, 8 bits de exponente, 1 bit de signo y 31 bits de mantisa.

Su Arquitectura es mostrada en la figura 2.7 [3].

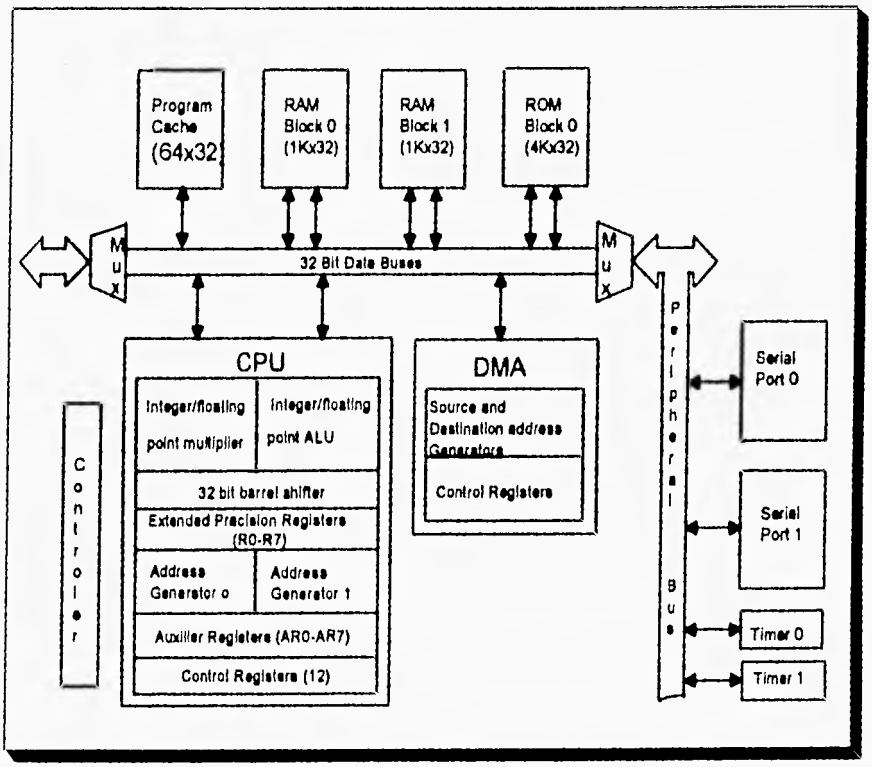


Figura 2.7. Arquitectura del TMS320.

La programación se puede realizar ya sea en lenguaje ensamblador o lenguaje C propio para el microprocesador. Es posible emular la ejecución de un programa por medio del Módulo Evaluador (EVM) que contiene las tareas necesarias para poder emular al micro. El EVM contiene un debug de monitor, editor, ensamblador y comunicación de software con una computadora Host o una línea de impresión.

REFERENCIAS.

[1] Carlini, U.. U. Villano. "Trasputers and Parallel Architectures". Ellis Herwood. 1991.

[2] Carling, A. "Parallel Procesing. The Transputer and OCCAM". Sigma Press. 1988.

[3] Chassaing, Rulph. "Digital Signal Processing with C and the TMS320C30". John Wiley & Sons. 1992.

[4] Gallety, John. "OCCAM 2". Pitman. 1990.

[5] INMOS. "Transputer Development System". Prentice Hall. 1990.

[6] Jájá, Joseph. "An Introduction to Parallel Algorithms". Addison-Wesley. 1992.

[7] Kerridge, Jon. "OCCAM Programming: a Practical Approach". Blackwell Scientific Publications. 1987.

[8] Lister, A. M. "Fundamentos de los Sistemas Operativos". Gustavo Gili. 1986.

[9] Papamichalis, Panos E.. "Digital Signal Processing Applications with the TMS320 family". Prentice Hall. Vol. 2. 1990.

[10] Perrot, R. H.. "Parallel Programming". Addison-Wesley. 1987.

[11] Pountain, Dick. May, David. "A tutorial Introduction to OCCAM Programming". INMOS. 1988.

[12] Priemer, Roland. "Introductory Signal Processing". World Scientific. Vol. 6. 1991.

[13] Stein, Richard M. "T800 and Counting". BYTE. Vol. 13. No. 11. p.p. 287-300. November 1988.

CAPITULO III.
ALGORITMOS GENETICOS

En los últimos años, la ciencia de la computación ha tratado de simular el pensamiento del hombre para poder emular un aprendizaje. Asimismo ha tratado de simular su evolución desde un punto de vista natural, tal es el caso de los Algoritmos Genéticos y los programas evolutivos.

En el presente capítulo se presenta la teoría de la evolución, base de los Algoritmos Genéticos, historia de los mismos y la forma como trabajan. De igual manera, se muestra la implementación de los Algoritmos Genéticos Paralelos, antecedentes importantes en el desarrollo del trabajo presentado en esta tesis.

III.1. Definición

Los Algoritmos Genéticos emulan la Teoría de Darwin o Teoría de la Evolución por medio de un algoritmo matemático que puede ser implementado en computadora. Una definición que explica ésto es:

Los Algoritmos Genéticos son procedimientos de adaptación, los cuales, están basados en los mecanismos de la evolución natural. Combinan la supervivencia de los individuos más aptos de una población, con operadores genéticos tomados de la naturaleza [15] para formar un sistema robusto que puede ser adaptado a una gran variedad de problemas.

Un Algoritmo Genético simula el comportamiento dinámico de una población genética por medio de estructuras, que tienen la capacidad de desenvolverse dentro de su ambiente, para poder generar nuevas generaciones que tengan un nivel adaptativo más alto.

La estructura general de los Algoritmos Genéticos viene representada de la siguiente forma:

-El cálculo de las condiciones de cada individuo de la población.
-La selección de individuos para producir descendientes de las siguientes generaciones.
-La reproducción de dichos individuos por medio de operadores genéticos.

En esta sección se presenta un panorama general de las principales características de los Algoritmos Genéticos, dando antes una breve explicación de la teoría de la evolución e historia de este importante algoritmo computacional.

III.2. Teoría de la Evolución.

Aunque la teoría de la Evolución fue desarrollada y expuesta en 1859 en "On the Origen of Species" de Charles Darwin [20], anteriormente se tenia un estudio al respecto. La idea de que distintos tipos de organismos pueden transformarse unos en otros fue expresada por filósofos griegos como Anaximandro y Empédocles [12]. Durante la edad media se tenía la idea de que, de la materia inorgánica podían surgir animales y dicha creencia era compatible con la filosofía cristiana dada por teólogos como Santo Tomás de Aquino. La primera teoría completa de la evolución fue difundida por Lamarck (1809), quien propuso dos ideas: la primera consistía en que los organismos son capaces de cambiar la forma, proporciones, color, etc. como respuesta

a cambios específicos del medio ambiente y la segunda contemplaba una progresión discontinua desde las formas más simples de organismos a otras más complejas [12].

La teoría de la Evolución de Darwin se basa principalmente en cuatro postulados:

-El mundo no es estático, sino que evoluciona; las especies cambian continuamente, se originan unas y se extinguen otras.

-El proceso de la evolución es gradual y continuo y no consiste en cambios discontinuos o cambios súbitos.

-Comunidad de descendencia. Los organismos semejantes estaban emparentados y descendían de un antepasado común.

-Selección natural. La selección es un proceso que consta de dos fases: la primera es la producción de variabilidad y la segunda es la lucha por la existencia.

La evolución biológica consiste en cambios en la constitución genética de los individuos de una población, esta información está codificada en una sustancia química llamada ácido desoxirribonucleico (ADN). Un Gen es un segmento de una de las moléculas extraordinariamente largas de ADN que almacena el material genético del organismo y el conjunto de genes formarán lo que es llamado Cromosoma. El número de cromosomas existentes en el núcleo celular difiere de una especie a otra [2].

Los genes pueden interactuar entre sí para dar características específicas, muchas veces un solo gen afecta varias características. Johannesen (1909) introdujo la diferencia entre dos términos muy importantes: Genotipo y Fenotipo. El Fenotipo es un organismo en el aspecto que podemos observar. El Genotipo es la suma total del

material hereditario de un organismo. El Genotipo permanece constante a lo largo de la vida del ser vivo mientras que el Fenotipo sufre siempre alteraciones.

El problema principal que tiene que sufrir un individuo (sea cual sea su hábitat) es la supervivencia y para ello se basa en tres formas diferentes:

Selección natural. Los individuos básicos y vigorosos tienen una mayor probabilidad de sobrevivir y dejar descendencia que los débiles. Darwin consideraba a la selección natural, supervivencia y reproducción, como el principal agente directo del cambio evolutivo

Cruzamiento sexual. El cruzamiento sexual se basa en el intercambio de material genético de dos individuos de una población para dar lugar a nuevas generaciones de la población. El cruzamiento sexual va ligado directamente con la selección natural, ya que el individuo más vigoroso tiene una mayor probabilidad de sobrevivir que un individuo débil.

Mutación. De vez en cuando existen errores en el proceso de paso de información hereditario, dichos cambios son denominados mutaciones. En forma general, las mutaciones son cambios que se producen en el material hereditario y que no se deben a la recombinación genética ni a la segregación independiente de los cromosomas que es característica del proceso sexual. Pueden ser mutaciones génicas, que afectan a los genes de un cromosoma o cromosómicas que afectan el número de cromosomas o el número de genes que forman un cromosoma.

Dentro de la teoría Evolutiva, hay que tomar en cuenta el azar, principalmente dentro de las mutaciones, no existe una tasa constante de mutaciones dentro de una población, por lo que es necesario tomar en cuenta variables aleatorias que intervienen dentro del proceso evolutivo.

Aunque la Teoría de la Evolución no fue demostrada en su totalidad por Darwin, siguen las investigaciones y se han verificado experimentalmente varias de sus afirmaciones. Actualmente se afirma que el hombre, cuenta con 22 parejas de cromosomas, 22 cromosomas son aportados por el espermatozoide y 22 son aportados por el óvulo, dentro de la fase reproductora, y que la mutación es considerada, por ejemplo, en personas con síndrome de Down en donde se tienen tres en lugar de dos cromosomas 21 y la selección natural forma un importante papel en la lucha de la supervivencia.

III.3. Historia de los Algoritmos Genéticos.

Como se sabe, desde hace muchos años, la ciencia ha estudiado el comportamiento del hombre, para así, intentar emularlo, es así como surge la idea de los robots, "seres" encargados de realizar el trabajo que no es grato o bien peligroso realizar por un ser humano. Pero, ¿Por qué no intentar simular su forma de existir?.

El primer indicio sobre la programación evolutiva data de la década de los cincuentas. A principios de 1960, Hans J. Bremermann de la Universidad de California, realiza un tipo de apareamiento determinado por la suma de genes de dos padres empezando así con más fuerza el estudio de este tema. En 1967, Bagley hizo un programa para jugar Hexapawn, que se juega en un tablero de ajedrez de tres por tres localidades. Cada oponente comienza con tres peones y trata de llegar al otro lado. Bagley construyó Algoritmos Evolutivos para buscar conjuntos de parámetros en funciones de evaluación del juego y las comparó con algoritmos de correlación. Encontró que los algoritmos de correlación requerían una gran correspondencia entre la no linealidad del juego y la no linealidad del algoritmo. Sin embargo, los algoritmos que él empleó no eran afectados por la no linealidad del juego. Pero no fue hasta la década de los setentas, cuando John Holland [4] [16] y sus colegas de la Universidad de Michigan se plantearon esto con dos objetivos [15]:

-Abstraer y explicar rigurosamente los problemas adaptativos de los sistemas naturales y,

-Diseñar sistemas artificiales de Software que utilizaran los mecanismos importantes de los sistemas naturales.

Cuando Holland empezó a estudiar esto, no tenía nombre para este campo de investigación, sin embargo, fue necesario bautizarlo. En referencia a su origen en el estudio de la genética, Holland decidió llamarlo Algoritmos Genéticos [9]. Con lo que comenzaron a desarrollar toda una teoría matemática que explicara el funcionamiento de los Algoritmos Genéticos y su forma de implementarlos a un problema dado.

III.4. Terminología.

Para poder entender los principios en los que se basan los Algoritmos Genéticos, es importante conocer la relación de terminologías entre lo natural y el algoritmo en sí. Esto es observado en la tabla 3.1 [15].

NATURAL	ALGORITMOS GENETICOS
Cromosoma	Individuo (cadena de bits)
Gen	Caracter (bit)
Genotipo	Estructura
Fenotipo	Conjunto de Parámetros
Epístasis	No linealidad

Tabla 3.1. Terminología.

III.5. Diferencias con otros métodos.

La flexibilidad que ofrecen los Algoritmos Genéticos para resolver problemas es muy grande ya que la estructura general se conserva siempre igual. Algunas diferencias entre los Algoritmos Genéticos con otros métodos se pueden resumir de la siguiente manera [15]:

1. Los Algoritmos Genéticos trabajan con un código que representa a los parámetros a buscar, no con los parámetros en sí, es decir, se trabaja con una representación binaria que se puede sintetizar sea cual sea el problema en cuestión.

2. Los Algoritmos Genéticos trabajan sobre una población de posibles soluciones, no con una única posible solución, es decir, es posible evaluar varios espacios solución al mismo tiempo.

3. Los Algoritmos Genéticos se basan en una "función objetivo", que no se deriva de otros conocimientos auxiliares, es así como el esquema, no cambia sea cual sea el problema que se tiene.

4. Los Algoritmos Genéticos usan reglas de transición probabilísticas, no determinísticas.

Muchos métodos requieren información adicional para poder ser utilizados, como es el caso de los métodos de gradientes en donde necesitamos derivadas para llegar a obtener los valores óptimos. En los Algoritmos Genéticos, no es necesario conocer información adicional, trabajan como una caja negra en donde únicamente se necesita conocer la función objetivo que regirá al algoritmo.

Otros métodos necesitan además, conocer el funcionamiento del método que se esté utilizando de acuerdo al problema que se esté resolviendo, así, es necesario tener

conocimientos de cinemática si se trata de resolver un problema de movimiento de un cuerpo o conocer termodinámica para un problema de temperatura. En los Algoritmos Genéticos, el esquema general no cambia, con lo cual, es independiente del problema que se esté tratando sin necesidad de profundizar en temas secundarios.

Muchos problemas de optimización trabajan punto a punto, empleando resultados anteriores para nuevos puntos a utilizar. En los Algoritmos Genéticos se estudia un espacio solución, formado por varios puntos que no siempre se encuentran en una vecindad, ésto nos lleva a no estar analizando puntos que aunque sean máximos o mínimos (según el problema) sean los críticos.

III.6. Algoritmo Genético básico

En general, un Algoritmo Genético estará formado por un esquema de reproducción y un conjunto de operadores genéticos como lo muestra la figura 3.1. por medio de un diagrama de bloques.

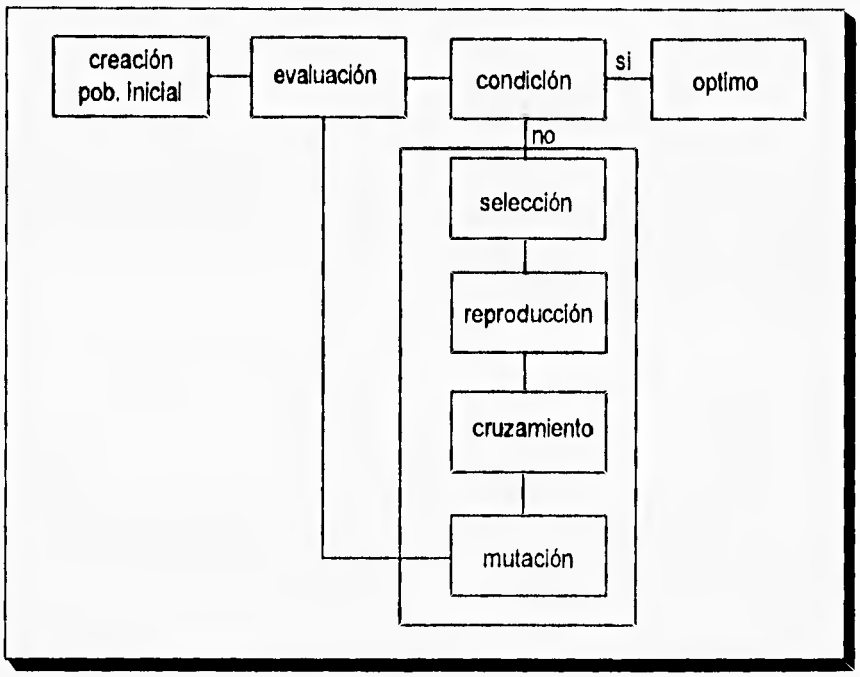


Figura 3.1. Algoritmo Genético Básico.

Al comienzo del algoritmo es necesario crear la población inicial de un tamaño específico, el cual nunca variará. Se realiza una Evaluación de dicha población inicial para posteriormente Seleccionar los individuos más aptos para la Reproducción y Cruzamiento, se realiza una Mutación y nuevamente la evaluación de la nueva generación creada. El proceso se repite hasta cumplir una *Condición* que puede ser un número específico de generaciones o la solución óptima.

III.7. Esquemas de reproducción.

En esta sección se presentan algunos esquemas de reproducción, constituidos principalmente por la Selección y la Reproducción.

Asimismo es importante definir un término que se utilizará en el estudio de los Algoritmos Genéticos: Fitness y la función de Fitness, también llamada función objetivo.

Dado un cromosoma particular, la Función de Fitness regresará un valor conocido como Fitness, que en términos generales es el valor de "utilidad" o de "habilidad" de un individuo para sobrevivir y ser incluido en siguientes generaciones [4]. Así pues, la función objetivo es lo único que habrá que cambiar cada vez que se desee resolver un problema diferente

III.7.1. Selección.

Su propósito es seleccionar padres que puedan reproducirse o cruzarse en la siguiente generación. Hay muchas formas de realizar esto, aquí se mencionarán sólo las más importantes:

-Método de la Ruleta.

El método de la ruleta consistirá en [9]:

Sumar el valor de Fitness de todos los miembros de la población, llamando a esta suma, el total de fitness, para así, generar un número aleatorio n, entre cero y el total de fitness. De acuerdo al valor aleatorio obtenido, es posible obtener, por medio de una distribución en forma porcentual, el individuo seleccionado.

Este método es llamado de la ruleta porque se puede hacer una semejanza con una ruleta como lo muestra la figura 3.2.

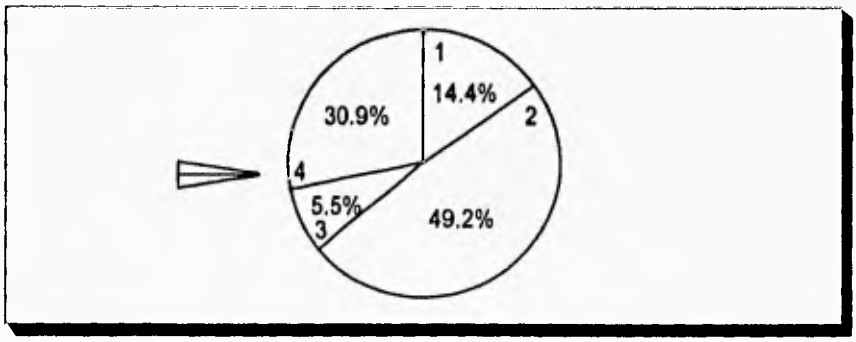


Figura 3.2. Método de la Ruleta.

En donde, el tamaño de cada bloque depende directamente del valor de fitness. La ruleta se hace girar y cuando ésta se detenga, el marcador indicará cual es el individuo escogido. Es de esperarse que, el individuo 2 sea el que más veces se seleccione ya que su valor de fitness es muy alto y, por consiguiente, tiene una mayor probabilidad de ser seleccionado.

-Método del Torneo.

La idea es muy simple, se revuelve la población y después se hace competir a los cromosomas que la integran en grupos de tamaño predefinido (normalmente compiten en parejas), resultarán ganadores aquellos que tengan valores de fitness más altos. Se puede observar que este método garantiza que para formar la siguiente generación, los individuos que tienen un valor de fitness más alto, ganarán más torneos y por consiguiente, se podrán obtener mejores individuos en cada generación [7].

-Método Ranking.

En este método, la selección está muy relacionada con el valor de fitness. Dada una población presente, el individuo que tenga el mayor valor de fitness es el que se va a

poder reproducir o cruzarse con un alto rango, el siguiente valor de fitness se reproducirá o cruzará con un menor rango y así sucesivamente. El método continúa hasta que se alcance el 100% de la población (hay que recordar que el tamaño de la población no varía de generación en generación). Así, el porcentaje de descendientes de un individuo será acorde a su valor de fitness asociado [3].

III.7.2. Reproducción.

La reproducción se basa únicamente en hacer una copia de individuos de una generación a otra sin cruzarse. Haciendo una analogía con la forma natural, son los individuos que no mueren de una generación a otra. Por medio del método del elitismo que consiste en lo siguiente:

Sea $a^*(t)$ el mejor individuo generado en el tiempo t, si después de obtener la generación (t+1) el individuo $a^*(t)$ no aparece, entonces incluir el individuo $a^*(t)$ en la generación (t+1) [17]. es decir, verificar cual es el individuo que contiene el valor de fitness más alto de la generación presente y se hace una copia de este a la generación siguiente.

O bien, una vez que se haya realizado la selección (por cualquiera de los métodos existentes), se realiza una copia de estos individuos para la siguiente generación.

Algunos autores como Lawrence [9] o Beasley [4] se refieren a la etapa de la reproducción como el cruzamiento y la mutación mientras que otros autores como Goldberg [15] hacen una diferencia entre lo que es reproducción, cruzamiento y mutación.

III.8. Operadores Genéticos.

III.8.1. Cruzamiento.

Para poderse llevar a cabo el cruzamiento entre individuos de una población, es necesario realizar una selección utilizando alguno de los métodos descritos anteriormente. Con la selección, se determinarán dos padres que podrán ya sea reproducirse o cruzarse. Los dos padres darán lugar a dos hijos que formarán parte de la siguiente generación [8]. La operación de cruzamiento no es necesariamente desarrollada en todas las cadenas de la población, existe una probabilidad de cruzamiento Px que rige este evento.

El cruzamiento se puede realizar de distintas formas, las más comúnmente utilizadas son: cruzamiento en un punto, en dos puntos y en forma uniforme [9] [10] [11] [15] [23] [24] [25], las cuales, se describen a continuación.

1. Cruzamiento en un punto.

El cruzamiento en un solo punto consiste en dividir cada individuo padre (padre 1 y padre 2) en dos subcadenas desde un punto escogido aleatoriamente entre [1 , l-1], donde l es la longitud del individuo. Así, se formarán dos hijos, uno con la primera subcadena del padre 1 y con la segunda subcadena del padre 2 y el otro con la primera subcadena del padre 2 y la segunda subcadena del padre 1. Por ejemplo, sean dos padres:

P1 = 10010110 y
P2 = 10111000

Si se obtiene un número aleatorio entre 1 y 8 (longitud de cada padre), por ejemplo 5. Entonces se dividen los individuos de la siguiente forma:

P1 = 10010 | 110
P2 = 10111 | 000

Para crear los descendientes:

H1 = 10010000 y
H2 = 10111110

2. Cruzamiento en dos puntos

Para el cruzamiento en dos puntos, se trata a cada uno de los padres como anillos en lugar de como cadenas, es decir, el último bit de cada padre se une con el primero de sí mismo. Los dos puntos de cruce son seleccionados aleatoriamente rompiendo el anillo en dos segmentos. El proceso que se sigue es igual que en el cruzamiento en un punto [11]. Siguiendo el mismo ejemplo anterior:

P1 = 10010110 y
P2 = 10111000

Generando dos números aleatorios, por ejemplo 2 y 5, los individuos padres se dividen:

P1 = 10 | 010 | 110 y
P2 = 10 | 111 | 000

Para generar los descendientes:

H1 = 10111110 y
H2 = 10010000

3. Cruzamiento uniforme.

En el cruzamiento uniforme, cada bit de los descendientes son formados ya sea por el padre 1 o por el padre 2 por medio de una selección aleatoria [9]. Esto se puede explicar por medio de una máscara como lo muestra el siguiente ejemplo:

P1 = 10010110 y
P2 = 10111000

mascara = 11010010

La máscara es seleccionada aleatoriamente para cada cruzamiento, generando, en este ejemplo:

H1 = 10111010 y
H2 = 10010100

Es decir, el hijo 1 estará formado por los bits del padre 1 si la máscara indica 1, de lo contrario, es formado por los bits del padre 2. De la misma forma, el hijo 2 estará formado por los bits del padre 2 si la máscara indica 1, de lo contrario, es formado por los bits del padre 2.

III.8.2. Mutación.

La mutación es equivalente a reemplazar con una cierta probabilidad un gen de un individuo de 0 a 1 o viceversa. La Mutación generalmente se lleva a cabo con una

probabilidad constante durante el transcurso de las generaciones, pero, es posible implementar la mutación con parámetros que son adaptados durante el desarrollo del algoritmo genético. Algunas de las variantes que se le da a la probabilidad de mutación son las siguientes:

Probabilidad constante. Durante el transcurso de las generaciones, la probabilidad de mutación permanece constante. De Jong muestra que esta probabilidad constante puede tomarse como una medida de la longitud de los individuos l, como Pm = 1 / l [18].

Probabilidad constante con alta probabilidad en la primera generación. En este tipo de variante, la probabilidad permanecerá constante en el transcurso de las generaciones, pero, en la primera generación, la probabilidad será mucho mayor, generalmente de 0.5 [14].

Probabilidad exponencial. En la probabilidad de mutación exponencial, como su nombre lo dice, lleva una forma exponencial decreciente constante en cada generación [5].

Probabilidad exponencial sobre la posición de bits. La probabilidad vendrá regida de acuerdo a la posición de los bits. Así, el bit más significativo tendrá menos probabilidad de mutación que el bit menos significativo [13].

III.9. Teorema del Esquema.

Hasta ahora se han descrito los pasos que intervienen en el desarrollo de un Algoritmo Genético, es importante conocer cual es la teoría matemática que sostiene este método.

Holland muestra la teoría en 1975 y la llama el Teorema del Esquema [1] [17]. En ella agrega un símbolo que puede formar a un individuo (*) llamado "no importa" en donde, si se tiene una representación binaria, cuando se encuentra el símbolo * podrá representar 1 ó 0. Por ejemplo, considérese el esquema:

H = *10011*

éste representará a los individuos:

A1= 0100110
A2 = 0100111
A3 = 1100110
A4 = 1100111

En donde, se puede decir que A1, A2, A3 y A4 son ejemplificaciones del esquema H.

Podemos definir a la cardinalidad k como la base en la que se está trabajando, en el caso de la representación binaria se tendrá k=2 y el número de esquemas con que cuenta será $(k + 1)^l$ donde l es la longitud de un individuo.

Los esquemas difieren unos de otros por el número de símbolos * con que se cuenta o la separación entre dígitos binarios. El orden del esquema, denotado por o(H), es el número de símbolos no * (0 ó 1) mostrados, así, en el ejemplo anterior, el orden será o(H) = 5 (porque contiene cinco dígitos binarios). La longitud del esquema, definido como $\delta(H)$ es la distancia entre el primer símbolo no * y el último símbolo no *, en el mismo caso, $\delta(H) = 6 - 2 = 4$. Si el esquema fuera H = 0******, o(H) = 1 y $\delta(H) = 0$.

Dadas estas definiciones, es posible conocer los efectos producidos por la reproducción, cruzamiento y mutación, en un esquema.

Supongamos que en un tiempo t, se tienen m ejemplificaciones del esquema H dentro de la población A(t), definido como m = m(H,t). Durante la reproducción, una cadena es copiada de acuerdo a su valor de fitness con una probabilidad:

$$p_i = \frac{f_i}{\sum f}$$

así, en el tiempo t+1, nosotros esperamos obtener m(H,t+1) representaciones m del esquema H definido por:

$$m(H, t+1) = \frac{m(H, t) \times f(H)}{f_m}$$

donde f(H) es el valor promedio del valor de fitness de todos los elementos m que pertenecen al esquema H y n es el tamaño de la población. Como el valor promedio de fitness de toda la población es:

$$f_m = \frac{\sum f}{n}$$

se puede escribir:

$$m(H, t+1) = \frac{m(H, t) \times f(H)}{f_m} \qquad ...3.1.$$

Que es como afecta la reproducción a un esquema. Así, si el esquema tiene un valor promedio mayor que el de la población, el esquema tendrá un número mayor de copias, si no es así, el número de copias disminuirá. Es importante mencionar que es posible estudiar varios esquemas al mismo tiempo, lo que es llamado paralelismo implícito. El incremento o decremento dependerá pues, del valor de fitness del

esquema, es decir, los mejores esquemas se incrementarán en la población y los peores esquemas desaparecerán de la misma.

Como mencionamos anteriormente, la reproducción implica copiar cadenas de una generación a otra, pero no explora nuevas regiones, lo que es posible con el cruzamiento. El cruzamiento implica el intercambio de información entre dos individuos, un esquema es afectado con mayor probabilidad si la longitud del esquema H es muy grande. Veamos un ejemplo: un individuo A forma parte de dos esquemas:

A = 1 0 11 0 01
H1 = * * * 1 0 * *
H2 = * * 1 * * * 1

cuyas longitudes son respectivamente δ(H1)=1 y δ(H2)=4. Supongamos un cruzamiento en un solo punto y que el cruce es en 3. Es decir:

A = 101 1001
H1 = * * * 10* *
H2 = * *1 * * *1

con lo cual, se observa que el esquema H1 sobrevivirá, lo que no sucederá con el esquema H2 el cual morirá. La probabilidad con que un esquema sea destruido es calculado como p_d = δ(H1) / (l-1) por lo que, la probabilidad con que un esquema sobreviva es ps = 1 - p_d. En forma general, como existe la reproducción, el cruzamiento también tiene una determinada probabilidad de desarrollarse pc, con lo que se hace:

$$p_s \geq 1 - \frac{p_c \times \delta(H)}{l-1}$$

Como existe una independencia entre la reproducción y el cruzamiento, la ec. 3.1 quedará:

$$m(H,t+1) \geq m(H,t) \times \frac{f(H)}{f_m} \times \left[1 - \frac{p_c \times \delta(H)}{l-1}\right] \qquad ...3.2.$$

Así, la sobrevivencia de un esquema dependerá de dos cosas: que el valor de fitness promedio del esquema esté por arriba del promedio f_m y que la longitud del esquema sea pequeña.

Por otro lado, la mutación opera con los genes de un individuo, con probabilidad pm, , es decir, un gen sobrevive con probabilidad (1 - pm) y los genes de un esquema que pueden mutar son o(H), con lo que, la ec. 3.2 quedará finalmente:

$$m(H,t+1) \geq m(H,t) \times \frac{f(H)}{f_m} \times \left[1 - \frac{p_c \times \delta(H)}{l-1} - o(H) \times p_m\right]$$

Donde la probabilidad de mutación estará involucrada por medio de su probabilidad de ocurrencia. Un esquema variará por medio de la mutación dependiendo del valor del orden del mismo.

Esta ecuación es llamada Teorema del esquema o Teorema Fundamental de los Algoritmos Genéticos [15].

III.10. Algoritmos Genéticos Paralelos.

En los últimos años y motivados por el constante desarrollo de las arquitecturas paralelas y la multiprogramación, muchas investigaciones dentro del campo de los Algoritmos Genéticos han sido dirigidos hacia el desarrollo de versiones de Algoritmos

Genéticos Paralelos. Gran parte de estos trabajos mantiene como objetivo el conservar la diversidad de los miembros de la población y la eficiencia al dividir una población genética en subpoblaciones para su paralelización.

Una de las razones principales de esta nueva implementación es el hecho de reducir tiempo de proceso al asignar subpoblaciones a cada procesador del sistema. Una razón más importante, se deriva de la observación de que, después de cierto número de generaciones, la mayoría de los cromosomas en la población presentarán un comportamiento similar. La diversidad genética se perderá, y la recombinación después de ésto puede no ser productiva. Un método para resolver este problema es evolucionar subpoblaciones independientemente.

Como los AG's son métodos aleatorios de búsqueda, las diferentes subpoblaciones estarán explorando diferentes regiones del espacio de soluciones. Si la función objetivo es la misma en las diferentes subpoblaciones, cada una de ellas será muy competitiva y más aun, presentará resultados únicos. Es decir, el uso de Algoritmos Genéticos Paralelos tiene como finalidad no solo ecelerar el proceso de cálculo, sino también generar soluciones de una mayor calidad.

De manera general, los Algoritmos Genéticos Paralelos están constituidos por un conjunto de Algoritmos Genéticos Nodales, cada uno de los cuales está residente en un nodo de un sistema multiprocesador, manteniendo una parte de la población al ser dividida en subpoblaciones, correspondientes al número de procesadores disponibles. El Algoritmos Genético Nodal básico puede escribirse como:

-Creación de la población inicial
-Evaluación
-Mientras se ejecute
    -Comunicación y Selección
    -Reproducción
    -Cruzamiento
    -Mutación
    -Evaluación
    -Substitución
-Fin

Existen varios modelos de paralelismo, de los que sobresalen el modelo de Migración [6], de isla [21] o distribuidos [26], de Difusión [6] y el modelo Farming.

En el modelo de migración, existen varias subpoblaciones, cada una es un Algoritmo Genético Nodal. La migración consiste en enviar un porcentaje de individuos a otra subpoblación y recibir el mismo porcentaje de individuos de otra subpoblación diferente. Es importante tener cuidado en que el número de individuos de una subpoblación no cambie con el paso de las generaciones.

Así, el esquema de los Algoritmos Genéticos Paralelos con el modelo de migración será:

-Creación de la Población Inicial
-Evaluación
-Mientras *condición*
    -Selección
    -Reproducción
    -Cruzamiento
    -Mutación
    -Migración y emigración
    -Evaluación
-Obtención de la solución óptima

Como ejemplo, consideremos una población dividida en 6 subpoblaciones, como se observa en la figura 3.3. Cada Subn corresponderá a una subpoblación. La migración se desarrollará en un sólo sentido, es decir, la Sub1 mandará individuos a la Sub2 y recibirá individuos de la Sub6. Cada Subn se ejecutará en un procesador y la migración se hará cada generación.

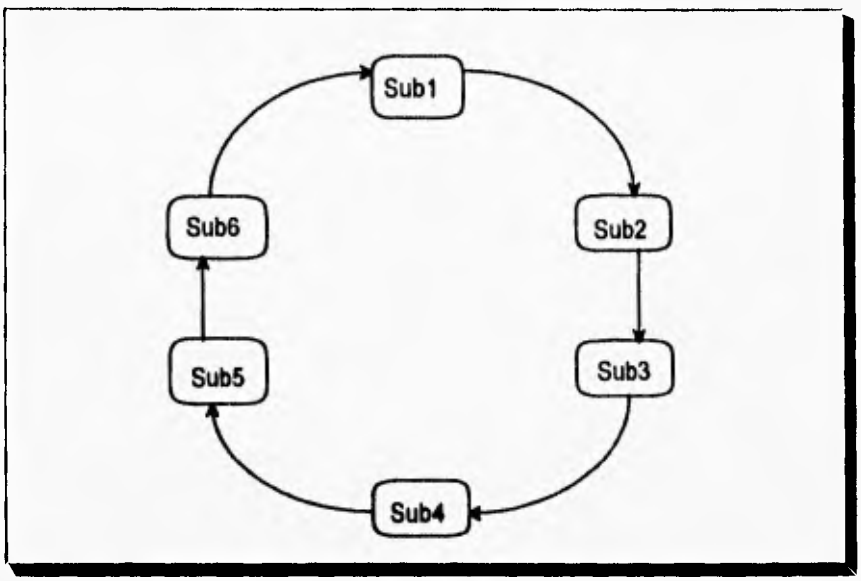


Figura 3.3. Modelo de Migración.

El modelo de Difusión es similar al modelo de Migración, con la diferencia de que las subpoblaciones no están aisladas completamente, sino que se tiene un acceso aleatorio a todos los individuos en algún tiempo determinado, los cuales están colocados dentro de una vecindad. Estos individuos están en continuo movimiento dentro de su vecindad, lo cual permite el acceso para la recombinación. El esquema general viene dado como:

```
-Creación de la población inicial
-Evaluación
-Mientras se ejecute
        PAR
          -Envía enmigrantes a subpobs. vecinas
          -Recibe inmigrantes de subpobs. vecinas.
        -Selección
        -Reproducción
        -Cruzamiento
        -Evaluación
        -Substitución
-Fin
```

La figura 3.4. muestra este modelo.

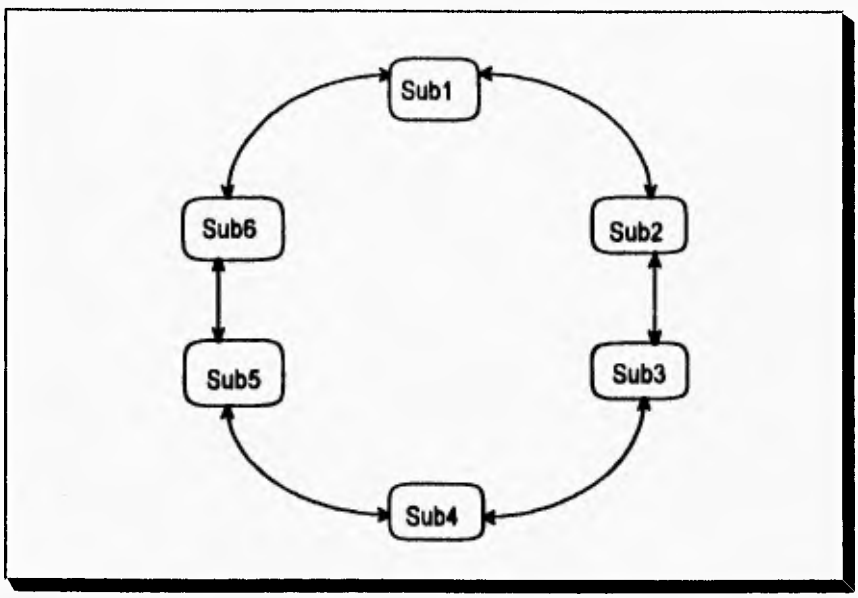


Figura 3.4. Modelo de Difusión.

El Modelo Farming está constituido por un proceso maestro y N procesos esclavos. El procesador maestro supervisará la población total y realizará la selección. Los procesadores esclavos reciben los individuos a ser recombinados para crear los

miembros de la nueva población y los nuevos individuos son evaluados antes de regresarlos al procesador maestro.

El esquema del nodo maestro y de cada nodo esclavo será:

Algoritmo Genético Maestro

```
-Creación de la población inicial
--Evaluación
-Mientras se ejecute
   -Selección
   -Comunicación maestro-esclavo
   -Comunicación esclavo-maestro
   -Substitución
-Fin
```

Algoritmo Genético Esclavo

```
-Mientras se ejecute
   -Comunicación maestro-esclavo
   -Selección
   -Cruzamiento
   -Mutación
   -Evaluación
   -Comunicación esclavo-maestro
-Fin
```

REFERENCIAS.

[1] Antonisse, Jim. "A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint". Proceeding of the third International Conference on Genetic Algorithms. p.p. 86-91. 1989.

[2] Ayala, Francisco J. "Mecanismos de Evolución". Libro de Investigación y Ciencia. Labor. p.p. 15-28. 1982.

[3] Baker, James Edward. "Adaptive Selection Methods for Genetic Algorithms". Proceeding of the first International Conference on Genetic Algorithms. p.p. 101-106. 1985.

[4] Beasley, David Bull. David R. Martin. Ralph R.. "An Overview of Genetic Algorithms: Part 1. Fundamentals". University Computing. p.p. 1-16. 1993.

[5] Bramlette, Mark F.. "Initialization, Mutation and Selection Methods in Genetic Algorithms for Function Optimization". Proceeding of the fourth International Conference on Genetic Algorithms. p.p. 100-107. 1991.

[6] Chipperfield, Andrew. Fleming, Peter. Fonseca, Carlos. "Genetic Algorithms toolbox for Use with Matlab". University of Sheffield. V. 1.2.

[7] Coello, Carlos A.. Introducción a los Algoritmos Genéticos". Soluciones Avanzadas. Vol. 3. No. 17. p.p. 5-11. 1995.

[8] Davis, Lawrance. "Adapting Operator Probabilities in Genetic Algorithms". Proceeding of the third International Conference in Genetic Algorithms. p.p. 61-69. 1989.

[9] Davis, Lawrance. "Handbook of Genetic Algorithms". Van Nostrand Reinhold. 1991.

[10] DeJong, Kenneth. A., Spears. William, M.. "An Analisis of Interacting Roles Population Size and Crossover in Genetic Algorithms". Proceeding for Parallel Problem Solving Form Nature. G. Goos. p.p. 38-47. 1990.

[11] Eshelman, Larry J.. Carvana, Richard A.. Schaffer, J. David.. "Biases in the Crossover Landscape". Proceeding of the third International Conference on Genetic Algorithms. p.p. 10-19. 1989.

[12] Facultad de Ciencias. "Evolución". Libro editado por la Facultad de Ciencias. UNAM. 1982.

[13] Fogarty, Terence C.. "Varying the Probability of Mutation in Genetic Algorithms". Proceeding of the third International Conference on Genetic Algorithms. p.p. 104-109. 1989.

[14] Gallety, John. "OCCAM 2". Pitman. 1990.

[15] Goldberd, David E.. "Genetic Algorithms in Search, Optimization, and Machine Learning". Addison-Wesley. 1989.

[16] Grant, Keith. "An Introduction to Genetic Algorithms". C/C++ Users Journal. Vol. 13. No. 3. p.p. 45-58. March 1995.

[17] Grefenstette, John J.. Baker, James E.. "How Genetic Algorithms Work: A Critical Look at Implicit Parallelism". Proceeding of the third International Conference on Genetic Algorithms. p.p. 20-27. 1989.

[18] Hesser, Jürgen. Männer, Reinhold. "Toward an Optimal Mutation Probability for Genetic Algorithms". Proceeding of the first Workshop. PPSNI. Lecture Notes in Computer Science. p.p. 23-31. 1990.

[19] Janikow, Cezary Z. Michelewicz, Zbigniew. "An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms". Proceeding of the fourth International Conference on Genetic Algorithms. p.p. 31-36. 1991.

[20] Mayr, Ernst. "Evolución". Libro de Investigación y Ciencia. Labor. p.p. 3-12. 1982.

[21] Mühlenbein, Heinz. H., Schomisch. M., J. Born. "The Parallel Genetic Algorithms as Function Optimizer". Proceeding of the fourth International Conference on Genetic Algorithms. p.p. 271-278. 1991.

[22] Mühlenbein, Heinz H. Schlierkamp-Voosen, Dirk. "Predivtive Models for the Breeder Genetic Algorithms I. Continuous Parameter Optimization". Evolutionary Computation. Vol. 1. No. 1. p.p. 25-49. Spring 1993.

[23] Spears, William M.. "Adapting Crossover in Evolutionary Algorithms". Proceeding of the fourth Annual Conference on Evolutionary Programming. 1995.

[24] Spears, William M.. "Crossover on Mutation?". Proceeding of the fourth Annual Conference on Evolutionary Programming. 1995.

[25] Syswerda, Gilbert. "Uniform Crossover in Genetic Algorithms". Proceeding of the third International Conference on Genetic Algorithms. p.p. 2-9. 1989.

[26] Tanase, Reiko. "Distributed Genetic Algorithms". Proceeding of the third International Conference on Genetic Algorithms. p.p. 434-439. 1989.

CAPITULO IV.
ESTIMACION ESPECTRAL PARAMETRICA

El presente capítulo presenta diversas técnicas de Estimación Espectral Paramétrica que incluyen los modelos Autorregresivo (AR), de Movimiento Promedio (MA) y Autorregresivo de Movimiento Promedio (ARMA). Aunque la demanda computacional presentada por dichas técnicas es alta comparada con los métodos tradicionales, el incremento en resolución que presentan es muy útil para el procesamiento de señales médicas, objeto del trabajo presentado en esta tesis.

IV.1. INTRODUCCION.

La Estimación Espectral juega un importante papel dentro del Procesamiento Digital de Señales. Una gran cantidad de señales utilizadas en diversos campos de la ingeniería provienen directamente de la naturaleza y, generalmente, estas señales no presentan una uniformidad o una periodicidad en sus muestras. Dichas señales deben ser estudiadas en profundidad para poder conocerlas.

La Densidad de Potencia Espectral, que puede ser definida como el módulo de la señal en términos de la frecuencia, es muy utilizada en problemas de Procesamiento de Señales, siendo de gran ayuda en problemas como la Flujometría Doppler en señales biomédicas.

Los métodos tradicionales calculan la Densidad de Potencia Espectral (PSD) por medio de la Transformada Rápida de Fourier, es decir, dada una muestra de datos, se calcula la FFT por medio de la ecuación:

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

donde $R(\tau)$ es la función de autocorrelación definida como:

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

y x(t) es la función a quien se le calcula la PSD, siendo ésta estacionaria [1] [3] [6] (una señal estacionaria se define como aquella en que sus características estadísticas no cambian con respecto al tiempo).

O en su forma discreta:

$$S(f) = \sum_{-\infty}^{+\infty} r_{xx}[k]\exp(-j2\pi fk)$$

donde $r_{xx}[k]$ es la función de autocorrelación definida como en la forma continua [8].

Cuando la función x(t) se define como un conjunto de datos de una señal aleatoria en un intervalo de tiempo dado, la Transformada de Fourier presenta limitaciones para el cálculo de la PSD. Existe una suposición para resolver este problema: considerar que los datos fuera de la ventana muestreada tienen un valor de cero o, dicho en otras palabras, la función de autocorrelación es cero para el intervalo fuera de la ventana [7]. Con esto, la Transformada de Fourier puede resolverse, aunque existen otros problemas: la señal aleatoria no es estacionaria. Todo esto lleva consigo errores en el cálculo de la PSD, así, aunque la FFT es eficiente en cuanto a tiempo de procesamiento, la resolución no es muy apropiada cuando se utiliza en el estudio de señales precisas como son las señales biomédicas; en el estudio de señales de voz,

en donde, es necesario reconocer diferentes fonemas o, en el estudio de señales de radar cuando la señal trae consigo ruido que es necesario diferenciar de la señal real.

La Estimación Espectral, nos ayuda así, a *predecir* el comportamiento de la señal fuera de la ventana de datos muestreada para poder eliminar la suposición de que los datos fuera de la ventana valgan cero.

La palabra "Spectrum" fue inicialmente usada por Isaac Newton cuando se refería a una imagen. Hoy en día, la Estimación Espectral es una herramienta de análisis de datos preliminar. Con la Estimación Espectral no es posible contestar preguntas específicas del comportamiento de la señal [4], pero sí, es útil cuando se desea obtener la PSD.

Aunque la Estimación Espectral data de muchos años atrás, no es sino hasta 1967 cuando John Burg publica su trabajo denominado *Maximum Entropy Spectral Analysis* estudiando las ventanas de datos mostradas por Emmanuel Parzen, en que se formaliza este nuevo estudio del procesamiento de señales. El método desarrollado es ahora considerado uno de los más importantes dentro de los métodos llamados paramétricos [7].

Dentro de la Estimación Espectral y de la Estimación de la PSD es posible seguir dos caminos diferentes. Uno es por medio de los modelos No Paramétricos y el otro es por medio de los modelos Paramétricos.

En los modelos Paramétricos el costo computacional es menor comparado con los modelos no Paramétricos, sin perder la estabilidad del sistema, por este motivo, se presentará únicamente el desarrollo matemático de los modelos Paramétricos.

Los modelos que se describen a continuación son el modelo AR (Autorregresive), el modelo MA (Moving Average) y la unión de los dos, el modelo ARMA (Autorregresive Moving Average).

Una vez presentados estos modelos, se describirá el método de covariancia modificada, método utilizado en el desarrollo de esta tesis.

IV.2. Modelos Paramétricos AR, MA y ARMA.

En esta sección se explica con detalle la teoría matemática que soporta los modelos AR, MA y ARMA

Para ésto se incluyen algunas definiciones importantes:

-Señal aleatoria. Una señal aleatoria es aquella que no es predecible, tal es el caso de, por ejemplo, las señales climatológicas.

-Señal Estacionaria. Una señal es estacionaria cuando sus características estadísticas no cambian con respecto al tiempo [5]. Es decir, si tenemos un número de muestras de una señal en un tiempo i y otro número de muestras de la misma señal en otro tiempo definido como i+τ y se calculan las Funciones de Densidad de Probabilidad (PDF) de cada una de las muestras, tenemos:

$$p(x_{i1}, x_{i2}, x_{i3}, \ldots, x_{in}) = p(x_{i1+\tau}, x_{i2+\tau}, x_{i3+\tau}, \ldots, x_{in+\tau})$$

en caso contrario, la señal es no estacionaria.

-Energía de una señal. La energía de una señal se define como:

$$\xi = \int_{-\infty}^{+\infty} |x(t)|^2 dt$$

en donde, una señal tiene energía finita cuando ξ está definida, en caso contrario, es de energía infinita.

-Función de Densidad de Probabilidad (PDF). Si existe una función f(x) tal que:

1. $f(x) \geq 0, -\infty < x < +\infty,$
2. $\int_{-\infty}^{+\infty} f(x)dx = 1,$
3. $P(a \leq X \leq b) = \int_{a}^{b} f(x)dx$

para cualesquiera a y b, entonces f(x) es la Función de Densidad de Probabilidad (PDF) de la variable aleatoria X.

Si la variable aleatoria se encuentra normalmente distribuida, su función de densidad de probabilidad está dada por:

$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{1}{2}\left(\frac{X-\mu}{\sigma}\right)^2\right) \qquad \text{para } -\infty < x < +\infty$$

con μ y σ^2 como la media y la variancia de la variable aleatoria [2].

-Ruido Blanco. El ruido blanco puede ser modelado como una función con distribución normal. Sus propiedades más importantes son [10]:

Función de Densidad de Probabilidad:

$$f(u) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{u^2}{2\sigma^2}\right)$$

Función de Autocorrelación (definida más abajo):

$$\tau_{xx}(u) = \sigma^2 \qquad \text{para n=0, en los demás casos vale cero.}$$

-Densidad de Energía Espectral. La Densidad de Energía Espectral es desarrollada partiendo de la ecuación:

$$X(F) = \int_{-\infty}^{+\infty} x(t)\exp(-j2\pi Ft)dt$$

en donde F está definida en hertz (Hz). Del teorema de Parseval, se tiene que:

$$\xi = \int_{-\infty}^{+\infty} |x(t)|^2 dt = \int_{-\infty}^{+\infty} |X(F)|^2 dF$$

en donde $|X(F)|^2$ representa la distribución de la energía de la señal en función de la frecuencia y es llamada la Densidad de Energía Espectral y se define como:

$$S_{xx}(F) = |X(F)|^2$$

con $S_{xx}(F)$ vista como la Transformada de Fourier de otra función llamada Función de Autocorrelación, definida como:

$$r_{xx}(\tau) = \int_{-\infty}^{+\infty} x^*(t)x(t+\tau)df$$

En el Modelo Autorregresivo, también llamado modelo de Máxima Entropía, no se hace la consideración de que la autocorrelación fuera de la ventana de datos vale cero, y como consecuencia, este nuevo análisis es más preciso que los métodos convencionales aunque el tiempo de procesamiento se incrementa.

Por otro lado, Burg utiliza el concepto de máxima entropía basándose en el cambio del espectro [7]. Así, parte de:

$$S(w) = \sum_{k=-\infty}^{+\infty} r_{xx}[k]\exp(-jwk) \qquad ...4.1$$

que representa a la Densidad de Potencia Espectral de una señal. Si nosotros quisiéramos obtener la señal r_{xx} se puede obtener por medio de la Transformada Inversa de Fourier:

$$r_{xx} = \frac{1}{2\pi}\int_{-\pi}^{+\pi} S(w)\exp(jwn)dw \qquad ...4.2$$

definido para w y para todo valor de n, con n=0, 1, 2, ..., p. Como se necesita que la entropía sea máxima, S(w) debe de ser máxima o bien, r_{xx} sea máxima. El trabajo de Shannon [7] indica que la entropía es proporcional a la integral del logaritmo de la Densidad de Potencia Espectral, o bien:

$$\int_{-\pi}^{\pi} \log S(w)dw \qquad ...4.3$$

Por lo cual, se requiere maximizar la Entropía de la Densidad de Potencia. Para $|n| \le p$ se tiene:

$$\frac{\partial S(w)}{\partial r_{xx}(n)} = \exp(-jwn)$$

que implica:

$$\frac{\partial \log S(w)}{\partial r_{xx}(n)} = \frac{\exp(-jwn)}{S(w)} = [S(w)]^{-1}\exp(-jwn) \qquad ...4.4$$

Para el intervalo $|n| > p$ la ecuación 4.3 puede ser derivada con respecto a $r_{xx}(n)$ e igualar a cero:

$$\frac{\partial}{\partial r_{xx}(n)} \int_{-\pi}^{\pi} \log S(w) dw = \int_{-\pi}^{\pi} \frac{\partial \log S(w)}{\partial r_{xx}(n)} dw = 0 \qquad \text{para } |n| > p \qquad ...4.5$$

y sustituyendo la ecuación 4.4 en 4.5:

$$\int_{-\pi}^{\pi} [S(w)]^{-1} \exp(-jwn) dw = 0 \qquad \text{para } |n| > p \qquad ...4.6$$

Se puede relacionar la Densidad de Potencia Inversa por medio de otra función de autocorrelación definida como $\psi(n)$ para tener:

$$[S(w)]^{-1} = \sum_{n=-\infty}^{\infty} \psi(n) \exp(-jwn) \qquad \text{para } -\pi \le w \le \pi \qquad ...4.7$$

y

$$\psi(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} [S(w)]^{-1} \exp(jwn) dw \quad \text{para toda n} \qquad ...4.8$$

de la ecuación 4.6, sustituyendo n por -n y multiplicando por $1/2\pi$ se tiene:

$$\frac{1}{2\pi} \int_{-\pi}^{\pi} [S(w)]^{-1} \exp(jwn) dw = 0 \qquad \text{para } |n| > p \qquad ...4.9$$

comparando 4.9 con 4.8 se puede ver que la máxima entropía se tiene cuando $\psi = 0$ para $|n| > p$. Así, en la ecuación 4.7 se tiene, cambiando límites superior e inferior:

$$[S(w)]^{-1} = \sum_{n=-p}^{p} \psi(n) \exp(-jwn) \qquad ...4.10$$

con lo cual, la Densidad de Potencia Espectral será:

$$S(w) = \frac{1}{\sum_{n=-p}^{p} \psi(n)\exp(-jwn)} \qquad ...4.11$$

sustituyendo $z=e^{jw}$ y desarrollando $\psi(n)$ conforme al método Fejér se tiene

$$\sum_{n=-p}^{p} \psi(n)\exp(-jwn) = \frac{1}{\sigma^2}[1 + a_1 z^{-1} ... + a_p z^{-p}][1 + a_1 z^{1} + ... + a_p z^{p}]$$

en donde σ^2 se define como la variancia de ruido blanco como señal de entrada (su origen se explicará más adelante). Si además se hace la consideración de:

$$A(z) = 1 + a_1 z^{-1} + ... + a_p z^{-p} \qquad ...4.12$$

tenemos finalmente:

$$S(w) = \frac{\sigma^2}{A(z)A(z^{-1})} = \frac{\sigma^2}{|A(z)|^2} \qquad ...4.13$$

o bien:

$$S(w) = \frac{\sigma^2}{\left|1 + \sum_{n=1}^{p} a[n]\exp(-jwn)\right|^2} \qquad ...4.14$$

el cual se conoce como el modelo Autorregresivo [7].

La ecuación 4.14 puede ser escrita también como:

$$S(w) = \sigma^2|H(z)|^2 \qquad ...4.15$$

donde nuevamente, σ^2 es la variancia de ruido blanco correspondiente a la secuencia de ruido blanco de entrada y $H(z)$ es la función de transferencia del sistema [12].

Así, el filtro del Modelo Autorregresivo se puede expresar como lo muestra la figura 4.1 [4].

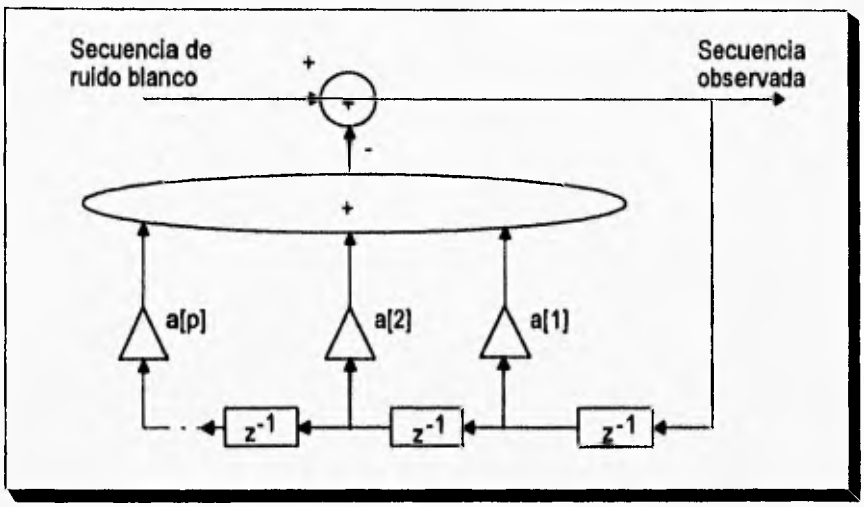


Figura 4.1. Filtro del modelo Autorregresivo.

De la ecuación 4.13 y 4.14, se puede observar que el sistema es estable si todos sus polos están dentro del círculo unitario en representación z o son negativos en el dominio de la frecuencia. De igual forma, se observa que todos los ceros de la función de transferencia valen cero. Si ahora, los ceros tienen un valor distinto de cero y los polos valen cero con a[0]=1, se tiene:

$$S(w) = \sigma^2 |B(z)|^2 \qquad ...4.16$$

Con B(z), de la misma forma que A(z):

$$B(z) = \sum_{k=0}^{q} b[z] z^{-k} \qquad ...4.17$$

quedando la ecuación 4.16 como:

$$S(w) = \sigma^2 \left| \sum_{k=0}^{q} b[z] z^{-k} \right|^2$$

conocido como modelo MA (Moving Average) [12].

La figura 4.2 muestra la forma como se puede expresar el filtro de Movimiento Promedio.

Es importante notar la diferencia entre el modelo de Autocorrelación y el de Movimiento Promedio. En el modelo de Autocorrelación, la salida depende de los valores a[n] y de la salida anterior como se observa en la figura 4.1 (la retroalimentación explica este detalle). El modelo de Movimiento Promedio no depende de la salida en un tiempo anterior, únicamente depende de las variable b[n] y de las entradas como se puede observar en la figura 4.2 [4].

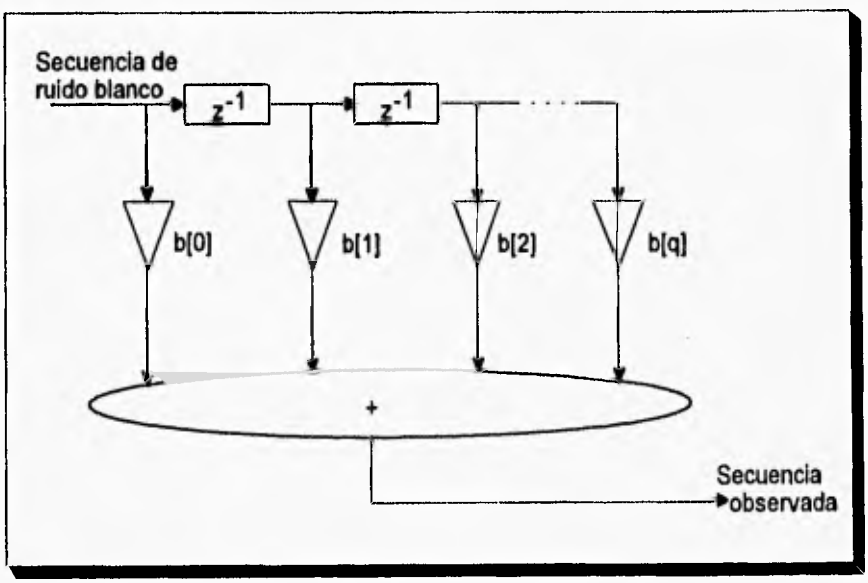


Figura 4.2. Filtro del modelo de Movimiento Promedio.

Más aun, es posible unir las dos representaciones de Autocorrelación y de Movimiento Promedio para dar lugar al modelo de Autocorrelación de Movimiento Promedio (ARMA). La función de transferencia será el producto del modelo AR con el modelo MA.

$$S(w) = \sigma^2 \frac{|B(z)|^2}{|A(z)|^2}$$

con A(z) y B(z) definidos como las ecuaciones 4.12 y 4.17 respectivamente.

El filtro ARMA se expresa como la unión del filtro AR y el filtro MA como lo muestra la figura 4.3 [4].

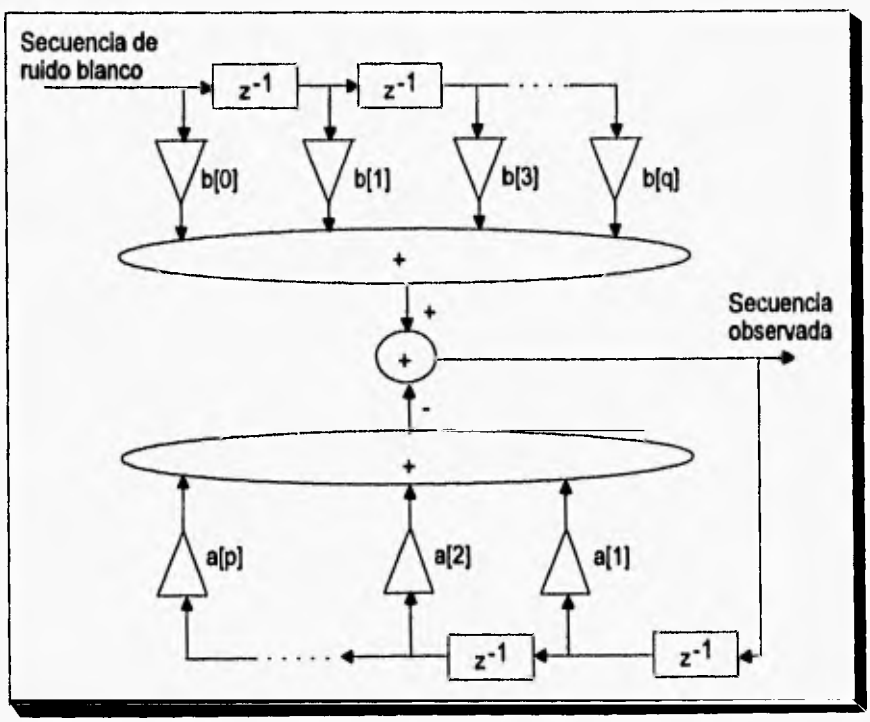


Figura 4.3. Filtro del modelo de Autocorrelación de Movimiento Promedio.

Todos estos modelos son conocidos como modelos paramétricos, siendo el más utilizado el modelo de Autocorrelación (AR) ya que es posible encontrar los valores del filtro como solución de un conjunto de ecuaciones lineales, mientras que, para obtener los parámetros de los modelos MA y ARMA es necesario resolver un conjunto de ecuaciones no lineales.

Para poder obtener los parámetros a[i] del modelos AR, nos podemos basar en el método de covariancia y, más aun, en el método de covariancia modificada.

IV.3. Método de Covariancia Modificada.

Como se describió anteriormente, es posible estimar la Densidad de Potencia Espectral por medio del modelo AR. Uno de los métodos utilizados para encontrar los parámetros a[i] del modelo AR es el método de covariancia modificada, ampliación del método de covariancia. En esta sección se describe dicho método con el objeto de poder utilizar el modelo AR como una herramienta completa de estimación.

Es importante notar en la figura 4.1 que la entrada del sistema es una secuencia de ruido blanco, si definimos a la secuencia de ruido blanco como u[x] y la secuencia de salida como x[n], es posible definir al filtro como [4]:

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

con p como número de parámetros a estimar, o bien:

$$u[n] = x[n] + \sum_{k=1}^{p} a[k]x[n-k] \qquad \ldots 4.18$$

Al desarrollar la ecuación 4.18 se tiene que:

$$u[p] = x[p]+a[1]x[p-1]+...+a[p]x[0]$$
$$u[p+1]= x[p+1]+a[1]x[p]+...a[p]x[1]$$
$$\vdots$$
$$\vdots$$
$$u[N-1]=x[N-1]+a[1]x[N-2]+...a[p]x[N-p-1]$$

con N número de datos por ventana; así, la PDF de $u=[\, u[p] \;\; u[p+1] \;\; \ldots \;\; u[N-1]\,]^T$ es:

$$p(u) = \prod_{n-p}^{N-1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{u^2[n]}{2\sigma^2})$$

o bien:

$$p(u) = \frac{1}{(2\pi\sigma^2)^{(N-p)/2}} \exp(-\frac{1}{2\sigma^2} \sum_{n-p}^{N-1} u^2[n]) \qquad ...4.19$$

sustituyendo la ecuación 4.18 en la ecuación 4.19 se tiene:

$$p(x, a, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{(N-p)/2}} \exp\left[-\frac{1}{2\sigma^2} \sum_{n-p}^{N-1}(x[n] + \sum_{j-1}^{p} a[j]x[n-j])^2\right] \qquad ...4.20$$

que se maximiza sobre *a* minimizando:

$$S_1(a) = \sum_{n-p}^{N-1} \left(x[n] + \sum_{j-1}^{p} a[j]x[n-j]\right)^2 \qquad ...4.21$$

Desarrollando la diferencial sobre *a*, se obtiene:

$$\sum_{j=1}^{p} \hat{a}[j] \sum_{n=p}^{N-1} x[n-j]x[n-k] = -\sum_{n=p}^{N-1} x[n]x[n-k] \quad k=1,2,\ldots,p \qquad \ldots 4.22$$

en forma de matriz:

$$\begin{bmatrix} c_{xx}[1,1] & c_{xx}[1,2] & \ldots & c_{xx}[1,p] \\ c_{xx}[2,1] & c_{xx}[2,2] & \ldots & c_{xx}[2,p] \\ \vdots & \vdots & \vdots & \vdots \\ c_{xx}[p,1] & c_{xx}[p,2] & \ldots & c_{xx}[p,p] \end{bmatrix} \begin{bmatrix} \hat{a}[1] \\ \hat{a}[2] \\ \vdots \\ \hat{a}[p] \end{bmatrix} = - \begin{bmatrix} c_{xx}[1,0] \\ c_{xx}[2,0] \\ \vdots \\ c_{xx}[p,0] \end{bmatrix} \qquad \ldots 4.23$$

donde:

$$c_{xx}[j,k] = \frac{1}{N-p} \sum_{n=p}^{N-1} x[n-j]x[n-k] \qquad \ldots 4.24$$

Dividiendo entre el factor [1/(N-p)] como estimador de autocorrelación.

Para encontrar el valor de σ^2 se diferencia el logaritmo de p (PDF) con respecto a la varianza, es decir:

$$\frac{\partial \ln p}{\partial \sigma^2} = 0$$

de donde se obtiene:

$$\sigma^2 = \frac{1}{N-p} S_1(a)$$

sustituyendo la ecuación 4.21 y basándonos en el resultado de la ecuación 4.22 se tendrá:

$$\sigma^2 = \frac{1}{N-p} \sum_{n=p}^{N-1} x^2[n] + \sum_{j=1}^{p} \hat{a}[j] \frac{1}{N-p} \sum_{n=p}^{N-1} x[n]x[n-j]$$

y finalmente:

$$\sigma^2 = c_{xx}[0,0] + \sum_{j=1}^{p} \hat{a}[j] c_{xx}[0,j] \qquad ...4.25$$

El Método de Covariancia se basa, pues, en minimizar el error de la estimación de la predicción de la potencia dada por [4]:

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

en donde, para encontrar los valores a[i] y c[i,j] nos apoyamos en las ecuaciones 4.23 y 4.24 por ser de la misma forma.

Con lo cual, la Densidad de Potencia Espectral se obtendrá por medio de la ecuación 4.14.

En el método de Covariancia se efectúa una predicción únicamente en adelanto. Una forma más general se logra efectuando la predicción tanto en adelanto como en atraso. Esto se logra por medio del Método de Covariancia Modificada [9] [11] en donde el error a minimizar está dado como:

$$\hat{p} = \frac{1}{2}(\hat{p}^f + \hat{p}^b)$$

donde $\hat{p}^f$ y $\hat{p}^b$ son los errores predictivos en atraso y en adelanto respectivamente y están dados por:

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

y

$$\hat{p}^b = \frac{1}{N-p}\sum_{n=0}^{N-1-p}\left|x[n]+\sum_{k=1}^{p}a^*[k]x[n+k]\right|^2$$

Así, para obtener los valores de a[i] tenemos:

$$\frac{\partial \hat{p}}{\partial a[i]} = 0$$

obteniendo:

$$\sum_{k=1}^{p}\hat{a}[k]\left(\sum_{n=p}^{N-1}x[n-k]x^*[n-i]+\sum_{n=0}^{N-1-p}x^*[n+k]x[n+i]\right) =$$

$$= -\left(\sum_{n=p}^{N-1}x[n]x^*[n-1]+\sum_{n=0}^{N-1-p}x^*[n]x[n+i]\right)$$

para i=1,2,...,p se tiene:

$$c_{xx}[j,k] = \frac{1}{2(N-p)}\left(\sum_{n=p}^{N-1}x^*[n-j]x[n-k]+\sum_{n=0}^{N-1-p}x[n+j]x^*[n+k]\right) \qquad ...4.27$$

escrito en forma matricial:

$$\begin{vmatrix} c_{xx}[1,1] & c_{xx}[1,2] & \dots & c_{xx}[1,p] \\ c_{xx}[2,1] & c_{xx}[2,2] & \dots & c_{xx}[2,p] \\ \vdots & \vdots & \vdots & \vdots \\ c_{xx}[p,1] & c_{xx}[p,2] & \vdots & c_{xx}[p,p] \end{vmatrix} \begin{vmatrix} \hat{a}[1] \\ \hat{a}[2] \\ \vdots \\ \hat{a}[p] \end{vmatrix} = -\begin{vmatrix} c_{xx}[1,0] \\ c_{xx}[2,0] \\ \vdots \\ c_{xx}[p,0] \end{vmatrix} \qquad ...4.28$$

y de la misma forma, la variancia del ruido blanco se obtendrá como:

$$\sigma^2 = c_{xx}[0,0]+\sum_{k=1}^{p}\hat{a}[k]c_{xx}[0,k] \qquad ...4.29$$

Una vez calculados los parámetros a[i] y la variancia de ruido blanco, es posible obtener la Densidad de Potencia Espectral por medio de la ecuación del modelo Autorregresivo:

$$S(w) = \frac{\sigma^2}{\left|1 + \sum_{n=1}^{p} a[n]\exp(-jwn)\right|^2}$$

Es decir, una vez obtenidos los parámetros c_{α} ya sea por el Método de Covariancia (ecuación 4.24) o por el Método de Covariancia Modificada (ecuación 4.27), se resuelve el sistema de ecuaciones (ecuación 4.23 ó 4.28) para determinar los valores estimados a[i]. Para calcular A(n) es posible basarse en la Transformada Rápida de Fourier. Una vez obtenido el módulo cuadrado de A(n), se calcula la variancia de ruido blanco por medio de la ecuación 4.25 ó 4.29 para así dar lugar a la PSD estimada por medio de la ecuación 4.13 ó 4.14.

Como se puede observar, el número de operaciones se incrementa de acuerdo a los modelos tradicionales, pero, la definición que guardan estos modelos paramétricos con respecto a los anteriores permite utilizarlos para poder obtener mejores resultados en problemas donde se necesita una buena precisión como es el caso de problemas biomédicos. En este tipo de problemas, los modelos tradicionales no dan buenos resultados por la suposición de hacer ceros los valores fuera de la ventana perdiendo así, definición en la respuesta.

Así pues, el método que se utilizará para el cálculo de los parámetros a[i] del modelo Autorregresivo será el método de Covariancia Modificada, que presenta una mayor exactitud respecto al método de covariancia. Como se observa, la demanda computacional es muy grande al utilizar éste método, si se requiere procesamiento en tiempo real, es prácticamente imposible con un procesamiento secuencial, es por eso que se recurre al Procesamiento Paralelo y a algunas herramientas que nos permitan resolver el problema como es el caso de los Algoritmos Genéticos Paralelos.

Referencias.

[1] Barkat, Mourad. "Signal: Detection & Estimation". Artech House. 1991.

[2] Canavos, George C.. "Probabilidad y Estadística, Aplicaciones y Métodos". Mc. Graw-Hill. 1992.

[3] Dietmar, Gunter. "Spectral Estimation". Signal Processing II: Theories and Application". p.p. 447-454. 1983.

[4] Kay, Steven M.. "Modern Spectral Estimation. Theory & Application". Prentice Hall. 1988.

Lynn, P. A. "An Introduction to the Analysis: A Modern Perspective". Proceeding of the IEEE. Vol. 69. 1981.

[5] Peebles, Peyton Z.. "Probability, Random Variables, and Random Signal Principles". Mc. Graw-Hill. 1980.

[6] Proakis, John G.. Rader, Charles, M.. Ling, Fuyun. Nikias, Chrysostomos L.. "Advanced Digital Signal Processing". Macmillan Publishing Company. 1992.

[7] Robinson, Enders A.. "A Historical Perspective of Spectral Estimation". Proceeding of the IEEE. Vol. 70. No. 9. p.p. 885-907. 1982.

[8] Ruano, M. Graca. "Investigation of Real-Time Spectral Analysis Techniques for Use with Pulsed Ultrasonic Doppler Blood-Flow Detectors". University College of Wales. February 1992.

[9] Ruano, M. Graca. García, D. F.. Fish, P. J.. Fleming, P. J.. "Alternative Parallel Implementations of an AR-Modified Covariance Spectral Estimator for Diagnostic Ultrasonic Blood Flow Studies". Parallel Computing. p.p. 463-476. 1993.

[10] Shanmugan, K. Sam. Breipohl, A. M.. "Random Signals. Detection, Estimation and Data Analysis". John Wiley & Sons. 1988.

[11] Solano, J.. García, D. F.. "Implementation of Parametric Spectral Estimator Using Genetic Algorithms". Control '94. No. 389. p.p. 754-759. March 1994.

[12] Therrien, Charles W.. "Discrete Random Signals and Statistical Signal Proceesing". Prentice Hall. 1992.

CAPITULO V.
ESTIMACION ESPECTRAL EN FLUJOMETRIA DOPPLER UTILIZANDO ALGORITMOS GENETICOS

En el presente capítulo se muestra la importancia del uso de técnicas de estimación espectral en Flujometría Doppler, así como su implementación mediante nuevas alternativas como lo son los Algoritmos Genéticos.

V.1. INTRODUCCION.

En la actualidad, las enfermedades cardiovasculares son la causa de un elevado número de decesos en la población víctimas de este tipo de padecimientos. Dispositivos como el Detector de Flujo Sanguíneo por efecto Doppler son comúnmente utilizados con la finalidad de detectar este tipo de problemas en una etapa temprana.

Este instrumento determina la velocidad del flujo sangulneo por medio del desplazamiento Doppler en frecuencia de la señal ultrasónica, dispersado por el flujo sanguíneo. El incremento en el rango de frecuencias Doppler puede ser el resultado de algún tipo de turbulencias, causado por una lesión estenótica.

La gran mayoría de los sistemas de Ultrasonido Doppler comercialmente disponibles, hacen uso de la Transformada Rápida de Fourier para calcular el espectro Doppler de la señal, generada al monitorear el flujo sanguíneo, extrayendo de este modo, importante información para su diagnóstico. De esta manera, se pueden detectar

padecimientos moderados y severos. Sin embargo, con esta técnica es muy difícil distinguir problemas de estenosis en su estado inicial, debido a las limitantes de la FFT asociadas a su resolución y segmentación.

Diversas investigaciones señalan que los métodos paramétricos de Estimación Espectral ofrecen un importante avance en cuanto a resolución de respuesta en frecuencia, e identifican al Método de Covariancia Modificada, como el más conveniente en cuanto al cálculo de los parámetros del modelo, presentando una menor complejidad y eficiencia computacional [1] [3] [4].

Este capítulo describe una alternativa al utilizar Algoritmos Genéticos para obtener la respuesta espectral de la señal Doppler en estudio y antecede a la implementación paralela presentada en capítulos subsecuentes.

El objetivo es reducir la complejidad computacional del estimador paramétrico utilizando la simplicidad inherente de los Algoritmos Genéticos, y así, poder incrementar el orden del modelo, además de abrir la posibilidad del uso de métodos más complejos.

V.2. Análisis Espectral de Señales Doppler.

En el contexto de esta aplicación, el análisis espectral es utilizado para detectar la presencia de lesiones estenóticas, al indicar el incremento de la velocidad en el flujo sanguíneo al pasar por la estenosis.

La figura 5.1. muestra una lesión estenótica, en la cual, se modifica la distribución de la velocidad y de la presión del flujo sanguíneo.

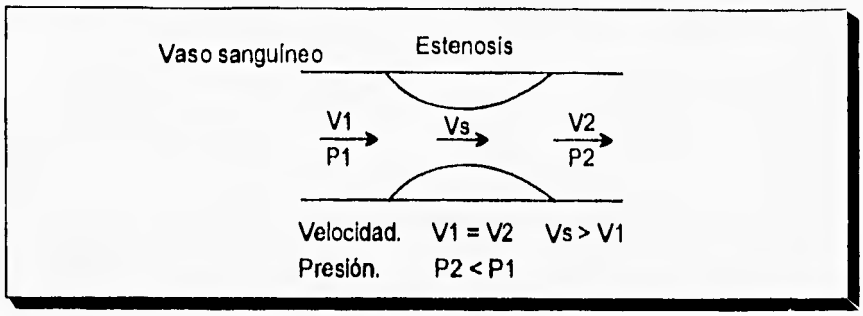


Figura 5.1. Lesión Estenótica.

Al agudizarse una lesión estenótica (decremento en el diámetro de los vasos sanguíneos), se genera una elevación progresiva de presión, un decremento en el flujo sanguíneo y un incremento en la velocidad. Es importante mencionar que es necesario tener un alto grado de la lesión antes que ésta se refleje en una caída significativa en el rango del flujo sanguíneo.

La velocidad en la estenosis se mantiene constante hasta el momento en que se presenta una lesión aguda. Debido a la naturaleza pulsátil del flujo en las arterias, una lesión estenótica afecta la forma de onda de la velocidad de flujo como se aprecia en la figura 5.2.

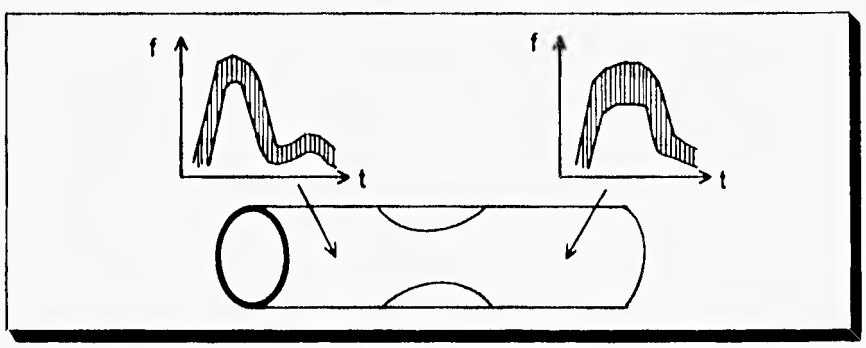


Figura 5.2. Velocidad del Flujo en las Arterias.

V.3. Estimación Espectral utilizando Algoritmos Genéticos

V.3.1. Medidor Doppler de Flujo Sanguíneo.

La figura 5.3. muestra el diagrama de bloques del detector Doppler de Flujo Sanguíneo.

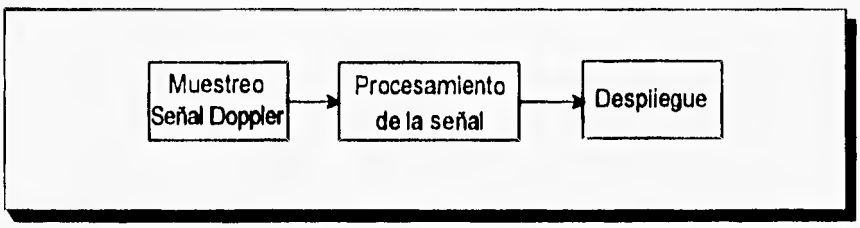


Figura 5.3. Diagrama de bloques del medidor de flujo sanguíneo.

El *Muestreo de la Señal Doppler* se realiza por medio de un transductor, colocado sobre la superficie del paciente, que detecta por medio de una señal de ultrasonido el cambio de presión local, envía una señal eléctrica que es traducida a una señal mecánica como transmisor y, las vibraciones mecánicas son cambiadas a señales eléctricas como receptor, las cuales pasarán al bloque de *Procesamiento de la señal.* Una vez obtenido el espectro, éste pasa al bloque de despliegue, el cual, se encarga de presentar en forma adecuada los resultados al usuario

V.3.2. Modelo Paramétrico.

Como se mencionó anteriormente, el modelo paramétrico para la estimación espectral consiste en elegir el modelo apropiado, estimar los parámetros de dicho modelo y

substituir los valores estimados en la expresión de Densidad de Potencia Espectral teórica. En la figura 5.4. se muestra el diagrama de bloques general de dicho proceso.

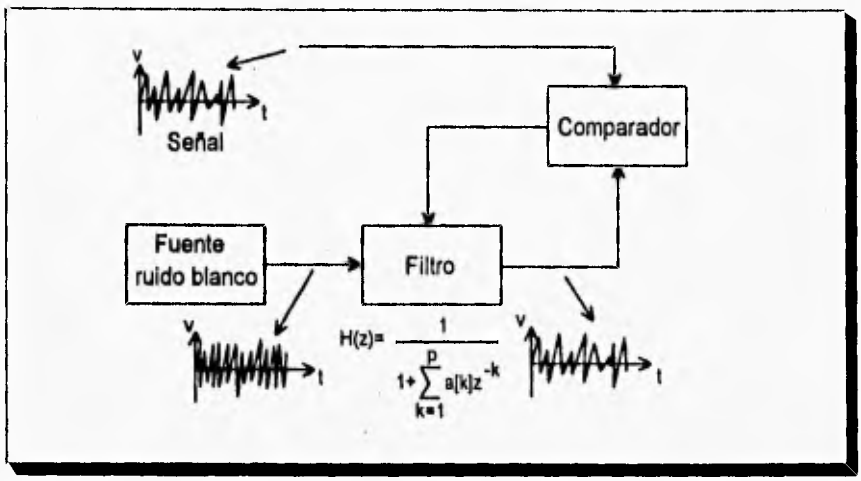


Figura 5.4. Proceso del Modelo Paramétrico.

El modelo Autorregresivo utiliza la función de autocorrelación de la señal de entrada con una señal de ruido blanco filtrado, esta igualdad se logra alterando los parámetros del filtro. La señal obtenida se compara con la señal real, así, los resultados obtenidos de dicha comparación representan la señal de error que es necesario minimizar. Estos resultados son tomados por el filtro para determinar los reajustes de los parámetros del mismo que generen una señal estimada más cercana a la real y, por tanto, reduzcan el error entre ambas señales.

V.3.3.. Implementación del modelo paramétrico Autorregresivo usando Algoritmos Genéticos.

Como se mencionó en el capítulo cuatro, para encontrar los parámetros que minimicen la función de error y modelen el sistema usando el estimador espectral AR, se requiere

primero calcular el valor de cada uno de los elemento que constituyen la matriz de covariancia para posteriormente resolver el sistema de ecuaciones resultante. Las dimensiones de la matriz de covariancia y por tanto el orden del sistema de ecuaciones, está en función del orden del modelo utilizado en el estimador espectral, con lo cual, el utilizar un orden mayor en el modelo implica un incremento en las dimensiones de la matriz, presentando el algoritmo un costo computacional mayor.

Una alternativa para el cálculo de los parámetros del modelo, la constituye el uso de Algoritmos Genéticos. Aquí el problema se reduce a minimizar la función de error dada por la siguiente expresión:

$$\hat{p} = \frac{1}{2}(\hat{p}^f + \hat{p}^b) \qquad ...5.1$$

donde:

$$\hat{p}^f = \frac{1}{N-p}\sum_{n=p}^{N-1}\left|x[n] + \sum_{k=1}^{p} a[k]x[n-k]\right|^2 \qquad ...5.2$$

y

$$\hat{p}^b = \frac{1}{N-p}\sum_{n=0}^{N-1-p}\left|x[n] + \sum_{k=1}^{p} a[k]x[n+k]\right|^2 \qquad ...5.3$$

son definidas como el error predictivo en adelanto y en atraso respectivamente.

Para efecto de la evaluación del Algoritmo Genético, es necesario definir una transformación a la función de fitness que está dada por:

$$f(x) = 1 - \frac{\hat{p}}{C_{max}} \qquad \text{para } \hat{p} \le C_{max}$$

y

$f(x) = 0$ para otros casos.

Donde C_{max} es el valor máximo de la función de error definida como 0.002 para esta aplicación [2].

En la implementación se utilizó una población de 30 individuos, donde cada individuo tiene una longitud de p parámetros por 8 bits de resolución por parámetro, usando codificación binaria. El método de selección es el método Ranking, un cruzamiento en un punto y una probabilidad de mutación de 0.2 durante tres generaciones [5] [6]. El Algoritmos es ejecutado en un procesador Transputer T805.

La figura 5.5. muestra los resultados comparativos entre los métodos de la FFT, Covariancia Modificada y Algoritmos Genéticos.

Los resultados obtenidos muestran que con pocas generaciones se puede tener una buena estimación del espectro de la señal. El paralelismo implícito de los AG's permite al algoritmo explorar diferentes regiones del espacio de soluciones simultáneamente reduciendo la complejidad que implica la implementación del algoritmo original. Esto permite la implementación de filtros de mayor orden, incrementando la resolución espectral y abriendo la posibilidad de usar métodos de estimación espectral más complejos.

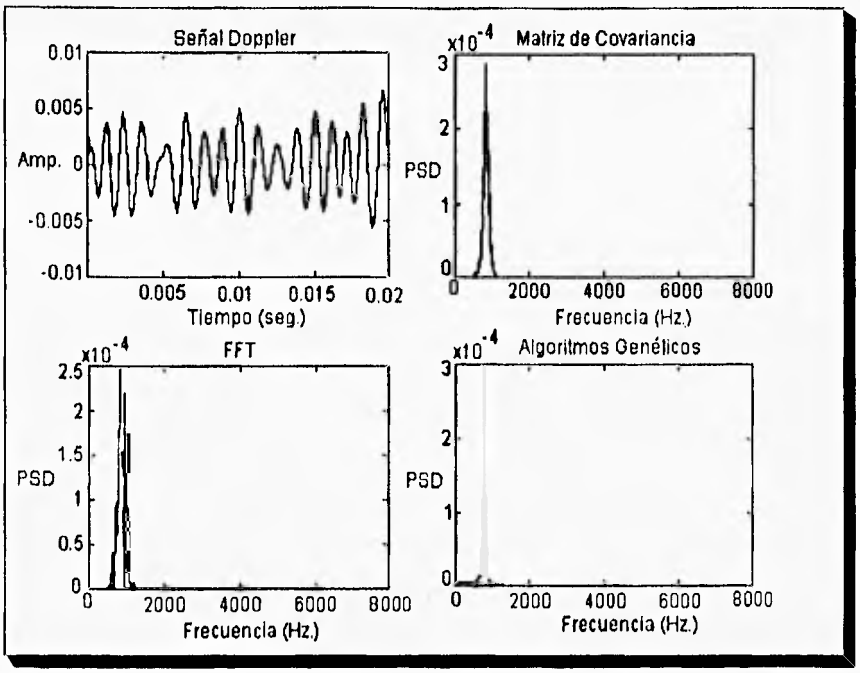


Figura 5.5. Respuestas de la señal.

No obstante y a pesar de las ventajas descritas, la implementación secuencial de los Algoritmos Genéticos dista mucho de tener la respuesta en tiempo real requerida en la aplicación médica tratada en esta tesis.

El siguiente capítulo muestra una extensión del método genético en donde se explora su naturaleza paralela, para disminuir su tiempo de procesamiento.

REFERENCIAS.

[1] García, D. F.. Solano, J.. Martínez, J.. "Heterogeneous Architecture for Parallel Real-Time Spectral Estimator in Doppler Blood Flow Instrumentation". IEE International Conference on Control 94. p.p. 37-41. 1994.

[2] Rodríguez, Katya. "Implementación de un Estimador Espectral de Señales Doppler Utilizando Algoritmos Genéticos". UNAM. 1995.

[3] Ruano, M Graca. García, D. F.. Fish, P. J.. Fleming, P. J.. "Alternative Parallel Implementationsof AR-Modified Covariance Spectral Estimator for Diagnostic Ultrasonic Blood Flow-Studies". Parellel Computing. p.p. 463-476. 1993.

[4] Solano, J.. García, D. F.. "Impiementation of a Parametric Spectral Estimator Using Genetic Algorithms". IEE International Conference on Control 94. p.p. 754-759. 1994.

[5] Solano, J.. Rodríguez, K.. "Implementación de un Estimador Espectral Paramétrico Mediante Algoritmos Genéticos". XVI Congreso Académico Nacional de Ingeniería Electrónica, ELECTRO 94. p.p. 484-494. Octubre, 1994.

[6] Solano, J.. Rodríguez, K.. "Determinación de Parámetros y Métodos de Selección en Algoritmos Genéticos". XVI Congreso Académico Nacional de Ingeniería Electrónica, ELECTRO 94. p.p. 616-626. Octubre, 1994.

CAPITULO VI.
ESTIMACION ESPECTRAL PARAMETRICA
EN TIEMPO REAL

VI.1. INTRODUCCION.

De acuerdo a los resultados mostrados en el capítulo anterior, la alternativa empleada para realizar la Estimación Espectral, es muy adecuada debido a la simplicidad inherente de los Algoritmos Genéticos.

Sin embargo, y a pesar de los buenos resultados obtenidos, la implementación realizada mostró una respuesta muy alejada al tiempo de respuesta requerido para el problema tratado en esta tesis (2 -20 ms.).

Este capítulo presenta una extensión de la aproximación genética, explotando su naturaleza paralela. El poder evolucionar diferentes subpoblaciones en forma independiente y en un número de procesadores, no sólo reduce su tiempo de ejecución, sino que mejora la calidad de respuesta obtenida por el sistema.

Varias consideraciones han tenido que ser tomadas en cuenta con el objeto de obtener una respuesta en tiempo-real. Primeramente, la reducción de la complejidad presentada por la función de error en términos de los coeficientes de la matriz de covariancia, el uso de una codificación real y la explotación en forma eficiente de las características paralelas de los Algoritmos Genéticos.

Este capítulo presenta además, un esquema de implementación paralela: el Esquema Homogéneo basado en Trasputers.

VI.2. Reducción de la Función Objetivo utilizando términos de la matriz de Covariancia Modificada.

De acuerdo a los descrito anteriormente, es necesario minimizar la ecuación de error definida por la ecuación 6.1., en la cual, intervienen dos términos definidos por 6.2. y 6.3.

$$\hat{p} = \frac{1}{2}(\hat{p}^f + \hat{p}^b) \qquad ...6.1$$

$$\hat{p}^f = \frac{1}{N-p}\sum_{n=p}^{N-1}\left|x[n] + \sum_{k=1}^{p} a[k]x[n-k]\right|^2 \qquad ...6.2$$

y:

$$\hat{p}^b = \frac{1}{N-p}\sum_{n=0}^{N-1-p}\left|x[n] + \sum_{k=1}^{p} a[k]x[n+k]\right|^2 \qquad ...6.3$$

Sin embargo, si observamos la ecuación 6.1, un gran número de operaciones son redundantes y como consecuencia, el tiempo de procesamiento requerido para evaluar dichas expresiones puede ser significativo respecto al tiempo total requerido por el Algoritmo Genético.

Considerando las características de los elementos de la matriz de covariancia, dada en la ecuación 6.4, el número de cálculos se puede reducir considerablemente.

$$c_{xx}[j,k] = \frac{1}{2(N-p)}\left(\sum_{n=p}^{N-1} x^*[n-j]x[n-k] + \sum_{n=0}^{N-1-p} x[n+j]x^*[n+k]\right) \qquad ...6.4$$

desarrollando las ecuaciones 6.2 y 6.3 se tiene

$$\hat{p}^f = \frac{1}{N-p}\sum_{n=p}^{N-1}\left|x^2[n] + 2\sum_{k=1}^{p} a[k]x[n-k]x[n] + \left(\sum_{k=1}^{p} a[k]x[n-k]\right)^2\right| \qquad ...6.5$$

y

$$\hat{p}^b = \frac{1}{N-p}\sum_{n=0}^{N-1-p}\left|x^2[n] + 2\sum_{k=1}^{p} a[k]x[n+k]x[n] + \left(\sum_{k=1}^{p} a[k]x[n+k]\right)^2\right| \qquad ...6.6$$

de la ecuación 6.4, cuando j=k=0:

$$c_{xx}[0,0] = \frac{1}{2(N-p)}\left(\sum_{n=p}^{N-1} x^*[n]x[n] + \sum_{n=0}^{N-1-p} x[n]x^*[n]\right) \qquad ...6.7$$

y si j=0 y k=1, 2, ..., p:

$$c_{xx}[0,k] = \frac{1}{2(N-p)}\left(\sum_{n=p}^{N-1} x^*[n]x[n-k] + \sum_{n=0}^{N-1-p} x[n]x^*[n+k]\right) \qquad ...6.8$$

observando el tercer término de las ecuaciones 6.5. y 6.6. y considerando las propiedades de la matriz de covariancia [1]:

$$c_{xx}[k-1,l-1] = c_{xx}[k,l] = c_{xx}[k+1,l+1] \qquad ...6.9$$

es equivalente a:

$$\sum_{k=1}^{p} a^2[k]c_{xx}[k,k] + 2\sum_{k=1}^{p-1}\sum_{m=k+1}^{p} a[k]a[m]c_{xx}[k,m]$$

Finalmente, la ecuación 6.1. puede ser reescrita de la siguiente forma:

$$\hat{p} = c_{xx}[0,0] + \sum_{k=1}^{p} a^2[k]c_{xx}[k,k] + 2\sum_{k=1}^{p} a[k]c_{xx}[0,k] + 2\sum_{k=1}^{p-1}\sum_{m=k+1}^{p} a[k]a[m]c_{xx}[k,m] \qquad ...6.10$$

Se puede observar que la sumatoria principal ha sido substituida por la constante $c_{xx'}$ la cual reduce considerablemente la expresión.

VI.3. Algoritmos Genéticos, representación real.

Generalmente, los Algoritmos Genéticos trabajan por medio de una representación binaria, la cual, no muestra una descripción del problema que se está resolviendo. Es por eso que han surgido diversas alternativas de representación de los individuos, como es el caso de la representación real.

Una representación real trae consigo ventajas sobre una representación binaria, por ejemplo, no se necesita decodificar los individuos en cada evaluación de la función objetivo, el uso de números reales hace posible un decremento de tiempo de proceso en procesadores que funcionan bajo esta representación, además de que existe una gran libertad en el uso de los diferentes operadores genéticos.

De acuerdo a las ventajas descritas, ésta es la representación utilizada en las implementaciones subsecuentes. La población inicial se crea generando N*p números reales aleatorios, donde N es el tamaño de la población y p es el orden del filtro del modelo paramétrico utilizado. Asimismo, se utiliza un esquema de selección Ranking y el método de cruzamiento y de mutación seleccionados serán los mencionados por Mühlenbein [2], quien hace un estudio comparativo entre la forma como afectan los operadores genéticos en una representación binaria y lo mapea a una representación real.

Cruzamiento.

$$z_i = x_i + \alpha(y_i - x_i) \qquad \text{con } i=1,2,\ldots,n$$

y α cambia dentro del intervalo [-0.25,1.25].

Mutación.

$$z_l = x_l \pm rango \times \delta$$

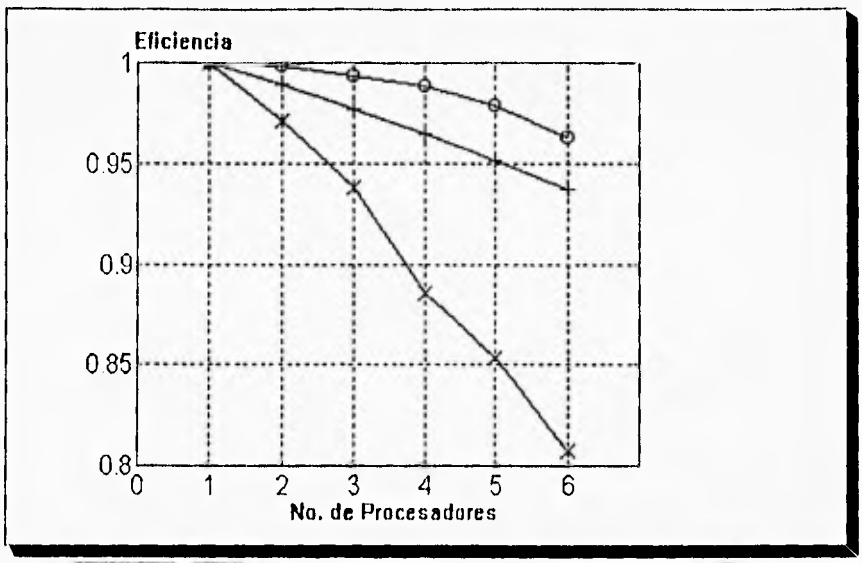
donde el signo tiene una probabilidad de 0.5 para cada evento, *rango* varía de acuerdo a los límites de posibles valores de p dentro de un individuo y δ es un numero aleatorio.

VI.4. Implementación utilizando Algoritmos Genéticos Paralelos.

De acuerdo a lo establecido anteriormente, la reducción de las expresiones que constituyen la función objetivo, así como el uso de una representación real, son fundamentales para poder reducir el tiempo de ejecución del estimador espectral tratado en este trabajo. Además de ésto, el esquema de los Algoritmos Genéticos ofrece la ventaja de poder explotar su naturaleza paralela, produciendo no sólo una aceleración en el tiempo de procesamiento, sino que la solución resultante puede mejorar.

En términos generales, un Algoritmos Genético Paralelo (AGP) está integrado por una serie de Algoritmos Genéticos Nodales, cada uno puede residir en uno o diferentes procesadores dentro del sistema. Cada nodo opera sobre una subpoblación y puede ejecutar un Algoritmo Genético, ya sea en forma independiente o interactuando con otras subpoblaciones.

Con la finalidad de evaluar la respuesta de los AGP's, en términos de eficiencia y escalabilidad, tres modelos han sido considerados [3]: el modelo de Migración, el modelo de Difusión y el modelo Farming, todos ellos implementados sobre una plataforma de Transputers.



	No. proc. 1	No. proc. 2	No. proc. 3	No. proc. 4	No. proc. 5	No. proc. 6
Migración o	1	0.9982	0.9934	0.9887	0.9789	0.9628
Difusión +	1	0.9899	0.9772	0.9643	0.9517	0.9375
Farming x	1	0.9708	0.9379	0.8859	0.8433	0.8071

Figura 6.1. Eficiencia de los modelos de Algoritmos Genéticos Paralelos.

La figura 6.1. muestra la eficiencia presentada por los tres modelos paralelos. Se observa que la eficiencia se decrementa más rápidamente para el modelo Farming que para los modelos de Migración y de Difusión. Una de las razones lo constituye la relación entre el tiempo de procesamiento y el tiempo de comunicación. En el modelo Farming, la selección es llevada a cabo exclusivamente en el procesador maestro y en consecuencia, las comunicaciones entre el maestro y los trabajadores se incrementan. Es claro que la escalabilidad mostrada por los modelos de Migración y Difusión es alta, alcanzando eficiencias del 96% y 93% respectivamente para seis procesadores.

A medida que aumenta el número de procesadores trabajando, el valor de la eficiencia es más favorecido para el modelo de Migración, a diferencia de los modelos de Difusión y Farming en donde decae considerablemente.

VI.5. Estimación Espectral Paramétrica, Esquema Homogéneo.

Con el objeto de obtener la respuesta espectral de la señal Doppler Ultrasónica, se emplea un anillo de tres procesadores que calculan los parámetros del filtro utilizado, utilizando un modelo paralelo de Migración. De acuerdo a la figura 6.2., los segmentos de datos son colectados por un Algoritmo Genético Nodal, que incluye un procesador monitor encargado de distribuir trabajo a los otros procesadores en el anillo. Al cumplir un número de generaciones predefinido, cada uno de los nodos envía su mejor elemento local hacia el monitor, que a su vez, escoge el mejor de todos y envía el conjunto óptimo de parámetros al siguiente procesador conectado en línea, para que efectúe el cálculo de la Densidad de Potencia Espectral.

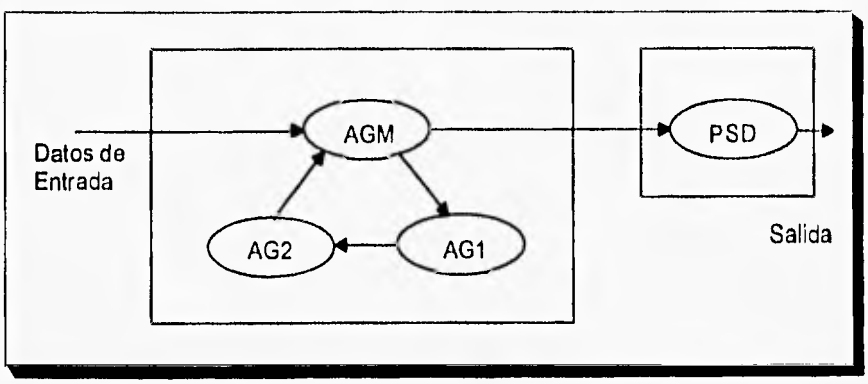
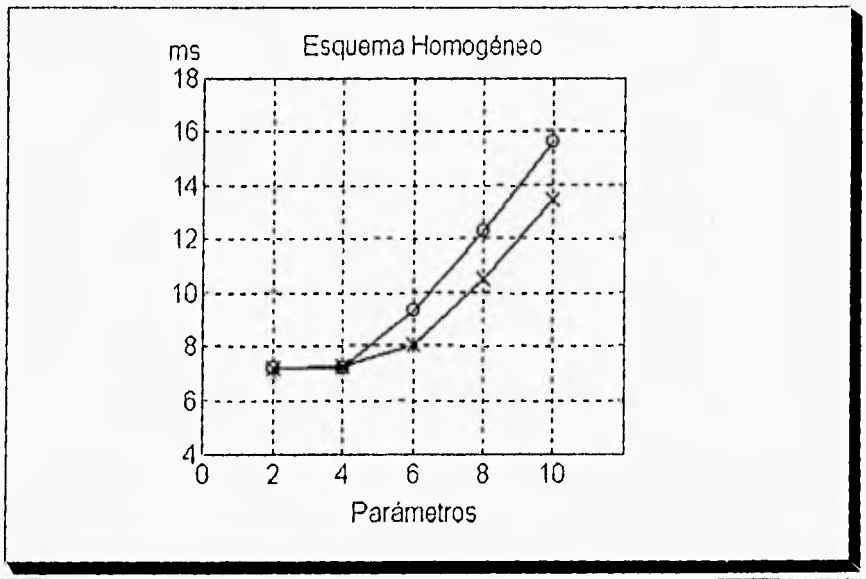


Figura 6.2. Esquema Homogéneo.

Varios segmentos de una señal Doppler simulada han sido utilizados para medir el desempeño de este esquema. Cada segmento contiene 256 muestras

correspondientes a un muestreo de 10 ms. Los resultados de la figura 6.3. muestran los tiempos de procesamiento cuando se aplica una serie de segmentos consecutivos a la estructura de pipeline. Se demuestra entonces, la efectividad del esquema obteniendo un tiempo promedio por segmento de 9.676 ms. para un modelo de orden p=6.

Asimismo, la figura 6.3. muestra los resultados obtenidos al agregar reinserción con GGAP=0.8, reduciendo los tiempos de procesamiento en un 13% y manteniendo su respuesta espectral.



	P=2	P=4	P=6	P=8	P=10
AGP o	7.1509	7.1723	9.2843	12.2368	15.6117
AGP+Reins. x	7.1509	7.1765	7.9786	10.4917	13.4186

Figura 6.3. Arquitectura Homogénea. AGP.

Los resultados muestran también que, para modelos de orden p=2 y p=4, no hace diferencia al utilizar reinserción, debido a que en ambos casos, el tiempo máximo secuencial está determinado por la evaluación de la Densidad de Potencia Espectral al final de la línea de procesadores.

Por tanto, al agregar procesadores al AGP, no se reflejaría en un beneficio en el tiempo global del sistema. Para el caso de modelos de orden $p \geq 6$, la inclusión del esquema de reinserción se hace evidente, debido a que el tiempo máximo secuencial está determinado por el cálculo de los parámetros.

Una alternativa para disminuir tiempo de procesamiento, es por medio de una implementación heterogénea, definida en el siguiente capítulo.

REFERENCIAS.

[1] Kay, Steven M.. "Modern Spectral Estimation. Theory and Application". Prentice Hall. 1988.

[2] Mühlenbein, Heinz. Schierkamp-Voosen, Dirk. "Predective Models for the Breeder Genetic Algorithms I. Continuous Parameter Optimization". Evolutionary Computation. Vol. 1. No. 1. p.p. 25-49. Spring 1993.

[3] Rodríguez, Katya. "Implementación de un Estimador Espectral de Señales Doppler Utilizando Algoritmos Genéticos". UNAM. 1995.

CAPITULO VII.

ESTIMACION ESPECTRAL PARAMETRICA, ESQUEMA HETEROGENEO

VII.1. INTRODUCCION.

Como se ha descrito en capítulos anteriores, la aproximación genética ha sido utilizada con el objeto de encontrar el conjunto óptimo de parámetros del filtro del modelo que minimice la función de error predictivo promedio. El reducir la complejidad de la expresión que modela dicho error, en conjunto con una codificación real de los elementos utilizados en el Algoritmo Genético Paralelo, son fundamentales para obtener una respuesta en el tiempo requerido por el sistema.

El capítulo anterior introdujo la implementación paralela del sistema basada en un Esquema Homogéneo, es decir, utilizando el mismo tipo de procesador para todos los elementos de procesamiento. Este capítulo introduce un Esquema Heterogéneo que incorpora un Nodo de Procesamiento Heterogéneo (NPH) con el fin de incrementar la eficiencia del sistema. Se presenta también un análisis de desempeño incluyendo un estudio de balanceo de cargas, fundamental para la eficiente implementación de este esquema.

VII.2. Esquema Heterogéneo.

De acuerdo a los resultados obtenidos hasta ahora, se ha demostrado que el esquema de pipeline homogéneo puede obtener una respuesta del sistema en tiempo real para modelos hasta orden p=6. Asimismo, se observó que se podría mejorar la respuesta del sistema al incrementar el número de procesadores que calculan los parámetros del modelo, con lo cual, se abre la posibilidad de poder utilizar modelos más complejos bajo el mismo esquema.

Por otro lado, el tiempo del cálculo de la FFT y, por consiguiente, el cálculo de la PSD implica un tiempo de procesamiento grande en procesadores de propósito general. Asimismo, no resultaría conveniente disminuir el número de datos al ejecutar la FFT para obtener un tiempo de procesamiento menor, ya que se estaría reduciendo también la resolución, muy importante en la obtención de la representación en frecuencia de las señales Doppler.

Esta sección describe el uso de un esquema distinto, usando un Nodo Heterogéneo (Transputer-DSP) tal que explote la naturaleza del algoritmo utilizado, procesando las tareas más adecuadas al procesador en turno, de tal forma que sean ejecutadas más eficientemente.

La figura 7.1. muestra el diagrama de bloques del Esquema Heterogéneo. A diferencia del Esquema Homogéneo, uno de los procesadores que realiza un AG nodal es substituido por un NPH.

El NPH incluye un transputer que contiene y ejecuta un AG nodal y trabaja en forma similar al Esquema Homogéneo. No obstante, cuando todos los AG nodales han generado sus subpoblaciones, parte de los elementos de cada subpoblación son enviados al DSP integrado al NPH para que éste les asigne un valor de fitness. Los valores resultantes son entonces regresados a su nodo correspondiente, de tal manera

que, el proceso del Algoritmo Genético Paralelo continúe como en el Esquema Homogéneo. Una vez encontrado el conjunto de parámetros óptimo, se envía a otro procesador en el sistema (pudiendo ser éste un Transputer o un NPH) para el cálculo de la Densidad de Potencia Espectral.

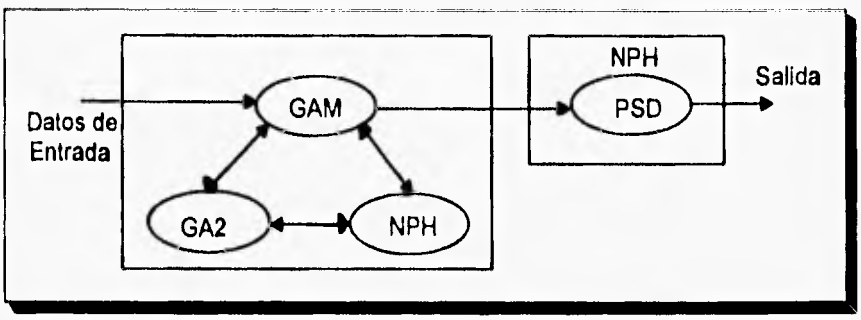


Figura 7.1. Esquema Heterogéneo.

VII.2.1. Nodo de Procesamiento Heterogéneo (NPH)

La figura 7.2. muestra el diagrama general del Nodo de Procesamiento Heterogéneo.

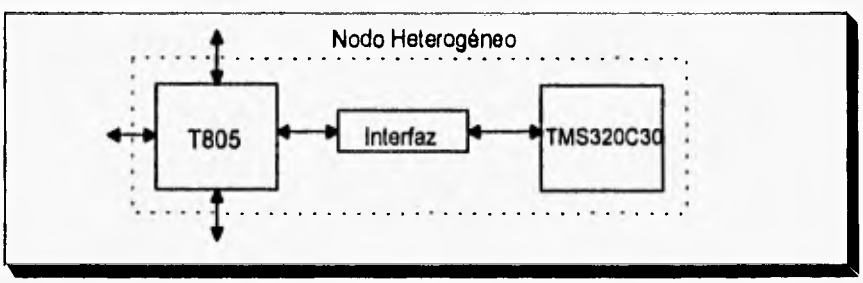


Figura 7.2. Esquema del Nodo de Procesamiento Heterogéneo.

El Nodo de Procesamiento Heterogéneo (NPH) puede ser fácilmente integrado a una plataforma de transputers, e incluye un número de bibliotecas para cargar y ejecutar

código en el DSP en forma transparente desde el transputer. A continuación se describen sus características.

El NPH integra un Transputer T805, un Procesador Digital de Señales TMS320C30 y una interfaz para la comunicación entre ellos.

El NPH utiliza un link del Transputer para comunicación con el DSP. Los tres links restantes del Transputer son utilizados para comunicación con otros Transputers. Por tanto, hasta tres procesadores pueden tener acceso al NPH, el cual, actuará como un coprocesador del sistema. Sin embargo, hay que remarcar la necesidad de que el *overhead* de comunicaciones entre el DSP y el Transputer sea lo más bajo posible, es decir, se requiere una interfaz muy eficiente.

Para su desarrollo se evaluaron dos alternativas diferentes, una de ellas usando un *link adaptor* controlado por lógica para la transferencia entre los dos componentes y la otra por medio del mapeo de memoria entre los dos procesadores [2]

VI.2.2. Interfaz de comunicación.

Considerando los dos procesadores a comunicar, existen grandes diferencias en cuanto a su hardware, así como su software y la representación de punto flotante. Mientras que el T805 utiliza el estándar IEEE para su representación, el DSP utiliza una representación propia ya descrita en el capítulo II.

La comunicación entre Transputers se realiza por medio de un link serial que manda datos con un cierto protocolo en bloques de bytes, así, si se necesita enviar un número en punto flotante de 32 bits, se envían por bloques de ocho bits con dos bits altos al inicio y un bit bajo al final por bit.

El DSP, en cambio, recibe los datos a través del puerto serie con un protocolo distinto y con bloques de 8 bits.

Interfaz por medio de un *Link Adaptor*.

El diagrama de bloques de la interfaz basada en un *link adaptor*, se muestra en la figura 7.3. En ella se observan sus componentes: Interfaz Link Adaptor, Bloque de Transmisión y Bloque de Recepción.

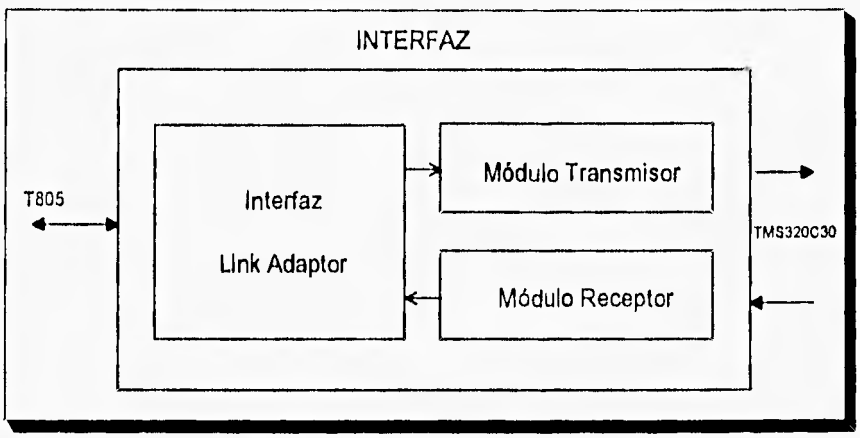


Figura 7.3. Interfaz de comunicación entre T805 y TMS320C30.

La Interfaz Link Adaptor formada por un driver diferencial DS8921, un oscilador de 5MHz, un conector de E/S de 8 pines y un link adaptor IMS C011 es el bloque que se encarga de, por un lado, convertir los datos de serie a paralelo (ésto con la finalidad de poder cambiar el tipo de protocolo para ser aceptado por el TMS320) y de cambiar los datos que vienen del bloque de Recepción de un formato paralelo a un formato serie con el protocolo del transputer para que éste pueda recibirlos.

El Módulo de Transmisión está formado por un contador síncrono de 4 bits, un registro de corrimiento de 8 bits con entrada paralela, salida serie, dos flip-flops y un circuito

de reset. Dicho módulo se encarga de cambiar de formato paralelo a formato serie. Los datos son enviados al TMS320 por medio de bloques de ocho bits, así, si se mandan datos en punto flotante compuesto por 32 bits, es necesario recibir los datos en cuatro bloques de 8 bits para poder completar el número.

El Módulo de Recepción está formado por un registro de corrimiento de 8 bits con entrada serie y salida paralela, un contador síncrono de 4 bits y dos flip-flops. Este módulo se encargará de recibir los datos por medio del TMS320 en bloques de 8 bits. Una vez obtenidos los 32 bits que forman a un número en punto flotante, se envía el dato a la Interfaz Link Adaptor para así, modificar su protocolo y ser enviado al transputer [1].

Interfaz por medio de mapeo de memoria.

En la interfaz de mapeo de memoria, la comunicación se realiza directamente sobre la memoria donde se almacenan los datos, es decir, no se requiere un mapeo de la información entre el puerto serie del DSP y la memoria en donde finalmente se alojará la información.

En la interfaz de mapeo de memoria, la comunicación puede darse al leer y escribir datos en la memoria compartida, utilizándose interrupciones para solicitar la atención de un procesador hacia la memoria dual. Por ejemplo, si un procesador A desea comunicarse con un procesador B, éste almacena la información y escribe a la bandera de interrupción de la memoria, lo cual, ocasiona que la línea de interrupción del procesador B se active. La interrupción será borrada por el procesador B al leer los datos. El procesador B ejecuta una serie de tareas y sus resultados son enviados de regreso a la memoria, donde son almacenados.

Los programas de comunicación entre el T805 y el DSP, necesarios ejecutar dentro del DSP son mostrados en el Apéndice B.

VII.3. ANALISIS DE DESEMPEÑO DEL ESQUEMA HETEROGENEO.

Los resultados obtenidos por el sistema homogéneo exhibieron que, para poder lograr tiempo-real en modelos de orden mayor a p=6, dos aproximaciones podían llevarse a cabo.

La primera es incrementar el número de procesadores involucrados en calcular los parámetros del filtro. Sin embargo, el uso de esta alternativa resultaría en un decremento de la calidad de la respuesta espectral debido a la reducción en el número de los miembros de las subpoblaciones de cada AG Nodal, además de que lo que implica agregar un Transputer entra en términos del costo del sistema.

La segunda aproximación es el integrar un NPH al sistema, donde el DSP actuaría como un coprocesador, no sólo para el transputer que integra el NPH, sino para los otros procesadores incluidos en la topología de anillo. El esquema mantiene la calidad de la respuesta y el costo del DSP extra es bajo.

Con el objeto de evaluar las ventajas de incluir un NPH al sistema en estudio, se efectúa primeramente un análisis respecto al tiempo de procesamiento cuando el NPH ejecuta la PSD, es decir, es incluido en la última etapa del Pipeline. Asimismo, se efectúa un estudio similar al ser incluido en la topología de anillo para el cálculo de los parámetros del filtro.

VII.3.1. Esquema Heterogéneo con NPH ejecutando la PSD.

El esquema utilizado se muestra en la figura 7.4, el cual, utiliza el Nodo de Procesamiento Heterogéneo para el cálculo de la PSD, mientras que el cálculo de los parámetros se realiza en tres procesadores T805, por medio de Algoritmos Genéticos Paralelos.

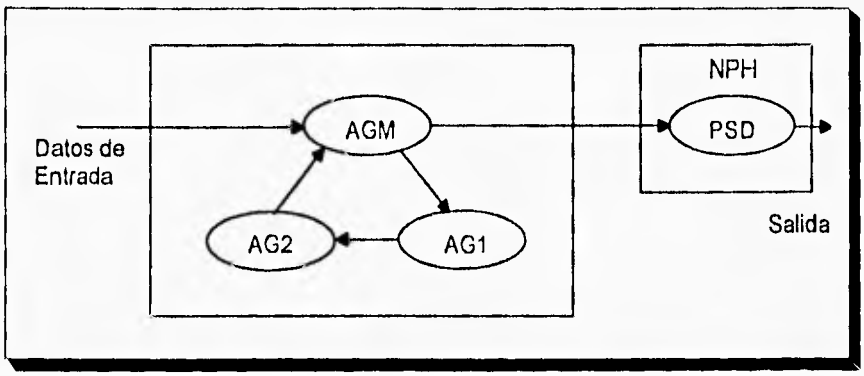


Figura 7.4. NPH en el cálculo de PSD.

Con el fin de evaluar esta implementación, primeramente se llevan a cabo mediciones de tiempo de procesamiento para el cálculo de la Densidad de Potencia Espectral, variando para ésto, el orden del modelo y efectuando el proceso exclusivamente en el NPH. Los tiempos obtenidos (que incluyen la comunicación a través de un link adaptor entre el T805 y el DSP) se muestran en la tabla 7.1.

	P=2	P=4	P=6	P=8	P=10
PSD	5.527	5.592	5.660	5.723	5.790

Tabla 7.1. Tiempo de procesamiento en el NPH (ms.)

Asimismo, con el objeto de obtener mediciones precisas del esquema total mostrado en la fig. 7.4., se utilizaron diez ventanas de datos para cada proceso, dando los resultados mostrados en la tabla 7.2.

ventana/ord	P=2	P=4	P=6	P=8	P=10
1	9.268	11.270	14.944	17.960	21.401
2	15.226	17.889	24.227	30.196	37.015
3	21.159	24.495	33.513	42.432	52.625
4	27.113	31.173	42.798	54.670	68.235
5	33.047	37.839	52.081	66.905	83.848
6	39.004	44.480	61.370	79.144	99.461
7	44.937	51.155	70.650	91.381	115.071
8	50.885	57.751	79.932	103.620	130.682
9	56.818	64.426	89.219	115.854	146.295
10	62.764	70.982	98.503	128.088	161.907

Tabla 7.2. Tiempo de una secuencia de diez ventanas de datos (ms.)

Con lo cual, el tiempo promedio entre ventanas, es el siguiente.

	P=2	P=4	P=6	P=8	P=10
AGP	5.944	6.645	9.284	12.237	15.612

Tabla 7.3. Tiempo de procesamiento (ms.)

Comparando el tiempo de procesamiento total mostrado en la tabla 7.3 y los tiempos obtenidos utilizando el Esquema Homogéneo mostrado en la figura 6.2., es claro que, para modelos de orden $p \leq 4$, el tiempo de ejecución está determinado por el cálculo de la PSD y, por tanto, el incluir un NPH en el último estado del pipeline se refleja en una reducción en el tiempo de procesamiento de dichos modelos. No así para modelos de orden $p \geq 6$, donde, el tiempo de ejecución está determinado por el cálculo de los parámetros. Cabe señalar que la reducción en el tiempo para modelos menores a $p \leq 4$, se logra, no obstante, utilizando una interfaz basada en un *link adaptor*, la cual, resulta ineficiente en términos de velocidad, como se verá más adelante.

VII.3.2. Esquema Heterogéneo con NPH en el cálculo de los parámetros.

De acuerdo a los resultados obtenidos en el Esquema Homogéneo, se tiene que, el tiempo del sistema para modelos de orden mayor a p=6 es regido por el cálculo de los parámetros. En estos casos, se puede utilizar el NPH para la evaluación de la función objetivo incluida en el AG Paralelo, como lo muestra la figura 7.5.

No obstante que se prevé una mejoría en el desempeño del sistema, es preciso efectuar un análisis previo de tal forma que el modelo paralelo de migración esté balanceado. Es necesario, por tanto, conocer cual será el porcentaje de individuos a mandar al DSP tanto por el Algoritmo Genético asociado a él, como por los otros dos procesadores incluidos en el anillo.

Existen dos alternativas para balancear el sistema propuesto, una por medio del balanceo dinámico y la otra por balanceo estático. En el balanceo dinámico, aún cuando es un procedimiento muy eficiente, en nuestro caso no resultaría así, debido al overhead adicional inherente al proceso y al número pequeño de tareas a distribuir. Por otro lado, el balanceo estático, aunque requiere un estudio más cuidadoso acerca de los procesos involucrados, se adapta más al problema abordado y, por tanto, es el procedimiento utilizado.

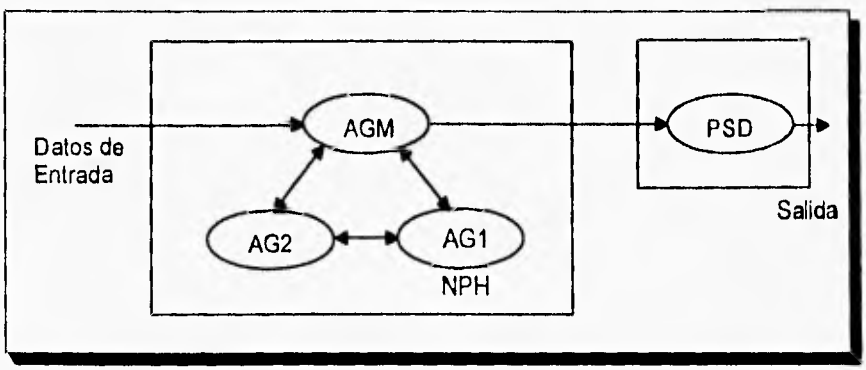


Figura 7.5. NPH como una población de los AGP's.

La tabla 7.4 muestra los tiempos de ejecución de la función objetivo para un individuo tanto en el Transputer T805 como en el DSP TMS320C30 de acuerdo al orden del modelo considerado.

Cabe señalar que el tiempo mostrado para el TMS320C30 ya incluye el tiempo requerido para cambio de formato entre los procesadores.

	P=2	P=4	P=6	P=8	P=10
T805	0.058	0.077	0.102	0.159	0.231
TMS320C30	0.044	0.055	0.069	0.103	0.146

Tabla 7.4. Tiempo comparativo de procesamiento (ms.)

En la figura 7.6. se puede observar la diferencia de tiempos entre los dos procesadores. Es claro que el DSP ejecuta la operación en un tiempo más reducido que el T805. También, se observa que a medida que aumenta el número de parámetros, el porcentaje de ahorro de tiempo se incrementa (24%, 28.5%, 32.3%, 35.2% y 37% respectivamente).

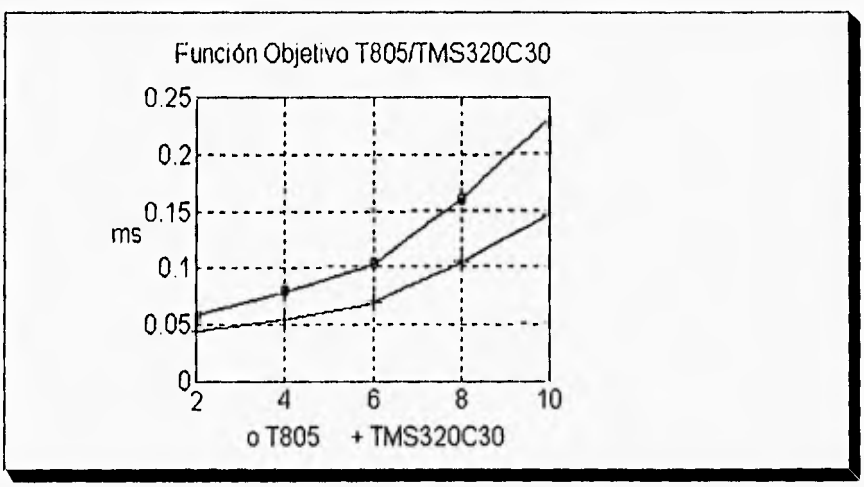


Figura 7.6. Estudio de tiempo de procesamiento.

La tabla 7.5. muestra el tiempo de evaluación de la función objetivo en el NPH y en el transputer, si este último trabajara como el DSP. Cabe señalar que se utiliza una interfaz basada en un *link adaptor* para comunicar el T805 y el DSP.

	P=2	P=4	P=6	P=8	P=10
T805	0.063	0.086	0.114	0.175	0.250
NPH	0.088	0.128	0.170	0.232	0.305

Tabla 7.5. Tiempo comparativo de procesamiento (ms.).

Es claro que, al integrar un NPH al anillo de procesadores que realiza el AG Paralelo, no reflejará ninguna disminución en el tiempo total de procesamiento del sistema, ya que, el tiempo que tarda el NPH en evaluar un individuo es mayor que en el transputer.

Lo anterior es debido a que, la comunicación a través de un *link adaptor* es ineficiente y, por tanto, esta interfaz no resulta conveniente para nuestro sistema.

VII.3.3. Esquema Heterogéneo con NPH por mapeo de memoria.

Este esquema de comunicación para el NPH se basa en el intercambio de información en forma directa entre los dos tipos de procesadores. El transputer seguirá mandando la información por medio de un link de comunicación, pero el DSP recibirá la información en memoria. Esto trae consigo la ventaja de que, no es necesario hacer un mapeo de la información del puerto serie al espacio de memoria en que residirá finalmente la información.

La tabla 7.6. muestra el tiempo que tarda el NPH de memoria compartida en la evaluación de la función objetivo de un individuo, para los diferentes parámetros que se han venido considerando, en relación a lo que tarda un T805, si este último trabajara como el DSP.

	P=2	P=4	P=6	P=8	P=10
T805	0.063	0.086	0.114	0.175	0.250
NPH	0.059	0.079	0.103	0.147	0.200

Tabla 7.6. Tiempo comparativo de procesamiento (ms.).

Se observa que el tiempo de procesamiento más comunicación en el NPH es menor en todos los casos.

De acuerdo a los resultados mostrados, es necesario encontrar el balanceo óptimo del sistema. Para ello, la subpoblación residente en el transputer del NPH debe mandar un porcentaje de individuos al DSP y dar servicio a las peticiones de las otras dos subpoblaciones que serán enviados por los otros procesadores al DSP (ver figura 7.5).

Caso	Individuos evaluados en AG1	Tiempo en AG1	AGM manda al DSP	AG2 manda al DSP	AG1 manda al DSP	Tiempo en DSP
1	10	1.020	0 (10)	0 (10)	0 (10)	0
2	9	0.918	1 (9)	1 (9)	1 (9)	0.207
3	8	0.816	2 (8)	2 (8)	2 (8)	0.414
4	7	0.714	3 (7)	3 (7)	3 (7)	0.621
5	6	0.612	4 (6)	4 (6)	4 (6)	0.828
6	6	0.612	3 (7)	3 (7)	4 (6)	0.690
7	6	0.612	2 (8)	2 (8)	4 (6)	0.552
8	5	0.510	1 (9)	1 (9)	5 (5)	0.483
9	5	0.510	2 (8)	2 (8)	5 (5)	0.621

Tabla 7.7. Casos de estudio para p=6.

La tabla 7.7. muestra nueve diferentes casos de distribución de actividades hacia el NPH. Las columnas 2 y 3 muestran el número de individuos evaluados en el nodo AG1 junto con el tiempo de procesamiento, sin contar comunicaciones. Las tres columnas siguientes indican el número de individuos que cada uno de los nodos (AGM, AG2 y AG1) envian hacia el DSP y entre paréntesis el número de individuos

procesados localmente. En la última columna se muestra el tiempo de procesamiento en el DSP para cada caso.

Se busca entonces, el mayor balanceo posible de la configuración, tratando de aprovechar al máximo el DSP.

Considerando el caso 8, se observa que el tiempo de procesamiento en AG1 y DSP está balanceado, sin embargo, los nodos AGM y AG2 deberán de evaluar 9 funciones respectivamente (0.918 ms.), con lo que, los nodos AG1 y DSP estarían desocupados un tiempo considerable. En cambio, para el caso 7, AGM y AG2 evaluarán 8 funciones respectivamente, disminuyendo el tiempo ocioso de los otros nodos, al incluir el tiempo de comunicación entre procesadores y, por tanto, es el balanceo óptimo del sistema.

Con estas condiciones, la tabla 7.8. muestra los resultados para las dos implementaciones realizadas (con y sin reinserción).

	P=2	P=4	P=6	P=8	P=10
AGP	7.151	7.172	8.468	10.965	13.764
AGP+REINS	7.151	7.176	7.113	9.2106	11.562

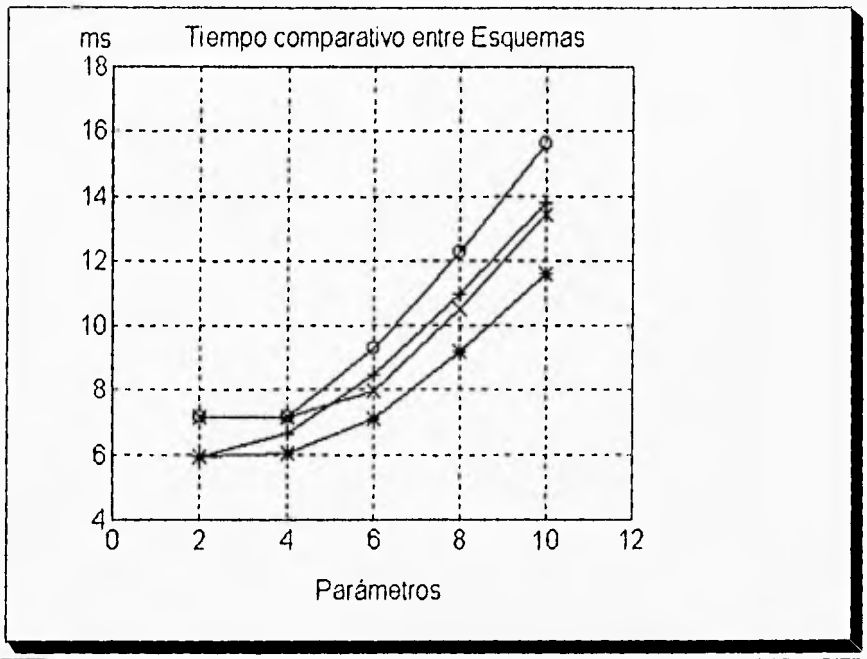
Tabla 7.8. Tiempo comparativo de procesamiento (ms.)

Se observa que el tiempo es regido por el cálculo de la PSD en p=2 y p=4. En los demás casos el tiempo es regido por el cálculo de los parámetros por medio de los Algoritmos Genéticos.

Los programas del Esquema Heterogéneo son mostrados en el Apéndice A.

La figura 7.7. resume los resultados obtenidos en la implementación heterogénea con las diferentes configuraciones utilizadas. Para modelos de orden $p \leq 4$ el tiempo de ejecución está determinado por el cálculo de la PSD y por tanto, un NPH sustituye al transputer en la última etapa de pipeline. Para modelos de orden p>4, se incluye un

NPH en el anillo de procesadores que realizan el cálculo de los parámetros del filtro. La figura exhibe una reducción en los tiempos de ejecución para todos los órdenes del modelo y, por tanto, demuestra la factibilidad de obtener respuestas del sistema en tiempo real para modelos hasta de orden p=8 (incluyendo reinserción) manteniendo además, su respuesta espectral.



Simbología	Esquema	P=2	P=4	P=6	P=8	P=10
o	Homo.	7.151	7.172	9.284	12.237	15.612
x	Homo+Re-ins.	7.151	7.176	7.978	10.492	13.418
+	Hetero.	5.944	6.646	8.468	10.965	13.764
*	Hetero+ Reins	5.933	6.017	7.113	9.211	11.562

Figura 7.7. Tiempo comparativo (ms.)

Referencias.

[1] García, D. F.. Solano, J.. Martinez, J.. "Heterogeneous Architecture for Parallel Real-Time Spectral Estimator in Doppler Blood Flow Instrumentation". IEE International Conference on Control 94. p.p. 37-41. 1994.

[2] Ramos, D. "Diseño e Implementación de un Nodo Heterogéneo de Procesamiento Digital de Señales Utilizando un Transputer y un DSP". UNAM. 1995.

CAPITULO VIII.
CONCLUSIONES

VIII.1 CONCLUSIONES GENERALES.

El trabajo presentado muestra conclusiones importantes que son necesarias subrayar dentro de esta sección.

Parte fundamental del trabajo desarrollado en esta tesis, ha sido el poder utilizar eficientemente el paralelismo inherente a los Algoritmos Genéticos, tanto para disminuir el tiempo de respuesta del sistema, como para mejorar la calidad de la solución encontrada. Asimismo, esta alternativa reduce la complejidad computacional que presentan otros métodos como el Método de Covariancia Modificada, manteniendo la respuesta espectral.

No obstante las ventajas descritas, varias consideraciones fueron tomadas en cuenta para obtener una respuesta del sistema en tiempo-real. Una reducción de la complejidad presentada por la función de error en términos de los coeficientes de la matriz de covariancia, el uso de una codificación real y la explotación en forma eficiente de las características paralelas de los Algoritmos Genéticos.

Tres Modelos Genéticos Paralelos fueron evaluados para su implementación: el Modelo de Migración, el Modelo de Difusión y el Modelo Farming. Los resultados obtenidos mostraron eficiencias del 96%, 93% y 80% respectivamente, utilizando hasta seis procesadores. Esto demuestra claramente el alto grado de escalabilidad de los modelos de Difusión y Migración a diferencia del modelo Farming.

CAPITULO VIII. Conclusiones

Con el objeto de obtener la respuesta espectral de la señal Doppler Ultrasónica, se empleó una estructura en Pipeline con un anillo de tres procesadores T805 que calculan los parámetros del filtro del modelo paramétrico, empleando para ello, un modelo paralelo de Migración en línea con otro procesador T805 para el cálculo de la PSD. De esta implementación, denominada Esquema Homogéneo, se observó que, cuando el orden del filtro es p=2 y p=4, el tiempo de procesamiento es regido por el cálculo de la PSD, mientras que, para $p \geq 6$, el tiempo lo rige el cálculo de los parámetros.

Los resultados obtenidos sugieren que para poder lograr una respuesta del sistema en tiempo-real para modelos de orden $p \geq 6$, dos alternativas pueden ser tomadas. La primera es incrementar el número de procesadores involucrados en el cálculo de los parámetros, es decir, en el anillo de Transputers. Sin embargo, ésto reducirá la calidad de la respuesta espectral debido a la reducción del número de elementos de cada subpoblación, además de incrementar el costo del sistema al agregar otro Transputer.

La segunda alternativa se basa en la integración de un NPH al sistema, de tal forma que, el DSP que incorpora este nodo actúe como coprocesador no solo del nodo, sino de los otros procesadores en el anillo. El esquema, además, mantiene la calidad de la respuesta y el DSP adicional es de bajo costo.

De esta forma, se llevó a cabo la implementación del Esquema Heterogéneo propuesto, donde se incluye un Nodo de Procesamiento Heterogéneo (NPH) formado por un T805, un TMS320C30 y una interfaz, para tratar de disminuir tiempo de procesamiento.

La utilización de un DSP, en conjunto con un Transputer, ha dado como resultado un elemento de procesamiento poderoso y flexible, aprovechando las ventajas que el DSP presenta al efectuar operaciones propias del procesamiento de señales en forma

eficiente e incorporando las excelentes características de comunicación entre procesadores propios del Transputer.

Los resultados demuestran que, aun cuando se logra una respuesta del sistema en tiempo real para ambos Esquemas, Homogéneo y Heterogéneo, posibles mejoras al sistema homogéneo, pueden resultar caras en términos de respuesta espectral y costo, a diferencia del Esquema Heterogéneo utilizado.

El Esquema Heterogéneo ha probado su factibilidad de implementación en el caso de Flujometría Doppler presentado, incrementando su alcance respecto a los resultados obtenidos con el Esquema Homogéneo y logrando una respuesta en tiempo-real para modelos de orden p=8. Más aun, es importante enfatizar que los resultados mejorarán conforme la complejidad del problema tratado sea mayor.

Finalmente, el trabajo descrito en esta tesis ha presentado una alternativa para el Análisis Espectral de Señales Doppler Ultrasónicas, en tiempo-real, mediante el uso de Algoritmos Genéticos Paralelos. No obstante, los métodos descritos no son exclusivos de esta aplicación, pudiendo ser empleados para dar solución a problemas más generales.

VIII.2 TRABAJO FUTURO.

Los resultados obtenidos en este trabajo abren nuevas posibilidades para el uso del Nodo de Procesamiento Heterogéneo en otras aplicaciones, tal es el caso del Procesamiento Digital de Imágenes, en donde, el tiempo de procesamiento es muy grande y con un Nodo Heterogéneo el tiempo se puede reducir en el cálculo de la Transformada Rápida de Fourier o en algoritmos de procesamiento como interpolación, difuminación, detección de bordes, etc.

Por otro lado, se puede continuar esta línea de investigación si se utiliza una nueva generación de Procesadores Digitales de Señales, el ADSP 21060 SHARC, de Analog Devices, el cual, es el DSP más poderoso en el mercado y que incorpora seis links para comunicación con otros procesadores.

APENDICE A.
PROGRAMAS, ESQUEMA HETEROGENEO

A.1. ESQUEMA HETEROGENEO CON EL NPH EJECUTANDO LA PSD.

Este grupo de programas utiliza el Nodo Heterogéneo en la evaluación de la PSD, para lo cual, es necesario los programas en OCCAM MONITOR.OCC, AGM.OCC, AG1.OCC, AG2.OCC y BUFFER.OCC para los transputers, así como los programas FFTPSD.C en C y FFT.ASM (este último desarrollado por Texas Instrument, presentado en el libro "Digital Signal Processing Aplications" de Panos E. Papamichalis) en lenguaje ensamblador para el DSP. Finalmente el programa CONFIGU.PGM para configurar la red de la figura 7.4.

--MONITOR.OCC

```
... INCLUDE'S "hostio.inc"
... USE'S
PROC monitor(CHAN OF SP fs,ts, CHAN OF ANY in, out, avisa)
  ... DECLARACION DE VARIABLES.
  SEQ
    error:=TRUE
    WHILE error
      SEQ
          ... LECTURA DE PARAMETROS
    ... CALCULO DE LA MATRIZ DE COVARIANCIA
    ... INICIALIZACION DEL DSP
    SEQ j=0 FOR 20
      SEQ
          ALT
            out ? fft
              SEQ
                ... RECIBE DATOS DE BUFFER
            avisa ? tiempo
              SEQ
                ... MANDA DATOS A AGM
    ... IMPRESION DE TIEMPO DE PROCESAMIENTO
    ... GUARDA DATOS EN ARCHIVO
:
```

--AGM.OCC

```
... INCLUDE'S
... USE'S
PROC agm(CHAN OF ANY in, t12, t13, t21, t31, une, avisa, avisa1)
  ... DECLARACION DE VARIABLES
  PRI PAR
    SEQ
      ... CREACION DE LA POBLACION INICIAL
      SEQ ven=0 FOR 10
          SEQ
            ... VALORES OBTENIDOS POR EL MONITOR
            ... VALORES MANDADOS A GA1
            ... EVALUACION
            ... ESTADISTICO
            ... SINCRONIZACION DE PROCESOS
            IF
              ven=0
                ... INICIALIZA EL DSP
              TRUE
                SKIP
            --INICIALIZACION DEL CICLO DEL ALGORITMO GENETICO HASTA QUE SE
            --CUMPLA EL NUMERO DE GENERACIONES maxgen
            SEQ generacion=0 FOR maxgen
              SEQ
                ... SELECCION, REPRODUCCION (SI EXISTE) Y CRUZAMIENTO
                ... MUTACION
                ... EVALUACION
                ... MIGRACION
                ... ESTADISTICO
            --FIN DEL ALGORITMO GENETICO
            ... MANDA OPTIMO AL BUFFER
    SKIP
:
```

--AG1.OCC

```
... INCLUDE's
... USE'S
PROC ag1(CHAN OF ANY t31, t32, t13)
  ... DECLARACION DE VARIABLES
  PRI PAR
    SEQ
      ... CREACION DE LA POBLACION INICIAL
      SEQ ven=0 FOR 10
          SEQ
            ... VALORES OBTENIDOS POR AgM
            ... VALORES MANDADOS A AG2
            ... EVALUACION
            ... ESTADISTICO
            IF
              Ven=0
                ... INICIALIZA EL DSP
              TRUE
                SKIP
            ... SINCRONIZACION DE PROCESOS
```

```
        --INICIALIZACION DEL CICLO DEL ALGORITMO GENETICO HASTA QUE SE
        --CUMPLA EL NUMERO DE GENERACIONES maxgen
        SEQ generacion=0 FOR maxgen
          SEQ
            ... SELECCION, REPRODUCCION (SI EXISTE) Y CRUZAMIENTO
            ... MUTACION
            ... EVALUACION
            ... MIGRACION
            ... ESTADISTICO
        --FIN DEL ALGORITMO GENETICO
        ... ENVIO DE MEJOR INDIVIDUO A AG2
  SKIP
:
```

--AG2.OCC

```
... INCLUDE'S
... USE'S
PROC ag2(CHAN OF ANY t12, t21, t32)
 ... DECLARACION DE VARIABLES
 PRI PAR
  SEQ
   ... CREACION DE LA POBLACION INICIAL
   SEQ ven=0 FOR 10
       SEQ
         ... VALORES OBTENIDOS POR AG2
         ... EVALUACION
         ... ESTADISTICO
         ... SINCRONIZACION DE PROCESOS
         --INICIALIZACION DEL CICLO DEL ALGORITMO GENETICO HASTA QUE SE
         --CUMPLA EL NUMERO DE GENERACIONES maxgen
         SEQ generacion=0 FOR maxgen
           SEQ
             ... SELECCION, REPRODUCCION (SI EXISTE) Y CRUZAMIENTO
             ... MUTACION
             ... EVALUACION
             ... MIGRACION
             ... ESTADISTICO
         --FIN DEL ALGORITMO GENETICO
         ... COMPARA OPTIMO DE AG1 CON EL PROPIO Y ENVIA A AGM
  SKIP
:
```

--BUFFER.OCC

```
PROC buffer( CHAN OF ANY une,out,tdsp,dspl, avisa1)
 ... DECLARACION DE VARIABLES
 PRI PAR
  SEQ
   ... INICIALIZACION DEL DSP
   SEQ ven=0 FOR 10
    SEQ
      ... ESPERA DATOS DE GAM
      ... MANDA DATOS AL DSP
```

```
      ... RECIBE DATOS DEL DSP
      ... MANDA DATOS AL MONITOR
   SKIP
:


/* FFTPSD.C */
#include "stdlib.h"
extern int  fft_rl(int N,int  M,float *data);  /* FUNCION QUE REALIZA LA FFT */
extern int begin(void);                        /* FUNCION QUE INICIALIZA EL PUERTO SERIE */
extern int recibe(void);            /* FUNCION QUE RECIBE P NUMEROS DE PUNTO FLOTANTE */
extern int manda(void);             /* FUNCION QUE MANDA P NUMEROS DE PUNTO FLOTANTE  */

float data_input[512];              /* ARREGLO DE DATOS CON LOS QUE SE TRABAJA       */
int P;                              /* NUMERO DE DATOS QUE SE RECIBEN Y MANDAN       */

float cxx[11];
float var;
int ven;

                              /*************************************************/
                             /********* INICIO DEL PROGRAMA **********/
                              /*************************************************/
main()
{
int  N;
int  M;
int k,j,l;
float *direc;
float temp1,temp2;
N=256; /* tamaño de la ventana de datos */
M=6;  /*log base 2 de N*/
data_input[0]=1.0;
for(l=1;l<(2*N);l++)  /* coloca ceros a los datos */
  data_input[l]=0.0;
begin();
  for(ven=0;ven<40;ven++)
  {
     P=7;
     recibe();              /*RECIBE P DATOS EN data_input */
     for(l=1;l<=P;l++)
          cxx[l-1]=data_input[l];
     var=cxx[0];
     data_input[P]=0.0;
     P=6;
     recibe();          /* P DATOS EN data_input[1] EN ADELANTE    */
     for(l=1;l<=P;l++)     /* OBTENCION DE LA VARIANZA DE RUIDO BLANCO */
          var=var+data_input[l]*cxx[l];

     /****BIT REVERSE ****/
     j=0;
     for(l=1;l<N-1;l++)
     {
          k=128;       /* N/2 */
          while(k<=j)
```

```
            {
                j=j-k;
                k=k/2;
            }
            j=j+k;
            if(i<j)
            {
                temp1=data_input[j];
                data_input[j]=data_input[i];
                data_input[i]=temp1;
            }
        } /***FIN DEL BITREVERSE****/

        *direc=0;
        l = 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;
}
```

/* FFT.ASM */

```
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

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

        LDI     0A97h, R0
        STI     R0, @INPUT          ;DIRECCION 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
```

--CONFIGU.PGM

```
VAL K IS 1024:
VAL M IS K*K:
NODE    monitor.p, agm.p, ag1.p, ag2.p, buffer.p:
ARC     hostlink:

--UTILIZADOS POR LA COMUNICACION CON EL DSP
EDGE adt102:
ARC adtlink:

--DESCRIPCION DE LOS PROCESADORES, SU CONFIGURACION FISICA Y SUS
--CARACTERISTICAS
NETWORK first
 DO
   SET monitor.p(type, memsize:="T805",M)
   SET agm.p(type, memsize:="T805",M)
   SET ag1.p(type, memsize:="T805",M)
   SET ag2.p(type, memsize:="T805",M)
   SET buffer.p(type, memsize:="T805",M)

   CONNECT monitor.p[link][0] TO HOST WITH hostlink
   CONNECT monitor.p[link][2] TO gamoni.p[link][1]
   CONNECT gamoni.p[link][2] TO ga.p[link][1]
   CONNECT ag2.p[link][2] TO ag1.p[link][1]
   CONNECT ag1.p[link][0] TO agm.p[link][0]
   CONNECT agm.p[link][3] TO buffer.p[link][3]
   CONNECT buffer.p[link][0] TO monitor.p[link][3]
   CONNECT buffer.p[link][2] TO adt102 WITH adtlink
:

--MAPEA LA CONFIGURACION FISICA A LA CONFIGURACION LOGICA.
--SEÑALA LOS CANALES A UTILIZAR POR EL DSP
NODE    monitor.l, agm.l, ag1.l, ag2.l, buffer.l:

CHAN OF ANY tdsp, dspl:

MAPPING
 DO
   MAP monitor.l ONTO monitor.p
   MAP agm.l ONTO agm.p
   MAP ag1.l ONTO ag1.p
   MAP ag2.l ONTO ag2.p
   MAP buffer.l ONTO buffer.p
   MAP tdsp, dspl ONTO adtlink
:

--ARCHIVOS A UTILIZAR EN EL DESARROLLO DEL PROGRAMA
#INCLUDE    "hostio.inc"
#USE        "monitor.lku"
#USE        "agm.lku"
```

```
#USE     "ag1.lku"
#USE     "ag2.lku"
#USE     "buffer.lku"

--CONFIGURACION LOGICA.  ASIGNA PROCESOS A LOS PROCESADORES, ASI COMO
--MUESTRA LOS CANALES A UTILIZAR
CONFIG
 CHAN OF SP fs:
 CHAN OF SP ts:
 PLACE fs, ts ON hostlink:

 CHAN OF ANY in:
 CHAN OF ANY t12, t13, une, avisa:
 CHAN OF ANY t21:
 CHAN OF ANY t31, t32:
 CHAN OF ANY out, avisa1:
 PLACED PAR
  PROCESSOR monitor.t
   monitor(fs, ts, in, out, avisa)
  PROCESSOR agm.t
   agm(in, t12, t13, t21, t31, une, avisa, avisa1)
  PROCESSOR ag2.t
   ag2(t12, t21, t32)
  PROCESSOR ag1.t
   ag1(t31, t32, t13)
  PROCESSOR buffer.t
   buffer(une, out, tdsp, dspt, avisa1)
:
```

A.2. ESQUEMA HETEROGENEO CON EL NPH EN EL CALCULO DE LOS PARAMETROS.

Este grupo de programas utiliza el Nodo Heterogéneo en el cálculo de los parámetros, para lo cual, es necesario los programas en OCCAM MONITOR.OCC, AGM.OCC, AG1.OCC y AG2.OCC para los transputers, así como los programas FOBJ.C en C para el DSP. Finalmente el programa CONFIGU.PGM para configurar la red de la figura 7.5.

--MONITOR.OCC

```
... INCLUDE'S
... USE'S
PROC monitor(CHAN OF SP fs,ts, CHAN OF ANY in, out)
 ... DECLARACION DE VARIABLES
 SEQ
  error:=TRUE
  WHILE error
   SEQ
        ... LECTURA DE PARAMETROS
  ... CALCULO DE LA MATRIZ DE COVARIANCIA
```

```
    ... ENVIO Y RECEPCION DE DATOS
    ... DESPLIEGUE DE RESULTADOS
:
```

--AGM.OCC

```
... INCLUDE'S
... USE'S
PROC agm(CHAN OF ANY In, t12, t13, t21, t31, une, avisa, avisa1)
  ... DECLARACION DE VARIABLES
  PRI PAR
    SEQ
      ... CREACION DE LA POBLACION INICIAL
      SEQ ven=0 FOR 10
          SEQ
            ... RECIBE DATOS DE MONITOR Y MANDA A AG1
            ... EVALUACION (AQUI Y MANDA INDIVIDUOS AL DSP VIA AG1)
            ... PROCESO ESTADISTICO
            ... SINCRONIZACION DE PROCESOS
            --INICIALIZACION DEL CICLO DEL ALGORITMO GENETICO HASTA QUE SE
            --CUMPLA EL NUMERO DE GENERACIONES maxgen
            SEQ generacion=0 FOR maxgen
              SEQ
                ... SELECCION, REPRODUCCION (SI EXISTE) Y CRUZAMIENTO
                ... MUTACION
                ... EVALUACION (AQUI Y MANDA INDIVIDUOS AL DSP VIA AG1)
                ... MIGRACION
                ... ESTADISTICO
            --FIN DEL ALGORITMO GENETICO
            ... SELECCIONA OPTIMO
    SKIP
:
```

--AG1.OCC

```
... INCLUDE'S
... USE'S

PROC ag1(CHAN OF ANY t31, t32, t13, t23, t3dsp, dspt3)
  ... DECLARACION DE VARIABLES
  PRI PAR
    SEQ
      ... CREACION DE LA POBLACION INICIAL
      SEQ ven=0 FOR 10
          SEQ
            ... VALORES OBTENIDOS DE AGM
            ... VALORES MANDADOS A AG2 Y AL TMS320C30
            ... EVALUACION (AQUI Y MANDA INDIVIDUOS DE AGM, AG1 Y AG2 AL DSP)
            ... ESTADISTICO
            ... SINCRONIZACION DE PROCESOS
            IF
              ven=0
                ... INICIALIZA EL DSP
              TRUE
```

```
          SKIP
        --INICIALIZACION DEL CICLO DEL ALGORITMO GENETICO HASTA QUE SE
        --CUMPLA EL NUMERO DE GENERACIONES maxgen
        SEQ generacion=0 FOR maxgen
          SEQ
            ... SELECCION, REPRODUCCION (SI EXISTE) Y CRUZAMIENTO
            ... MUTACION
            ... EVALUACION (AQUI Y MANDA INDIVIDUOS DE AGM, AG1 Y AG2 AL DSP)
            ... MIGRACION
            ... ESTADISTICO
        --FIN DEL ALGORITMO GENETICO
        ... MANDA OPTIMO A AG2
  SKIP
:
```

--AG2.OCC

```
... INCLUDE'S
... USE'S
PROC ag2(CHAN OF ANY t12, t21, t32, t23)
  ... DECLARACION DE VARIABLES
  PRI PAR
    SEQ
      ... CREACION DE LA POBLACION INICIAL
      SEQ ven=0 FOR 10
          SEQ
            ... VALORES OBTENIDOS DE AG1
            ... EVALUACION (AQUI Y MANDA INDIVIDUOS AL DSP VIA AG1)
            ... ESTADISTICO
            ... SINCRONIZACION DE PROCESOS
            --INICIALIZACION DEL CICLO DEL ALGORITMO GENETICO HASTA QUE SE
            --CUMPLA EL NUMERO DE GENERACIONES maxgen
            SEQ generacion=0 FOR maxgen
              SEQ
                ... SELECCION, REPRODUCCION (SI EXISTE) Y CRUZAMIENTO
                ... MUTACION
                ... EVALUACION (AQUI Y MANDA INDIVIDUOS AL DSP VIA AG1)
                ... MIGRACION
                ... ESTADISTICO
            --FIN DEL ALGORITMO GENETICO
            ... COMPARA MEJOR INDIVIDUO DE AG1 Y AG2 Y MANDA A AGM
    SKIP
:
```

/* FOBJ.C */

```
#include "math.h"
/* FUNCIONES A UTILIZAR PARA LA COMUNICACION CON EL T805 */
extern int begin(void);
extern int recibe(void);
extern int manda(void);

/* DEFINICION DE VARIABLES GLOBALES */
float c[7][7]; /* tamaño de la matriz de covariancia
```

```
float fitness;
float a[49];
int P,N,IND;

                                /*******************************/
                                /* INICIO DEL PROGRAMA */
                                /******************************* */
main()
{
 /* DEFINICION DE VARIABLES LOCALES */
 float static suma1, suma2, suma3, cmax;
 int static i,j,k,l,n;
 suma1=0;
 begin();
 N = 256; /* tamaño de elementos de la fft */
 IND = 300;  /* numero de individuos a evaluar en total */

 /* RECIBE LA MATRIZ DE COVARIANCIA MODIFICADA */
 P=49;  /* numero de datos a recibir por llamada a recibe() */
 recibe();
 k=0;
 for (i=0;i<11;i++)
  {
    for (j=0;j<11;j++)
     {
       c[i][j]=a[k];
       k++;
     }
  }
 /* CALCULO DEL ERROR DE UN INDIVIDUO */
 P=6;  /* numero de datos a recibir por llamada a recibe() */
 cmax=0.002;
 for (n=0;n<IND;n++)
  {
    recibe();
    suma1=0; suma2=0; suma3=0;
    for(i=0;i<P;i++)
      suma1+=a[i]*a[i]*c[i+1][i+1];
    for(i=0;i<P;i++)
      suma2+=a[i]*c[0][i+1];
    for(i=0;i<P-1;i++)
      for(j=i+1;j<P;j++)
    suma3+=a[i]*a[j]*c[i+1][j+1];
    fitness = c[0][0] + suma1 + 2*suma2 + 2*suma3;
    if (fitness<0.0)
      fitness=cmax;
    fitness=1-(fitness/cmax);
    manda();
  }
P=0;
}
```

--CONFIGU.PGM

```
VAL K IS 1024:
VAL M IS K*K:
NODE     monitor.p, agm.p, ag1.p, ag2.p:
ARC      hostlink:

--VARIABLES UTILIZADAS POR LA COMUNICACION CON EL DSP
EDGE adt102:
ARC adtlink:

--DESCRIPCION DE LOS PROCESADORES, SU CONFIGURACION FISICA Y SUS
--CARACTERISTICAS
NETWORK first
  DO
    SET monitor.p(type, memsize:="T805",M)
    SET agm.p(type, memsize:="T805",M)
    SET ag2.p(type, memsize:="T805",M)
    SET ag1.p(type, memsize:="T805",M)

    CONNECT monitor.p[link][0] TO HOST WITH hostlink
    CONNECT monitor.p[link][2] TO agm.p[link][1]
    CONNECT agm.p[link][2] TO ag2.p[link][1]
    CONNECT ag2.p[link][3] TO ag1.p[link][3]
    CONNECT ag1.p[link][0] TO agm.p[link][0]
    CONNECT ag1.p[link][2] TO adt102 WITH adtlink
:

--MAPEA LA CONFIGURACION FISICA A LA CONFIGURACION LOGICA.
--SEÑALA LOS CANALES A UTILIZAR POR EL DSP
NODE     monitor.l, agm.l, ag1.l, ag2.l:
CHAN OF ANY t3dsp, dspt3:
MAPPING
  DO
    MAP monitor.l ONTO monitor.p
    MAP agm.l ONTO agm.p
    MAP ag1.l ONTO ag1.p
    MAP ag2.l ONTO ag2.p
    MAP t3dsp, dspt3 ONTO adtlink
:

--ARCHIVOS A UTILIZAR EN EL DESARROLLO DEL PROGRAMA
#INCLUDE    "hostio.inc"
#USE        "monitor.lku"
#USE        "agm.lku"
#USE      "ag1.lku"
#USE      "ag2.lku"

--CONFIGURACION LOGICA.  ASIGNA PROCESOS A LOS PROCESADORES, ASI COMO
--MUESTRA LOS CANALES A UTILIZAR POR CADA UNO DE ELLOS
CONFIG
  CHAN OF SP fs, ts:
  PLACE fs, ts ON hostlink:
  CHAN OF ANY in:
  CHAN OF ANY t12, t13, out:
  CHAN OF ANY t21:
```

```
CHAN OF ANY t31, t32:
PLACED PAR
  PROCESSOR monitor.I
    monitor(fs, ts, in, out)
  PROCESSOR agm.I
    agm(in, t12, t13, t21, t31, out)
  PROCESSOR ag2.I
    ag2(t12, t21, t32)
  PROCESSOR ag1.I
    ag1(t31, t32, t13, t3dsp, dspt3)
:
```

APENDICE B

PROGRAMAS DE COMUNICACION T805-TMS320C30

El conjunto de programas que se presentan a continuación son utilizados para la comunicacion del DSP TMS320C30 con el transputer T805. Dichos programas, realizados en lenguaje ensamblador, son utilizados como funciones dentro de un programa central realizado en C para el mismo procesador. Los programas utilizados son BEGIN.ASM que inicializa el puerto serie y el timer del DSP así como la comunicación con el transputer, RECIBE.ASM que recibe un número *P* de datos definido en el programa en C como variable global y MANDA.ASM que manda un número *P* de datos de la misma forma que en la función RECIBE.ASM. Los datos recibidos y mandados se encuentran alojados dentro del vector data_input también declarada como global dentro del programa en C. Los datos recibidos y mandados son almacenados en la dirección de memoria definida en las líneas marcadas con el comentario:

;MODIFIED ADDRESS

en donde la instrucción es:

LDI XXXXh , R0

ésta direccion puede ser la de data_input[0] o cualquier otra que se requiera según el problema que se esté tratando.

Si se tiene alguna duda de qué dirección utilizar es posible conocer la dirección de memoria de data_input[0] en el archivo ARCHIVO.MAP asociado al archivo ARCHIVO.C después de compilar el paquete de programas necesarios para la ejecución del programa principal.

El programa original (del cual se implementaron las funciones) fue elaborado por Daniela Ramos en su tesis "Diseño e Implementación de un Nodo Heterogéneo de Procesamiento Digital de Señales Utilizando un Transputer y un DSP".

Para realizar las llamadas a las funciones es necesario declararlas en el programa en C de la siguiente forma:

```
extern int begin(void);
extern int recibe(void);
extern int manda(void);
```

BEGIN.ASM

```
                .title   "begin"
FP              .set     AR3
                .global _begin
                .global _main
;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    _begin
                .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:**********************
;***********************************************
_begin:         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
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
                pop FP
                RETS
```

RECIBE.ASM

```
                .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
                ;MODIFIED ADDRESS
                ldi     0A98h, AR0      ;address to DATA_INPUT[1]
                ldi     @serial_1, AR1  ;address to serial port
                ldi     @timer_1, AR2   ;address to timer
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(dm), 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(dm), 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(dm), 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(dm), 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
                ;MODIFIED ADDRESS
                ldi     0A98h, AR0          ;address to DATA_INPUT[1]
                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
                idiz    *+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
                ;DIRECCION A MODIFICAR
                ldi     0A98h, AR0          ;address to DATA_INPUT[1]
                pop     FP
```

```
                RETS
```

MANDA.ASM

```
                .title   "MANDA"
FP              .set     AR3
                .global _manda
                .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    _manda
                .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:**********
;*********************************************

_manda:         push     FP
                ldi      SP, FP
                ;MODIFIED ADDRESS
                ldi      0A97h, AR0            ;address to DATA_INPUT[0]
```

```
                ldi     @serial_1, AR1          ;address to serial port
                ldi     @timer_1, AR2           ;address to timer
;TMS320C30 to IEEE Floating-Point Format Coversion
                ;MODIFIED ADDRESS
                ldi     0A97h, AR0              ;address to DATA_INPUT[0]
                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
                ;MODIFIED ADDRESS
                ldi     0A97h, AR0              ;address to DATA_INPUT[0]
                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             ;number of parameters
                cmpi    IR0, R6
                bnz     inicia_tx
                pop     FP
                RETS
```