UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DISEÑO DE UNA ARQUITECTURA PARA PROCESAMIENTO PARALELO DE SEÑALES DOPPLER DE ULTRASONIDO

T E S I S

Que para obtener el Título de:

INGENIERO MECANICO ELECTRICISTA

p r e s e n t a

HECTOR BENITEZ PEREZ

Director de Tesis Dr. Fabian Garcia Nocetti

México, D. F. *1994*

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

A MIS PADRES PORQUE FUERON
EL INICIO DE TODO

CONTENIDO

CONTENIDO

CONTENIDO

CAPITULO I

INTRODUCCION

1.1 GENERALIDADES

Los sistemas convecionales de cómputo operan en forma secuencial, en donde las instrucciones de un programa son ejecutadas una a la vez. Esta característica ha sido forzada por la arquitectura secuencial de computadoras convencionales (arquitecura Von Neumann), en la cual un procesador central es conectado a un banco de memoria por medio de un bus. Desde entonces la mayoría de sucesivas generaciones de computadoras disponibles han seguido este diseño. Sin embargo, una gran variedad de problemas asociados a las áreas de control, visión, procesamiento de voz, imágenes y pocesamiento digital de señales poseen un paralelismo intrínseco. De hecho el resolver este tipo de problemas en forma secuencial ha sido en gran número de casos computacionalmente intensivo y restrictivo, sobre todo cuando se trata con aplicaciones en tiempo real, en donde se manejan intervalos de muestreo muy cortos del orden de milisegundos.

Un caso típico en el área de procesamiento de señales e ingeniería de control es la implantación de algoritmos de estimación espectral en tiempo real, en donde el cálculo de los estimadores utilizados debe ser realizado dentro de un intervalo de muestreo típico del orden de 5-10 milisegundos. Aun considerando el poder de cómputo de modernos procesadores secuenciales convencionales, ésto puede ser difícil de alcanzar, sobre todo cuando se manejan múltiples variables. Claramente, a mayor complejidad de los algoritmos corresponde una mayor dificultad en el problema de realizar los cálculos necesarios en tiempo-real. Aplicaciones diferentes imponen demandas variables. De este modo, ciertas aplicaciones reales están sobrepasando los límites de desempaño de arquitecturas convencionales. Aunque el uso de la tecnología de integración ha traido como resultado un incremento en la velocidad de los procesadores y en consecuencia de los sistemas de cómputo, la tecnología no ha logrado incrementar la velocidad en la misma proporción que el nivel de integración. La velocidad máxima a la que las componentes electrónicas operan ha marcado un límite en el diseño de procesadores más veloces. La alternativa ha sido entonces modificar la arquitectura típica de los sistemas de cómputo. Esto ha estimulado la investigaión y el uso de arquitecturas alternativas que satisfagan las nuevas demandas computacionales da una manera efectiva y práctica. Procesamiento paralelo ha sido una de las alternativas mas viables.

La disponibilidad actual de arquitecturas de procesamiento paralelo, que permiten distribuir tanto algoritmos como información, sobre un número de procesadores, ha creado nuevas oportunidades para el diseño e implantación de sistemas más rápidos y complejos. Procesamiento paralelo está siendo cada

véz más atractivo como un medio para construir sistemas de alto desempeño y confiabilidad. Su buen desempeño es debido a el uso de múltiples procesadores y eficientes redes de interconexión, mientras que su alta confiabilidad es debida a su inherente redundancia.

La tecnología de cómputo paralelo de hecho ha tenido un gran impacto en el diseño e implantación de sistemas de tiempo real asociados a diversos tipos de problemas. Dicha tecnología ha expandido el dominio factible de sistemas que operan en tiempo real y sistemas integrados, donde las limitaciones de procesamiento han restringido tradicionalmente sus capacidades.

En particular la introducción del transputer y su lenguaje asociado OCCAM, para el soporte de procesamiento paralelo, ha tenido un gran impacto en un número de aplicaciones, permitiendo el diseño y construcción de sistemas más rápidos y complejos, de una forma simple y estructurada. El transputer y OCCAM, han sido diseñados específicamente para ser usados en sistemas de múltiples procesadores , y diversas topologías, ofreciendo el prospecto de un desempeño escalable a medida que más procesadores son agredados al sistema, siendo éste un factor determinante para su utilización en una creciente variedad de aplicaciones.

En instrumentación médica Doppler, los instrumentos hasta ahora disponibles hacen uso del algoritmo FFT para calcular el espectro de la señal Doppler generada por el flujo sanguíneo , extrayendo de este modo la información diagnóstica cuantitativa. Sin embargo, con esta técnica es difícil distinguir ciertos problemas en estado inicial debido a su resolución en frecuencia que influye en la abilidad de distinguir las respuestas espectrales de dos o mas señales, otra limitación es el ventaneo implicito en el análisis de la señal. Recientes trabajos de investigación han conducido a la conclusión de que los métodos paramétricos de estimación espectral ofrecen una importante incremento en la resolución de frecuencia . Estudios previos realizados para investigar el desempeño de un número de estimadores espectrales en señales Doppler, han identificado el método paramétrico AutoRegresivo denominado Método de Covariancia modificada como costo-efectivo. El costo está asociado a su complejidad computacional y la efectividad del algoritmoestá asociada a que es suceptible de paralelización.

En este trabajo se presenta el diseño de un sistema de procesamiento paralelo basado en la arquitectura del transputer y en el concepto de modularidad llamado TRAM. Este sistema esta programado en lenguaje OCCAM y es utilizado en la implantación de algoritmos de estimación espectral para señales Doppler de ultrasonido . En particular 2 métodos espectrales han sido implantados. El primero es un método no paramétrico basado en la FFT. El segundo es un método basado en el algoritmo de Covariancia Modificada. Ambos métodos son evaluados con respecto a la velocidad de ejecución , eficiencia y resolución, al ser implantados en un sistema de procesamiento paralelo.

1.2 OBJETIVOS

1.- Diseño y desarrollo de un sistema de procesamiento paralelo basado en la arquitectura del transputer y en el concepto de modularidad llamado TRAM, para ser utilizado en el procesamiento de señales Doppler en ultrasonido.

2.- Desarrollo e implantación en paralelo de técnicas de estimación espectral basadas en métodos paramétricos y no paramétricos

3.- Estudio de desempeño del sistema de procesamiento paralelo mediante el análisis de los resultados cualitativos y cuantitativos de las técnicas de estimación espectral utilizadas.

1.3 CONTENIDO

Resta señalar el orden en el cual es planteada esta tesis para darle así un sentido lógico en su desarrollo.

En el Capítulo 1 se da una revisión de las generalidades de Procesamiento Paralelo y en particular del impacto de esta teconología en el procesamiento de señales.

El Capítulo 2 trata sobre una serie de conceptos generales acerca del paralelismo, los cuales se relacionan con la plataforma ha desarrollarse y el procesador a usar. Por lo que después de presentar estos conceptos se ubica al sistema a partir de ellos.

El Capítulo 3 presenta las características del transputer y del lenguaje OCCAM, además se muestra una serie de ventajas para el desarrollo del sistema en base al transputer.

El Capítulo 4 trata acerca del diseño, desarrollo e implantación del sistema de procesamiento paralelo.

En el Capítulo 5 se presenta un caso de estudio relacionado con instrumentación médica Doppler, el cual ha sido utilizado para ilustrar la aplicación del sistema de procesamiento paralelo en un problema práctico.

El Capítulo 6 expone las conclusiones generales de este trabajo.

El Apéndice A muestra una descripción general del sistema de desarrollo para transputers utilizado en el curso de este trabajo.

En el Apéndice B se describen los programas de prueba utilizados para validar el desarrollo del sistema de procesamiento paralelo diseñado.

Por último el Apéndice C muestra los programas de aplicación desarrollados para implantar algorítmos de estimación espectral en el sistema diseñado.

CAPITULO II

GENERALIDADES DE PROCESAMIENTO PARALELO

2.1 GENERALIDADES

Dado el avance que se ha tenido en la búsqueda de computadoras más rápidas y más económicas; se han desarrollado nuevas técnicas y caminos para alcanzar este objetivo. En base a este desarrollo, existe un campo extenso que es la evolución lógica de la computación y el paralelismo. Su desarrollo esta basado desde los años 50; pero no es, sino hasta la década de los 90, que se ha tenido un impulso comercial y al parecer definitivo de este tipo de arquitecturas.

Tres son los campos en los que se puede dividir al procesamiento paralelo :algoritmos, lenguajes de programación y arquitecturas[2].

La definición más lógica que podemos dar de paralelismo, sería respecto a aquellas máquinas que no cumplen con el paradigma de Von Neumann; es decir; que sus procesos contrastan con el procesamiento secuencial, en el cual una sola unidad de procesamiento es conectada a una sola unidad de memoria[2].

Tomando como concepto preliminar de paralelismo a la capacidad para traslapar o procesar simultáneamente un grupo de tareas; podemos decir que, los principales caminos que se introdujeron en el paralelismo en base a las arquitecturas propuestas son los siguientes :

Pipeline - La aplicación de técnicas de líneas de ensamblado que proveé el rendimiento de una unidad aritmética o de control generando un traslape en ciertas etapas de la ejecución de instrucciones.

Funcional - Proveé algunas unidades independientes para el desarrollo de diferentes funciones, como lógicas, multiplicación o división y cualquier manejo de operandos simultáneamente en diferentes tipos de datos.

Arreglo - Proveé un arreglo idéntico de elementos de procesamiento sobre un control común , todo el rendimiento está dado sobre las mismas operaciones simultáneamente pero en diferentes datos almacenados en memorias privadas.

Multiprocesamiento - Es provisto de algunos procesadores, cada uno trabajando sobre sus propias instrucciones, y usualmente comunicado vía una memoria común o bus común de comunicación (memoria distribuida o memoria compartida y por lo común sistemas MIMD).

Por supuesto , cada diseño se puede combinar en alguna o todas las características que se acaban de señalar . Por ejemplo un arreglo de procesadores puede tener una unidad aritmética pipeline con sus elementos de procesamiento y una unidad funcional en una unidad multiunidad que puede ser un arreglo de procesadores.

Dentro de una remembranza del paralelismo y tomando en cuenta el desarrollo que se tuvo a partir de las unidades aritméticas de pipeline, podemos dar la Figura 2.1 como parte del desarrollo que se tuvo a partir de UNIVAC1 en 1951 hasta aquellas supercomputadoras como la CRAY-1[2].

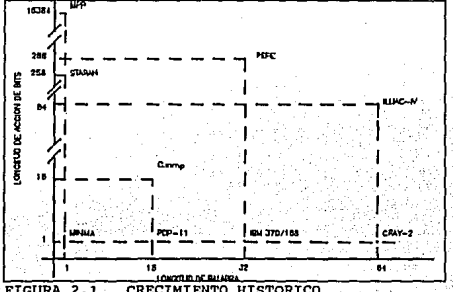


FIGURA 2.1 CRECIMIENTO HISTORICO DEL PROCESAMIENTO PARALELO

En este esquema encontramos el desarrollo de diferentes tipos de arquitecturas según la técnica aplicada para su crecimiento como son los arreglos de procesadores o las técnicas de pipeline. Cabe señalar que ninguna técnica de diseño está totalmente divorciada de la otra, pues, como se muestra en el esquema hay máquinas que pueden compartir estas características sin llegar a dañar el concepto esencial del paralelismo.

Dentro del contexto del esquema 2.1 podemos señalar que la historia muestra al paralelismo usado desde los diseños más primitivos de computadoras , dando así, distintos niveles de paralelismo según sea el caso de avance:

1.- NIVEL DE TRABAJO
    i) ENTRE LAS FASES DE UN TRABAJO
2.- NIVEL DE PROGRAMA
    i) ENTRE LAS PARTES DE UN PROGRAMA
3.- NIVEL DE INSTRUCCION
4.- NIVEL DE BIT Y ARITMETICO
    i) ENTRE LOS ELEMENTOS DE UN VECTOR O UN ESCALAR

Si tomamos en cuenta que el más alto nivel que una computadora debe de tener es la relación del procesamiento de trabajos, veremos que el estado del trabajo a desarrollar puede tener diferentes caminos o alternativas, de las cuales encontramos el paralelismo como una manera de desarrollo de aplicaciones complejas desde el punto de vista de la división de trabajos para la implementación de diferentes fases del mismo en un tiempo similar al de cada una de las fases. El desarrollo de cualquier trabajo tendrá como base el diseño de un programa por lo tanto su estructuración ,paralelización y vectorización se hará dentro de las medidas posibles de los recursos de cada máquina; lo que permitirá aumentar la eficiencia de esta última. A su vez, para el desarrollo del paralelismo y su propia eficiencia, tenemos que, cada instrucción debe poder estar pensada para un ambiente de paralelismo, pues, será muy difícil plantear al paralelismo como un sistema eficiente si los recursos que se ofrecen no son los óptimos para este tipo de desarrollos.

2.2 MEDICION DE RENDIMIENTO

Hay muchos caminos para medir el rendimiento de algoritmos paralelos corriendo en sistemas de procesamiento paralelo. Sin embargo no hay una sola métrica para estimar de forma absoluta el rendimiento. Es posible comparar el rendimiento de dos computadoras para resolver el mismo problema, pero los resultados de esta comparación pueden variar de una manera muy fuerte dependiendo del algoritmo y del tamaño del problema. Esta sola comparación no forma una adecuada base para evaluar el rendimiento relativo de aquellos sistemas con alguna otra aplicación.

2.2.1 RELACION R/C

Los beneficios del rendimiento en el uso de multiprocesadores depende de la relación R/C^1, donde R es el tiempo de ejecución en una tarea cualquiera y C es el tiempo utilizado para las comunicaciones de ésta misma. La relación expresa que tanto tiempo de espera es usado por unidad de cálculo. Cuando la relación es muy baja, llega a ser ineficiente el uso del paralelismo. Cuando la relación es muy alta, el paralelismo es muy útil. La relación pequeña da un paralelismo pobre porque se presenta un gran volumen de tiempo de espera en las comunicaciones. Sin embargo para relaciones extremadamente altas usualmente se refleja una pobre explotación del paralelismo. Por lo que, para un máximo rendimiento, es necesario balancear al paralelismo con los tiempos de espera. La relación R/C es una medición de la granularidad de las tareas[4]. Para una granularidad gruesa,

la relación R/C es relativamente alta, es decir, cada unidad de cálculo tiene un uso relativamente pequeño de las comunicaciones. En la granularidad fina, R/C es bajo , hay un gran número de comunicaciones por unidad de cálculo. Obteniendo a las tareas ejecutadas en la forma de granularidad gruesa con grandes volúmenes de cálculos a realizar y pequeñas pérdidas de tiempo en la comunicación. Por el contrario, las tareas que siguen el concepto de granularidad fina requieren de un gran volumen de comunicaciones para interactuar.

2.2.2 GRADO DE PARALELISMO

Si decimos que, el número máximo de bits que pueden ser procesados en una unidad de tiempo por un sistema computador se denomina máximo grado de paralelismo P[3]. Sea Pi el número de bits que pueden ser procesados por el iésimo ciclo de procesador considerando T ciclos de procesador indicados por i=1,2,...,T. Podemos decir que, el grado medio de paralelismo Pm esta dado por:

$$P=\frac{\sum_{T}^{i=1} P_i}{T}$$

En general Pi <= P. Por tanto, definimos la tasa de utilización u de un sistema durante T ciclos como:

$$u=\frac{P_m}{P_i}=\frac{\sum_{T}^{i=1} P_i}{T*P}$$

De lo cual podemos decir, que si la potencia del procesador esta totalmente utilizada tenemos Pi = P. Para lo cual podemos diseñar una gráfica (Figura 2.2) basada en el concepto de ortogonalidad con lo que mostramos una serie computadoras según el máximo grado de paralelismo que pueden alcanzar[1].

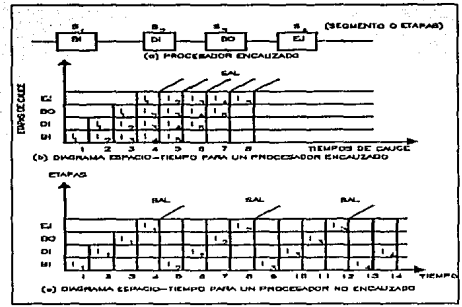


FIGURA 2.2 GRADOS DE PARALELISMO

Sobre el eje horizontal se indica la longitud n de palabra.El eje vertical corresponde a la longitud m de sección de bits.Ambas medidas de longitud vienen dadas en términos de números de bits contenidos en una palabra o en una sección de bits.Una sección de bits es una cadena de bits,uno por cada una de las palabras sobre las que se opera en paralelo. Por lo que el máximo grado de paralelismo P(C) de un sistema computador C viene representado por el producto de la longitud n de palabra y la longitud m de sección de bits(como lo mencionado en la primera parte de este capítulo); es decir,

$$P(C) = n*m$$

El P(C) es igual al área del rectángulo definido por los enteros n y m. Existen cuatro tipos de métodos de procesamiento que pueden deducirse de este diagrama':

1) PALABRA-SERIE Y BIT-SERIE (PSBS)
2) PALABRA-PARALELO Y BIT-SERIE (PPBS)
3) PALABRA-SERIE Y BIT-PARALELO (PSBP)
4) PALABRA-PARALELO Y BIT-PARALELO (PPBP)

El tipo PSBS ha sido llamado procesamiento bit-serie porque procesa un bit (n=m=1) cada vez,lo cual resulta bastante lento.Se aplicó solamente en las computadoras de la primera generación.El tipo PPBS (n=1, m>1) ha sido llamado procesamiento por sección de palabra porque procesa una sección de m bis cada vez.El PSBP (n>1, m=1),el tipo más frecuente en las computadoras,ha sido llamado por procesamiento por sección de palabra porque procesa una palabra de n bits cada vez. Finalmente, el tipo PPBP (n>1, m>1) se conoce como procesamiento totalmente paralelo y en él

se procesa una matriz de n*m bits cada vez; éste es el modo más rápido de procesamiento de los cuatro.

Cabe señalar que las arquitecturas más comunes de los transputers (como los TRAMS) pueden ser ubicadas dentro de la taza de alto paralelismo, pues, como se verá mas adelante (capítulo 3 y 4) en el manejo de topologías; los transputers permiten utilizar de una forma más funcional la comunicación entre ellos mismos.

2.2.3 EFICIENCIA

La relación de velocidad es generalmente medida al correr el mismo programa en un número de procesadores[4]. La relación de velocidad es tomada entre el tiempo necesario por el procesador 1 dividido por el tiempo necesario en p procesadores:

$$s=\frac{T[1]}{T[p]}$$

Teniendo con ésto una relación lineal con respecto al paralelismo a medir. Es decir, si un sistema A da una relación de velocidad mayor a la presentada por un sistema B , se dice que el sistema A provee un mejor soporte para la paralelización que su contraparte B.

Escalando la relación de velocidad por el número de procesadores , se obtiene una medición mas representativa del paralelismo[4]. A esta métrica se le conoce como eficiencia y es definida como

$$e=\frac{T[1]}{pT[p]}=\frac{s}{p}$$

Al igual que la relación de velocidad , la eficiencia es una medida lineal y directamente proporcional del paralelismo, con lo que , para bajos valores de eficiencia se dice que los recursos están siendo desaprovechados.

2.3 PIPELINE

Muchos de los procesadores que tienen el fundamento de funcionamiento de alta velocidad y de manejo de memoria a velocidad constante en el "pipeline" y en el arreglo de procesadores dependen en mucho, del uso del software encaminado a la optimización de algoritmos y del empleo de sus posibilidades de comunicación con otros dispositivos[3].

Dentro de este esquema de manejo de rendimiento , tenemos que, la técnica de pipeline es una de las más importantes en la reducción de tiempos y en la optimización de algoritmos[3].

Existen varios tipos de pipeline que cumplen con los objetivos determinados. El más común de ellos es el pipeline por instrucción, el segundo es el pipeline aritmético y el tercero pipeline multiproceso.

El objetivo del pipeline es el traslape de los procesamientos realizados. De esta primera definición vemos que quien cumple con esta propuesta , es el pipeline por instrucción; pues como se muestra en la Figura 2.3 la secuencia de instrucciones es ejecutada según el nivel en que se encuentre y el tiempo en que se logre esto. Para obtener lo anterior se tiene que dividir a la instrucción, en varios procedimientos (s1,s2,s3,...,sn) los cuales se ejecutarán en determinado tiempo[3]. Dado que los procedimientos de cada instrucción son independientes de cualquier otro que se este ejecutando en ese mismo tiempo, es posible llegar al traslape entre procesamientos de instrucciones. Cabe señalar que el traslape se realiza en los tres tipos de procesamiento más comunes entre los microprocesadores; búsqueda,decodificación y búsqueda de los operandos de las instrucciones.

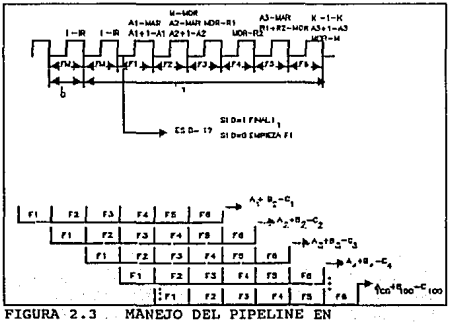


FIGURA 2.3 MANEJO DEL PIPELINE EN INSTRUCCIONES DE TIPO CONDICIONAL

Para los condicionales , tenemos que, este es un problema convencional en las instrucciones de pipeline, donde el sucesor actual (I_{2F} o I_{2T}) de la instrucción es I_1; el cual puede ser fetcheado solo con el mismo, sin tener que manejar la fase de traslape con cualquier instrucción (Figura 2.4). Sin embargo, para las instrucciones convencionales de pipeline, el traslape puede ser generado como en cualquier tipo de instrucción.

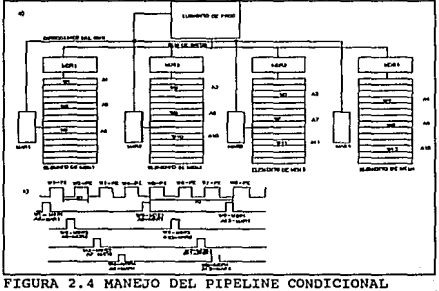


FIGURA 2.4 MANEJO DEL PIPELINE CONDICIONAL

Para el manejo de memoria en el pipeline por instrucción ; tenemos que, el acceso de datos a la memoria, por diferentes puertos de datos, se puede llegar por medio de vectores concurrentes como lo muestra la Figura 2.5[1]. Para el caso de una memoria no concurrente y de un mismo puerto, tenemos que, el pipeline se realizará en un tiempo de retardo con respecto a la operación en ejecución.

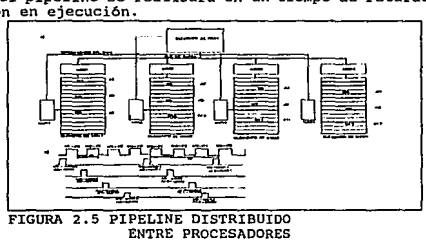


FIGURA 2.5 PIPELINE DISTRIBUIDO
ENTRE PROCESADORES

Además del pipeline por instrucción; tenemos al pipeline aritmético; el cual fragmenta la operación aritmética en fases dentro de la instrucción . La operación puede representar :

-Una secuencia que puede ser hecha de una serie de operaciones aritméticas simples.
-Una sola operación representada por un algoritmo computacional por pasos. (multiplicación , división y operaciones de punto flotante).

Las instrucciones de pipeline se fragmentan en fases de acceso de memoria. Desde el acceso a memoria mismo pasando por el ciclo de fetch hasta la ejecución de la misma operación.

Hay dos tipos de pipeline aritmético básicos[1] :

El pipeline uniproceso, en el cual la ejecución de una o de algunas instrucciones se realiza por medio de un solo procesador

El pipeline multiproceso del cual la ejecución de algunas operaciones aritméticas se realiza por medio de una secuencia de procesadores que trabajarán según el estado de pipeline en que se encuentren.

Cabe señalar que existe una contraparte al desarrollo del pipeline; conocido como arreglo de procesadores, el cual tiene como principal característica al desarrollo de la misma operación sobre un set de diferente pares de datos[1]. Mientras que al pipeline lo podemos definir como trabajo realizado sobre diferentes fases de la misma operación las cuales son puestas en paralelo sobre un set de diferentes pares de datos. Figura 2.6 y Figura 2.7 respectivamente[1].

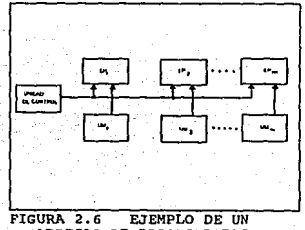


FIGURA 2.6 EJEMPLO DE UN ARREGLO DE PROCESADORES

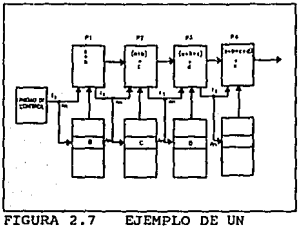


FIGURA 2.7 EJEMPLO DE UN SISTEMA DE PIPELINE EN UN ARREGLO DE PROCESADORES

En un arreglo de procesadores , una operación atribuida a una instrucción es ejecutada en paralelo o síncronamente sobre un arreglo (vector) de operandos de datos . El rendimiento de un sistema de este tipo es medido en la velocidad de operación para un par de datos (Ai,Bi) (Figura 2.4). En un sistema de pipeline, el desarrollo de la fase de paralelismo por medio del cual un sistema de este tipo de n estados ejecuta n diferentes operaciones en un tiempo , cada uno de los cuales es desarrollado sobre un par separado de operandos y es dado en la iésima fase de ejecución donde i es el número de estados de pipeline (Figura 2.5).

La mayor distinción entre los arreglos de procesadores y el pipeline', es que cada elemento de procesamiento en un arreglo ejecuta todas las fases de una instrucción microprogramada localmente, mientras que,en el pipeline, cada elemento del procesamiento ejecuta solo una fase de la instrucción microprogramada; la siguiente fase es ejecutada por el siguiente elemento de procesamiento; como se muestra en la figura 2.8. En la cual se ilustra los diferentes caminos en el rendimiento de una operación aritmética en una arquitectura serial , pipeline y arreglo de procesadores.

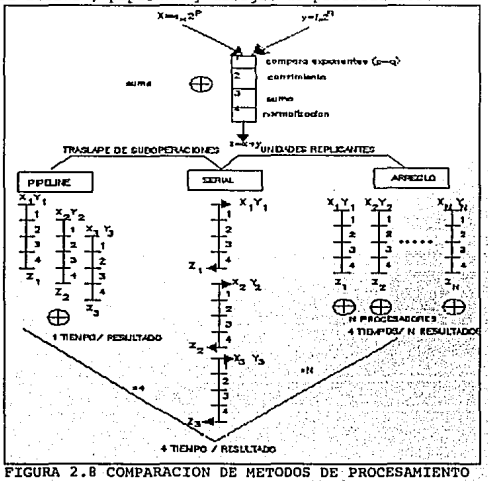


FIGURA 2.8 COMPARACION DE METODOS DE PROCESAMIENTO

2.4 ARREGLO DE PROCESADORES

Entendiendo el concepto expuesto anteriormente acerca del arreglo de procesadores , podemos dar ahora una serie de características que se tendrán en un arreglo de procesadores:

1) UN NUMERO DE PROCESADORES
2) UN NUMERO DE BANCOS DE MEMORIA
3) ALGUNAS FORMAS DE COMUNICACION EN RED
4) ALGUNAS FORMAS DE COMUNICACION LOCAL
5) ALGUNAS FORMAS DE CONTROL GLOBAL

Una organización típica de un arreglo de procesadores es dada en las Figuras 2.9 y 2.10, en las cuales vemos dos clasificaciones clásicas de Flynn que se diferencian básicamente en el número de bancos de memoria , pues, en la primer gráfica se usa el mismo número de bancos de memoria , mientras que para la segunda, el número de bancos de memoria no es igual al número de procesadores . A pesar de estas particularidades, el arreglo puede funcionar con la misma eficiencia en cualquiera de los dos casos. Hay que tomar en cuenta que la eficiencia del esquema propuesto en la Figura 2.9 dependerá del manejador de procesos , mientras que, para la Figura 2.9 dependerá de la red de interconexión entre las memorias y los procesadores.

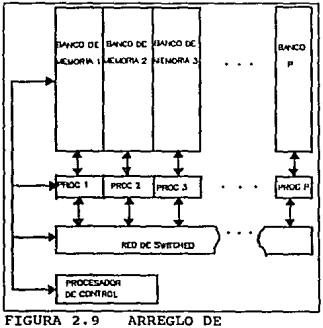


FIGURA 2.9 ARREGLO DE PROCESADORES DE MEMORIA DISTRIBUIDA

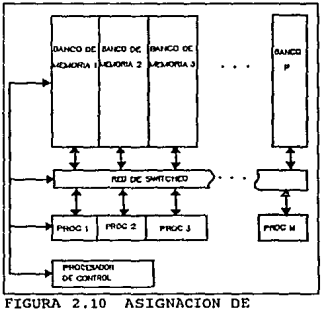


FIGURA 2.10 ASIGNACION DE MEMORIA DEL ARREGLO DE PROCESADORES POR MEDIO DE LA RED DE SWITCHEO

Uno de los mas controversiales temas en el manejo de arreglos de procesadores es en la distribución del poder de procesamiento, pues, ésto depende de que tan paralelizable pueda ser cualquier aplicación, lo cual nos lleva a pensar en el paralelismo como una medida de eficiencia en sistemas digitales de computo encauzados hacia el paralelismo. Dentro de esto último, la eficiencia del paralelismo se puede dar por el Hardware o por el Software. Para el hardware, la eficiencia se presentará en la medida de los retrasos de tiempo presentados en los diferentes niveles de integración (microcódigo, unidades aritméticas, comunicación entre procesadores, comunicación con la memoria, etc), lo que lleva a medidas bastante complejas sobre el retraso de tiempo presentado para diferentes aplicaciones con diferentes arquitecturas.

Dentro del arreglo de procesadores, tenemos como caso particular a los sistemas multiprocesadores[3]. La organización del sistema hardware multiprocesador viene determinado por la estructura de interconexión entre las memorias y los procesadores. Tres tipos de interconexiones diferentes se han practicado en el pasado:

1) BUS COMUN DE TIEMPO COMPARTIDO
2) RED COMPUTADORA DE BARRAS CRUZADAS
3) MEMORIAS MULTIPUERTO

Cabe decir que, los arreglos de procesadores son una forma bastante amplia de arquitecturas que girarán entorno a los diferentes sistemas de flujo de instrucciones que a su vez tendrán diferentes tipos de categorías:

-FLYNN
-MEMORIAS
-TAZAS DE PARALELISMO

Por último podemos decir que,los sistemas de arreglos de procesadores para la categoría de Flynn serán considerados como sistemas SIMD y para el manejo de memorias podrán entrar en cualquier arquitectura según sea la arquitectura; sin embargo al arreglo de multiprocesadores se ubicará en la categoría MIMD.

2.5 TAXONOMIA DE FLYNN

Dentro de los esquemas presentados como clasificatorios de los sistemas paralelos, encontramos los que están dados por la clasificación de Flynn. La cual, por su flexibilidad ante el análisis de diferentes arquitecturas ocupa un lugar preponderante para las expectativas de este trabajo[2].

Esta clasificación se basa en el flujo de datos o instrucciones según sea el tipo de sistema analizado[1] , cabe mencionar que entendemos por término flujo al que se emplea para denotar una secuencia de elementos que

ejecuta o sobre los que opera un solo procesador . La instrucciones o los datos se definen con respecto a una máquina referenciada. Un flujo de instrucciones es una secuencia ejecutada por un elemento de procesamiento; un flujo de datos es una secuencia de entradas y salidas parciales o totales producidas por el flujo de instrucciones.

Las diferentes organizaciones de computadores se caracterizan por la multiplicidad de hardware provisto para atender a los flujos de instrucciones y datos. A continuación mostramos las cuatro categorías propuestas por Flynn:

-SISD. una Sola Instrucción un Solo Dato
-SIMD. una Sola Instrucción Múltiples Datos
-MISD. Múltiples Instrucciones un Solo Dato
-MIMD. Múltiples Instrucciones Múltiples Datos

Los cuatro tipos de organizaciones se encuentran mostrados en la Figura 2.11[1]. La categorización depende de la multiplicidad de sucesos simultáneos que ocurren en los componentes del sistema. Conceptualmente sólo son necesarios tres tipos de componentes , como se muestra en el diagrama. Las instrucciones y los datos se toman de los módulos de memoria. Las instrucciones se decodifican en la unidad de control, que envía el flujo de instrucciones decodificadas a las unidades procesadoras para su ejecución. Los flujos de datos circulan entre los procesadores y la memoria bidireccionalmente. Se pueden utilizar múltiples módulos de memoria en el subsistema de memoria compartida . Cada flujo de instrucciones es generado por una unidad de control independiente . Los múltiples flujos de datos se originan en el módulo de memoria compartida .

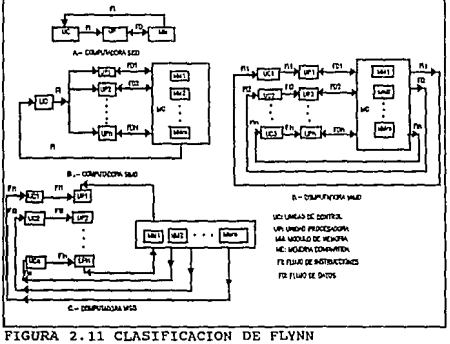


FIGURA 2.11 CLASIFICACION DE FLYNN

Los recursos de entrada y salida no se muestran en estos diagramas de bloques simplificados. Para tener un mejor entendimiento de esta clasificación, podemos ver la Figura 2.12 que nos permite ubicar de forma mas estratégica las aplicaciones de la categoría de Flynn que se explica a continuación:

ORGANIZACIÓN SISD . Esta organización, mostrada en la Figura 2.11a, representa la mayoría de las computadoras serie de hoy. Las instrucciones se ejecutan secuencialmente pero pueden estar traslapadas en las etapas de ejecución.

ORGANIZACIÓN SIMD . Esta clase corresponde a los procesadores matriciales . Como se ilustra en la Figura 2.11b existen muchos elementos de procesamiento (EP) supervisados por una misma unidad de control. Todos los EP reciben la misma instrucción emitida por la unidad de control pero operan sobre diferentes conjuntos de datos procedentes de flujos distintos. El sistema de memoria compartida puede contener múltiples módulos .

ORGANIZACIÓN MISD . Este tipo de arquitectura se muestra en la Figura 2.11c .Existen n unidades procesadoras; cada una recibe distintas instrucciones que operan sobre el mismo flujo de datos y sus derivados . Los resultados de un procesador pasan a ser la entrada del siguiente procesador en el macrocauce. Esta arquitectura ha recibido mucha menos atención y ha sido tachada de poco práctica por algunos arquitectos de computadoras.

ORGANIZACIÓN MIMD . La mayoría de los sistemas multiprocesadores y de los sistemas con múltiples computadoras pueden incluirse en esta categoría (Figura 2.11d). La computadoras MIMD intrínsecamente implica interacciones entre los n procesadores porque todos los flujos de memoria se derivan del mismo espacio compartido por todos los procesadores. Si los n flujos de datos provinieran de subespacios disjuntos dentro de memorias compartidas, entonces tendríamos la llamada operación SISD múltiple (MSISD), que no es más que un conjunto de n sistemas monoprocesadores SISD independientes. Una computadora MIMD intrínseco es fuertemente acoplado si el grado entre interacciones es elevado. En caso contrario los consideraremos débilmente acoplado . La mayoría de computadoras MIMD comerciales son débilmente acoplados.

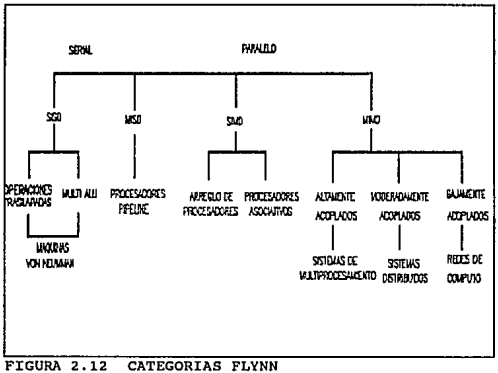


FIGURA 2.12 CATEGORIAS FLYNN SEGUN SU APLICACION

Cabe señalar que, este tipo de categorías no son estrictas, pues, hay diferentes sistemas de computo que desde varios puntos de vista pueden entrar en más de alguna de las categorías que propone Flynn[7].

2.6 TOPOLOGIAS

En esta sección se busca la revisión clara de las diferentes topologías que se pueden dar entre los sistemas de procesadores, tratando con esto,el alcance de una idea mas clara de lo que es sistema paralelo.

La primera consideración que se debe de tener, es el número de elementos que conforman a la red (procesadores o elementos de procesamiento) y los patrones de interconexión que se usa entre ellos.

Hay que señalar que, los sistemas considerados, serán distribuidos, lo cual nos lleva a pensar en sistemas homogéneos que requieren un equilibrio en cuanto al manejo de la información se refiere ; por ende las topologías tienden a presentar formas homogéneas pero versátiles en su integración para permitir el manejo de estructuras variables[6].

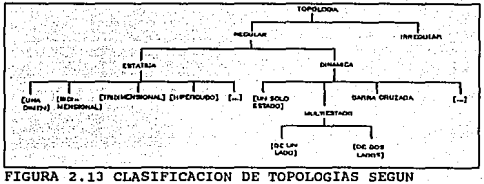


FIGURA 2.13 CLASIFICACION DE TOPOLOGIAS SEGUN SU MANEJO DE LAS COMUNICACIONES

La Figura 2.13 muestra una forma en la cual, podemos clasificar a las diferentes topologías en base a su versatilidad en el manejo de la información.

Los patrones de integración pueden ser considerados en dos grupos; patrones regulares y grupos de arreglo. El patrón regular implica un rol básico, siguiendo la construcción de topologías de diferentes formas. Una estructura bien conocida de este tipo es el cubo N-dimensional o N-cubo, el cual es mostrado en la figura 2.14, para N=0,1,2,3 y 4. Note que al final el cubo (N+1), el cubo enésimo es trasladado a (N+1) dimensiones. El número de nodos son doblados en cada estado logrando que un N-cubo tenga 2^N nodos y un total de N2^N/2=N2^(N-1) ejes (N ejes por nodo). Se han propuesto computadores de propósito general (supercomputadoras) usando esta estructura.

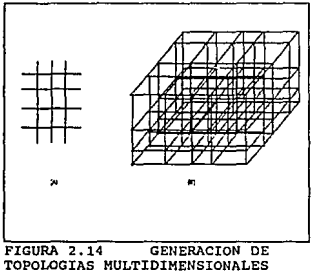


FIGURA 2.14 GENERACION DE TOPOLOGIAS MULTIDIMENSIONALES A TRAVES DEL SWITCHEO

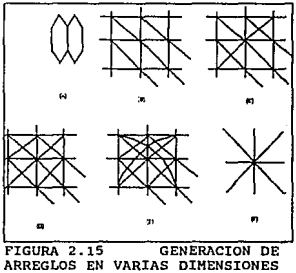


FIGURA 2.15 GENERACION DE ARREGLOS EN VARIAS DIMENSIONES POR MEDIO DE MTV

Otra estructura típica regular son los arreglos mostrados en la Figura 2.15 donde topologías de dos y tres dimensiones son expuestas. En la Figura 2.15a los nodos en el contorno del cuadrado son conectados al otro lado , de modo que, la superficie obtenida es cerrada. Esto nos da una forma toroidal . El número de nodos esta dado por M^N donde M es el número de nodos en una dimensión. El número de ejes está dado por NM^N.

Muchas estructuras simples pueden ser concebidas en el contenido de una topología regular[6]; por ejemplo un hexágono es más versátil que un simple cuadrado , como lo muestra la Figura 2.15e.

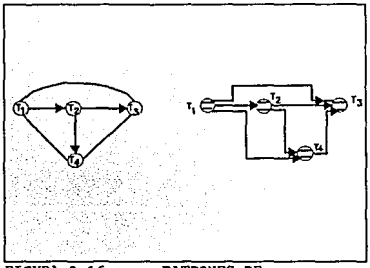


FIGURA 2.16 PATRONES DE INTERCONEXION PARA CONSTRUIR TOPOLIGIAS REGULARES

Cuando la comunicación de la topología no es completa, las aplicaciones que se hagan en ella no tendrán la necesidad de utilizar la comunicación total, ni la comunicación entre nodos intermedios, lo cual nos da mayor rapidez en el procesamiento. En este caso la noción del promedio de las trayectorias es muy importante. Esto se refiere al número de estados intermedios que son necesarios entre los nodos que proveen la comunicación total . Esto es, desde el nodo i al nodo j , si existe una trayectoria pequeña con a(i,j) de nodos intermedios, entonces

$$\overline{a}=\sum_{i,j}\frac{a(i,j)}{L}$$

donde L es el número de pares de nodos (L=N(N-1)).

Además de esta clasificación, existen los sistemas altamente acoplados y los sistemas de bajo acoplamiento[6].

Se dice que un sistema es altamente acoplado cuando un número de procesadores tiene acceso a una memoria común . Esta es una de las configuraciones más comúnmente usadas , las cuales son llamadas sistemas de formas limitadas , generalmente arriba de 16 procesadores . Una memoria común provee un medio de transferencia de datos bastante rápido , tan bueno como para compartir la memoria en códigos comunes. Aunque para sistemas pequeños se requiere un sistema particular, el cual, puede disminuir la velocidad y la rapidez de la comunicación de datos. El manejador del sistema de memoria también provee problemas de software. Hay cuatro tipos de sistemas de alto acoplamiento:

1.- BUS COMPARTIDO
2.- MEMORIA MULTIPUERTO
3.- VENTANA DE BUS
4.- SWITCH DE BARRAS CRUZADAS

Dado que en esta sección, el estudio de los formatos de las memorias no es lo que se busca, dejaremos este tema para retomarlo posteriormente.

Por otro lado existe los sistemas bajamente acoplados, los cuales, están construidos con microprocesadores autónomos con sus memorias locales y logran tener una comunicación a traves de sistemas de entrada y salida de información. A diferencia de los sistemas altamente acoplados, el medio de comunicación es bajo y requiere un número de procesos intermedios. El medio de interconexión es más flexible, cubriendo distancias desde algunos centímetros hasta cientos de metros.

Los medios de interconexión de sistemas bajamente acoplados pueden ser paralelos, o seriales[6]. Estas son las estructuras más comunes que se presentan en este tipo de sistemas:

1.- SISTEMAS DE CONEXION DE BUS (ETHERNET)
2.- MALLAS
3.- REDES DE MICROPROCESADORES (multicomputadoras de topología variable MTV)

Los dos primeros puntos que se proponen quedan fuera del alcance de este trabajo, pues, son sistemas que se usan en redes de computadores; aunque pueden ser considerados como sistemas multiproceso. Para el tercer sistema propuesto, tenemos que, es parte fundamental de aquellos algoritmos que se refieran al alto rendimiento de la arquitectura usada.

Estas redes de microprocesadores nos dan la facilidad del manejo de diferentes topologías según el algoritmo que se este aplicando[6].

El objetivo de las redes MTV es el desarrollar métodos para la comunicación y sincronización referente a una multiplicidad de computadoras con la flexibilidad del manejo de la topología para aplicaciones

particulares. Para sistemas de tiempo real, es fácil identificar las tareas paralelas puesto que son usualmente distribuidas espacialmente. Sin embargo un arreglo es más factible para resolver ecuaciones diferenciales en dos dimensiones . Para simulación de sistemas dinámicos cada procesador puede tomar una forma irregular de cada tarea de la simulación, logrando con esto el manejo de variables con una mayor eficiencia. De lo anterior podemos concluir que para cada sistema la topología óptima varía según las necesidades de cada algoritmo . En algunas aplicaciones, los patrones de definición de cada topología dependen de la interactuabilidad de tareas y su comunicación entre ellas, así como su proximidad en lo que se refiere a su topología. En ciertas aplicaciones los patrones de la red pueden ser conmutados para la posibilidad de la expansión de algunas características puntuales.

El principal componente de las redes MTV es el nodo de la computadora (NC), el cual consiste de una computadora local (CL), las comunicaciones de computadoras (CC), comunicador entre computadoras (CEC). La comunicación de la computadora contiene alguna memoria local para buscar información. El CEC contiene terminales de entradas y salidas para comunicarse entre otros NC .

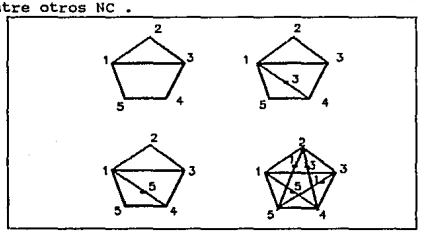


FIGURA 2.17 TOPOLOGIAS GENERADAS POR MEDIOS DE LINKS

Considerando la Figura 2.16a donde cuatro tareas paralelas son mostradas con líneas de interconexión indicando los requerimientos de interconexión entre ellas . El sistema puede ser implementado por una red MTV como la mostrada en la Figura 2.16b. Note que aquí cada tarea ha sido alojada en un nodo de computadora y una línea ha sido dedicada para cada requerimiento entre dos tareas.

Los sistemas MTV habilitan las redes de computadoras donde la topología puede ser alterada en dos niveles : (1) físico, (2) lógico. Para establecer las redes físicas entre varios nodos de computadoras , se tiene que obtener una topología dada. Las conexiones y desconexiones posteriores en la red significará que se altera la topología. Esto puede ser implementado por medio de tarjetas base, donde los cambios requieren ser

hechos frecuentemente. La topología obtenida por los nodos de computadoras y los links físicos entre ellos es llamada máquina gráfica (MG). La topología requerida para una aplicación es llamada un programa gráfico (PG).

Dado un PG el cual satisface ciertos contrastes ,su correspondiente MG puede ser atacado como se muestra en la Figura 2.16. Sobre una MG dada es posible establecer links lógicos entre dos NC con conexiones no directas entre ellos , ocupando nodos intermedios para este propósito (Figura 2.17). El MTV implementa dos esquemas :

1.- PAQUETE DE SWITCHEO
2.- CIRCUITO DE SWITCHEO

El paquete de switcheo requiere la determinación de una estrategia de ruteo donde cada NC contiene un vector de ruteo en paquetes directos que actúan sobre los NC. Esto es hecho por medio de la identificación del destino de la información, para poder así encontrar la forma mas eficiente de comunicación. Desde que se usan nodos intermedios para la transmisión de información el tiempo se puede considerar largo. De esto podemos decir que, pueden existir links virtuales entre dos NC , pues, la comunicación se da por medio de los nodos intermedios. Por ejemplo desde una MG mostrada en la Figura 2.17a, un número de líneas pueden ser dibujadas a partir del concepto de links virtuales. La Figura 2.17b y c muestran como la misma topología es obtenida usando dos diferentes nodos intermedios , indicando la redundancia en el sistema. La Figura 2.17d muestra como una red totalmente conectada puede ser obtenida. Si MG satisface la condición de que todos los links entre los nodos son bidireccionales, es posible construir una matriz de conexiones entre ellos por medio de la inclusión de links virtuales. El manejador de links virtuales, esta hecho por la computadora de comunicaciones y es totalmente transparente entre los nodos.

La comunicación más efectiva es provista en el MTV por un circuito de switcheo de links el cual puede ser mantenido mientras dura el proceso realizándose la comunicación por medio de transferencia de bloques. Cada circuito de switcheo no necesita establecer una fuente de transmisión y de recepción de información ; esto incrementa su uso en comparación del paquete de switcheo. Note que para ambos casos el retraso de tiempos es el mismo, siendo dedicado por un número de nodos intermedios que existen sobre un arreglo dado.

Un punto importante es que la trayectoria del switcheo del circuito , lógicamente hace dos nodos adyacentes , esto es, implementa la fuente y el destino. Esto es visto como el equivalente del establecimiento de links

temporales entre nodos distantes. Para ilustrar esto , consideremos una red rectangular formada por 16 nodos , la cual es una máquina MTV como la mostrada en la Figura 2.18a . Supongamos que tenemos un problema que tiene la geometría mostrada en la Figura 2.18b. Para ser capases de resolver el problema geométrico dentro de la máquina, nosotros estableceremos una trayectoria de circuito virtual (TCV) entre los nodos 4 y 12 vía el nodo 8, y otra TCV entre los nodos 8 y 11 vía el nodo 7 (líneas punteadas). Esto hace funcional a la geometría de la máquina, pues, la convierte en la mostrada en la Figura 2.18c, la cual ahora corresponde al problema geométrico dado.

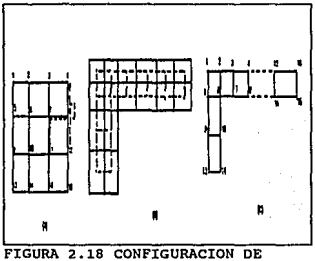


FIGURA 2.18 CONFIGURACION DE NODOS PARA LA CREACION DE TOPOLOGIAS VIRTUALES

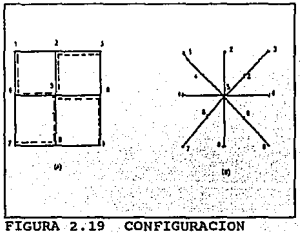


FIGURA 2.19 CONFIGURACION DE REDES VIRTUALES

La posibilidad de utilizar links TCV introduce a nuevos vecinos de lo cual se basa la topología virtual (TV) . Para ilustrar esto considere la MG dada en la Figura 2.19a . Usando el concepto de TCV entre los nodos 4 y 1, 5 y 3, 5 y 9, 5 y 7, 5 y 4 una configuración estrella puede ser conseguida.

2.7 MANEJO DE MEMORIAS

Generando una dicotomía del procesamiento paralelo desde la expectativa del manejo de la memoria , vemos que, existen dos grandes campos en el desarrollo de sistemas ; memorias compartidas y memorias distribuidas[3].

La forma en como los sistemas tienen acceso a una memoria común es conocido como arquitecturas de memoria compartida. Mientras que para un sistema en donde las memorias sean propiedad exclusiva de cada microprocesador serán llamadas arquitecturas de memoria distribuida. Una gama de arquitecturas han sido diseñadas utilizando cualquiera de los dos conceptos . Es necesario recalcar que, los sistemas que se han mencionado en todo el desarrollo del capítulo , pueden entrar en diferentes esquemas de diseño ; teniendo con esto, un traslape entre las diferentes categorías de clasificación , y no con esto verse afectados entre si , pues cada clasificación ve a un sistema desde diferentes puntos de vista .

Ahora bien, englobando los conceptos antes vertidos para catalogar al sistema , que se propone en el presente trabajo (Capítulo 4) tendremos que definirlo en base a sus siguientes características :

1.- TIPO DE PROCESAMIENTO (escalar o vectorial)
2.- ARREGLO DE PROCESADORES
3.- TAXONOMIA DE FLYNN
4.- MANEJO DE MEMORIA
5.- TIPO DE ACOPLAMIENTO

En principio de cuentas, el sistema propuesto está basado en el diseño de elementos de procesamiento llamados TRAM que a su vez fundamentan su construcción a partir del microprocesador llamado Transputer , el cual, como se verá en el capítulo 3, es un microprocesador de tipo RISC escalar que presentará ciertas facilidades en su modo de comunicación y en la forma de procesamiento.

Además, los elementos de procesamiento propuestos, trabajarán la información en paralelo con un cierto tiempo de desperdicio producido por las comunicaciones entre ellos, con lo que podemos decir que el sistema en su conjunto es un arreglo de multiprocesamiento.

Habiendo definido lo anterior es conveniente categorizar al sistema desde el punto de vista de FLYNN . Para lo cual se requiere definir al elemento de procesamiento por la forma del flujo de datos que se da en el y entre el sistema.

El elemento de procesamiento , como se ve en el Capítulo 4 , es un sistema de memoria distribuida que recibe un algoritmo de un puerto serial (Capítulo 3) que posteriormente lo ejecutará devolviendo la información procesada por medio del mismo puerto serial.

Por lo que se refiere al sistema en su conjunto , este presenta una topología del tipo MIMD , con un manejo de memoria distribuida de acoplamiento bajo. Estos dos últimos puntos están definidos en base a que cada EP tiene su propia memoria y solo se comunica con el mundo exterior cuando un proceso lo requiere sin necesidad de revisar su contorno cada determinado tiempo.

2.8 LENGUAJES DE PROGRAMACION PARALELA

Los lenguajes de alto nivel y sus traductores han sido esenciales para el desarrollo de aplicaciones en mono-procesadores. Lo mismo , sin embargo, no se puede decir para sistemas de multiprocesamiento. La inmensa variedad de aplicaciones y arquitecturas, y la diversidad de filosofías acerca de éstos sistemas ocasiona una falta de estructura que genera, gran dificultad en el diseño de lenguajes, dando con esto, un retraso a las aplicaciones posibles para este tipo de sistemas[7]. Lo anterior ocasiona que el diseño de lenguajes no sea completamente general y modular con lo cual se pudiera exportar hacia otras máquinas.

Otra de las dificultades en usar lenguajes aplicables a sistemas multiprocesadores es la necesidad de un traductor[1]. El traductor es el que se encarga de generar la compatibilidad entre el lenguaje y el sistema donde se esta integrando. Desde que decimos que no hay una arquitectura uniforme, el traductor tendrá que ser capaz de poder modificarse para varias configuraciones . Idealmente, el traductor deberá tomar las ventajas de cada estructura y ser capaz de actuar en forma modular .

Muchos de los lenguajes propuestos en el área de programación concurrente también se han desarrollado en el modelo de computo distribuido.

Algunas veces el trato de modelos en lenguajes es inseparable. Cada uno de los modelos se ve afectado por la metodología de programación lo que nos da una buena técnica de prueba para ciertos tipos de lenguajes. Un modelo da un esqueleto conceptual en el cual se discute y entiende la conducta de la computación concurrente, con lo que se quiere capturar la filosofía de entendimiento de la programación de un lenguaje.

Algunas de las características de un lenguaje concurrente se presentan en la siguiente lista:

Poder de expresión.- Este se refiere a la habilidad del lenguaje para expresar ciertas estructuras , como la recursividad , la modularidad , el no determinismo, etc. . Esta habilidad es referida a la capacidad o suficiencia de cada lenguaje.

Probabilidad.- Esto se refiere al poder del lenguaje usado para características específicas acerca de las propiedades de programación. Algunas de estas características es la lógica usada para ser capaz de manejar formalmente las propiedades del sistema.

Facilidad y eficiencia de implementación.- Aun cuando un programa pueda ser diseñado , ésto , no nos indica que sea eficiente para el tipo de sistemas que manejamos, por lo tanto habrá que ver y medir la forma en que se maneja las características de cada sistema .

Facilidad de uso.- La presencia de características poderosas no significa que esto sea fácil de usar. El uso fácil y el poder de expresión son criterios complementarios. La facilidad de uso , también esta basada en la flexibilidad de la construcción del lenguaje en cuestión así como de las aplicaciones que se deseen.

Impacto de los cambios.- Si la construcción no incluye o forja a un alto grado o modularidad, un cambio en la definición de un proceso puede necesitar muchos cambios sobre el resto del sistema. Esto debe de ser tomado en cuenta, particularmente si el número de procesos envueltos es muy alto.

En este trabajo se describe en particular el lenguaje de programación paralela OCCAM y su arquitectura asociada (véase Capítulo 3), el transputer, mismos que son utilizados en los ejemplos de aplicación de esta tesis.

2.9 ALGORITMOS PARALELOS

Las arquitecturas paralelas del tipo MIMD y principalmente sistemas de memoria distribuida como el transputer , ofrecen un aprovechamiento modular a la construcción de computadoras los cuales pueden ser tolerados con aplicaciones individuales.

Idealmente un programa corre N veces mas rápido en N procesadores que en un solo procesador , sin embargo la velocidad puede ser mucho menor[1]. El diseño de algoritmos que trabajan con esta clase de velocidad tienen toda una área de investigación. Desde el algoritmo, el lenguaje hasta el hardware están íntimamente conectados , este ejercicio es complicado entenderlo en forma general. Desafortunadamente las computadoras paralelas marcan mucho la diferencia entre el rendimiento de un buen y un mal

programa , al menos más que una computadora serial. Es aquí donde se dice que el usuario debe de pensar en paralelo , pues si no, las aplicaciones que se realicen serán poco eficientes.

En suma, las consideraciones finales para un análisis numérico de una aplicación en paralelismo son las siguientes:

a) COMO UN DATO ES CAPAZ DE SER DISTRIBUIDO EN MEMORIA
b) COMO LOS CALCULOS SON DISTRIBUIDOS A LO LARGO DE TODOS LOS PROCESADORES
c) LA COMUNICACION ENTRE PROCESADORES
d) LAS CONEXIONES ENTRE PROCESADORES, SI SON RECONFIGURABLES

El ánimo de hacer paralelismo en un algoritmo depende del paralelismo de la computadora como camino para minimizar el tiempo de ejecución de un programa. En cualquier punto de un algoritmo, el paralelismo de éste es el número de operaciones que son independientes y pueden ser trabajadas concurrentemente. Esto puede variar de un estado a otro. El paralelismo de un hardware natural es el número de procesadores que pueden correr concurrentemente , incluyendo procesos aritméticos y procesos de liga en el transputer. Nosotros podemos medir la eficiencia por medio de algoritmos paralelos , como se mostró al inicio de este capítulo.

$$eficiencia = \frac{t_1}{N * t_N}$$

t_1= tiempo tomado por un programa en un procesador

t_N= tiempo tomado por un programa en N procesadores

Si el tiempo tomado para rutear datos entre los procesadores es comparable a el tiempo de la operación aritmética, entonces este jugará un importante rol en el manejo del algoritmo. De hecho, en el tiempo presente, el factor más importante es la relación de los procesos y la velocidad de comunicación.

Para lo que tendremos, que basarnos en características muy propias de los microprocesadores a usar .

En base a lo anterior vemos que el transputer permitirá un manejo total de la programación sobre el tipo de paralelismo que se maneje en algún programa. Sobre de esto, tendremos que, cualquier tipo de sistemas tendrán por lo común los siguientes tipos de oportunidades para atacar a su propia programación:

a) Tareas independientes/ eventos de paralelismo.- en el cual cada procesador esta ejecutando el mismo programa en la separación de todos los otros procesadores.

b) Paralelismo geométrico.- en el cual cada procesador ejecuta el mismo programa en datos correspondientes a la subregión del sistema siendo simuladas y comunicadas mediante datos de los procesadores vecinos en los límites de las regiones vecinas.

c) Palaralelismo algorítmico.- En esta parte cada procesador es responsable de una parte del algoritmo , y todo el transporte de datos para cada procesador.

2.10 RECAPITULACION

Ahora bien, retomando los conceptos vertidos para catalogar al sistema que se propone en este trabajo (Capítulo 4), tendremos que definirlo en base a las siguientes características:

- TIPO DE PROCESAMIENTO, ESCALAR O VECTORIAL
- ARREGLO DE PROCESADORES
- TAXONOMIA DE FLYNN
- MANEJO DE MEMORIA
- TIPO DE ACOPLAMIENTO

En principio, el sistema propuesto esta basado en el diseño de elementos de procesamiento llamados TRAM (Transputer Module) que a su vez fundamentan su construcción a partir del microprocesador llamado Transputer[3]. El cual, como se verá en el capítulo 3, es un microprocesador tipo RISC escalar que presentará facilidades en su modo de comunicación y en la forma de procesamiento; Además los elementos de procesamiento propuestos trabajarán la información en paralelo con un cierto tiempo de desperdicio, producido por las comunicaciones entre ellos con lo que podemos decir, que el sistema en su conjunto es un arreglo de procesadores.

Habiendo definido lo anterior es conveniente categorizar al sistema desde el punto de vista de la Taxonomía de Flynn. Para lo cual, se requiere definir al elemento de procesamiento por la forma del flujo de datos que se da en el y entre el sistema.

El elemento de procesamiento , es un sistema de memoria distribuida que recibe un algoritmo de un puerto serial (capítulo 3) que posteriormente lo ejecutará devolviendo la información procesada por medio del mismo puerto serial. Por lo que, se define al sistema en su conjunto como un sistema tipo MIMD, con un manejo de memorias distribuida de

acoplamiento bajo. Estos dos últimos puntos están definidos en base a que cada EP tiene su propia memoria y solo se comunica con el mundo exterior cuando un proceso (tarea) lo requiere.

2.11 REFERENCIAS

1.- FLEMING P., GARCIA F., PARALLEL PROCESSING IN DIGITAL CONTROL, SPRINGER-VERLAG, PRIMER EDICION, 1992

2.- HOCKNEY RW, JESSHOPE CR, PARALLEL COMPUTERS, ADAM HILGER LTD BRISTOL,PRIMER EDICION,1981

3.- HWANG KAI, ARQUITECTURAS DE COMPUTADORES Y PROCESAMIENTO PARALLELO MAC GRAW HILL, SEGUNDA EDICION , 1988

4.- HWANG KAI, PARALLEL PROCESSING FOR SUPERCOMPUTERS AND ARTIFICIAL INTELLIGENTS, MAC GRAW HILL, 1989

5.- INMOS, iq SYSTEMS DEVELOPMENT, INMOS, SEGUNDA EDICION , 1989

6.- PAKER Y., MULTIMICROPROCESSOR SYSTEMS, ACADEMIC PRESS, PRIMER EDICION, 1983

7.- TREW ARTHUR,WILSON G., PAST PRESENT PARALLEL, SPRINGER VERLAG,1989

8.- REIGNS G. L., HIGHLY PARALLEL COMPUTERS, ELSERVIERS CIENCE PUBLISTERS, IFIP,1987

CAPITULO III

TRANSPUTER Y OCCAM

3.1 INTRODUCCION

El rápido desarrollo de la tecnología de cómputo ha tenido un impacto importante en los sistemas de procesamiento de información. Siendo crítico en aplicaciones de procesamiento digital de señales e imágenes en tiempo real, en donde se requiere la capacidad de realizar una gran cantidad de operaciones en intervalos de muestreo muy cortos, del orden de 5 a 10ms.

Debido a las crecientes demandas, este tipo de sistemas de procesamiento requieren de una capacidad de cómputo que ha empezado a rebasar el desempeño de las arquitecturas secuenciales, con las que hasta ahora se han implantado distintos sistemas.

Esto ha estimulado la investigación y desarrollo de nuevas arquitecturas que ofrezcan la alternativa de alta velocidad y bajo costo. La disponibilidad actual de un rango de arquitecturas de procesamiento paralelo, está creando nuevas oportunidades para la implantación de sistemas en tiempo real , permitiendo su desarrollo en sistemas de procesamiento rápido como los requeridos en aplicaciones de procesamiento digital de señales.

Teniendo como base a estos antecedentes y por las características internas que se explicarán mas adelante se eligió al transputer como el microprocesador apto para el desarrollo de arquitecturas de procesamiento paralelo de memoria distribuida (Capítulo 4).

El transputer y su lenguaje de programación OCCAM, han sido diseñados para el desarrollo de procesamiento paralelo. El rendimiento es escalable linealmente, debido a la incorporación de procesadores al sistema. Estos pueden conectarse por medio de puertos seriales (links), no requiriendo de ningún circuito adicional para su comunicación.

Como se ha señalado, el objetivo del trabajo es el diseño de una arquitectura capaz de procesar en tiempo real una señal tipo Doppler de ultrasonido. La técnica de multiprocesamiento nos es útil para el procesamiento de la señal, puesto que lo que se pretende realizar , es el tomar una serie de bloques consecutivos de la señal y distribuirlos en la misma cantidad de módulos de procesamiento para que en el mismo tiempo de un intervalo de muestreo se puedan procesar varias ventanas de información simultáneamente. Lo anterior es mostrado en la Figura 3.1.

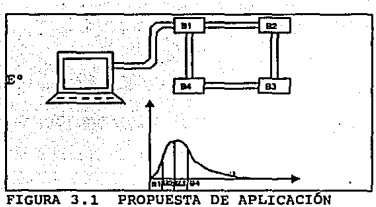


FIGURA 3.1 PROPUESTA DE APLICACIÓN

El algoritmo de procesamiento es el mismo para los diferentes bloques de la señal, buscando ser congruentes con lo señalado anteriormente.El algoritmo será tratado en el capítulo cinco e implementado en el Apéndice C.

Para muchos de los sistemas de memoria distribuida , la topología del procesador es la principal consideración, además de saber como estructurar la arquitectura entre los transputers. Con cuatro ligas (links) por transputer hay varias topologías o redes de configuración , que son posibles. Por ejemplo, se pueden estructurar cuatro transputers en un RING con cada procesador conectado con otros dos. La Figura 3.2 ilustra cuatro procesadores en una topología de RING. Las topologías RING son comunes en redes para computadoras dado que ellas solo requieren una línea de entrada y una de salida.

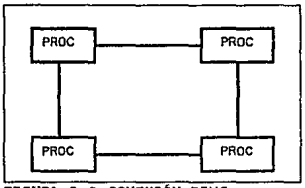


FIGURA 3.2 CONEXIÓN RING

Sin embargo, el transputer tiene cuatro links con lo cual es posible, proveer más interconexiones entre elementos de procesamiento. Si cualquier LINK fuera usado para conectarse con cualquier elemento de procesamiento un gran número de configuraciones podrían ser posibles.Cuando

cada elemento de procesamiento en una red es conectado a todos los otros elementos, se dice que es altamente conectada "full-conected" . Una topología RING altamente conectada es mostrada en la Figura 3.3.

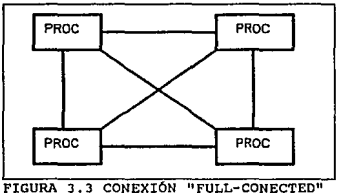


FIGURA 3.3 CONEXIÓN "FULL-CONECTED"

Los transputers no están limitados a las topologías RING. En general , la topología de una red de computadoras es determinada por la comunicación necesaria en el programa de aplicación. Esas necesidades reflejan la solución del algoritmo para el problema que se esta tratando de resolver.

Para algunas aplicaciones, es posible configurar una serie de transputers dentro de una estructura de árbol como se muestra en la Figura 3.4

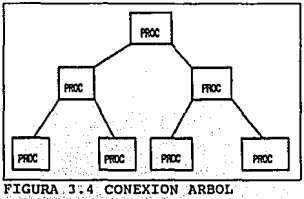


FIGURA 3.4 CONEXION ARBOL

En base a los requerimientos del proyecto , nosotros elegiremos una red de pipeline como lo explicaremos en el Capítulo 4.

Recordando las cuatro categorías propuestas de FLYNN (Capítulo 2), vemos que, la potencia de los transputers es en base a la flexibilidad para ser configurados en cualquiera de esas categorías. Con esto, el diseño de un sistema digital de computo, en general, puede ser planteado según el problema que se ataque. Esto provee un gran desarrollo a bastantes aplicaciones que estaban atadas en la estructura tradicional de los sistemas tipo SISD.

Cada uno de los cuatro links seriales en un transputer permiten que se le conecte con una serie de elementos de procesamiento iguales a el, con lo cual se logra construir una red de forma parecida a una red de computadoras. Sobre el camino en que las topologías de las redes varían; también las redes de transputers pueden variar.Es decir, el análisis que se aplica a las redes de computadoras también es válido aplicarlo a las redes de transputers. La principal idea es dar una mayor capacidad al computo, usando, múltiples procesadores juntos para la solución de un problema. El único aspecto que hace esto posible es la capacidad de comunicación entre elementos de procesamiento de memoria distribuida (así definidos como TRAMS en el Capítulo 2).

Así pues, los transputers son máquinas de procesamiento de memoria distribuida que pueden ser configuradas dentro de una variedad de arquitecturas .

Como cualquier microprocesador, el transputer se adaptará, a la arquitectura del sistema que sea necesario aplicar para desarrollos específicos. Usualmente la topología de la red es cambiada; lo cual es posible; al construir redes de transputers que pueden se reconfiguradas al estar corriendo una cierta aplicación (configuración dinámica).

El uso de redes reconfigurables permite que el tiempo de comunicación pueda ser reducido, por medio de la desaparición de elementos de procesamiento que tienen como única función; el recibir y transmitir datos sin desarrollar ningún proceso . Una red reconfigurable puede ser usada por muchas aplicaciones que es particularmente común para grandes desarrollos de sistemas de multiprocesamiento . Otro importante beneficio de las redes switcheadas o reconfigurables es la facilidad de programación, puesto que los sistemas desarrollados en base a esta filosofía , tienen la flexibilidad de mapear los algoritmos de forma óptima logrando facilitar la conceptualización del software. La reconfiguración permite el omitir procesadores intermedios que solo cumplirán con la recepción y transmisión de datos sin realizar ninguna modificación sobre ellos.

Si la configuración es implantada por switches controlados por software, la topología puede ser modificada estáticamente , casi estáticamente o de forma dinámica[5]. En el switcheo estático, la topología es

configurada antes que la aplicación sea cargada en la red. El switcheo casi estático es usado cuando todos los procesadores pueden ser sincronizados en determinados puntos del programa ; esto es; en un punto de sincronización , todas las comunicaciones y todos los procesadores esperan para que las conexiones de los links puedan ser cambiadas. Esto es aplicable al procesamiento de imágenes en una máxima relación de datos , por ejemplo, si la red es configurada dentro de un pipeline; el cual tiene una máxima "superficie de trabajo", cuando los datos han sido cargados en el sistema este puede ser reconfigurado dentro de un arreglo bidimensional para operaciones de bajo nivel, las cuales serán ejecutadas en un modo SPrMD. Cuando todas las operaciones de bajo nivel han sido computadas por todos los procesadores, el arreglo puede ser configurado como un árbol , para un procesamiento de alto nivel.

El switcheo dinámico permite nuevos métodos de programación, como es la carga balanceada dinámica ,la cual en una red de transputers es relativamente sencilla , pues estas máquinas son de flujo de control que pueden ser usadas como máquinas de flujo de datos o como máquinas de reducción[4].

Hay tres grandes áreas de aplicación para los transputers ; como tarjetas de aceleración para PC's y estaciones de trabajo, como sistemas de uso específico y como sistemas de uso general, siendo nuestra aplicación un sistema de uso específico por necesidades del proyector pero no por restricciones propias del diseño de elementos de procesamiento. Con esto, podemos llegar a conclusiones satisfactorias de lo que es el uso de los transputers para diversas aplicaciones , sin olvidar que la ventaja más palpable en este dispositivo es la comunicación entre otros dispositivos iguales, además, de su capacidad de procesamiento .

3.2 OCCAM

Durante el desarrollo de aplicaciones es necesario que los sistemas operativos faciliten el acceso a sistemas como el manejador de discos y terminales o facilidades para correr editores de texto , ensambladores , compiladores de alto nivel etc[2]. Un camino simple para proveer este tipo de herramientas es por medio de una máquina Host en el cual corra un programa servidor que comunique al sistema con la plataforma paralela (Figura 3.5). También se tiene la ventaja de que los compiladores y los sistemas de apoyo pueden estar escritos para correr en el transputer , mas que usar compiladores de cruce que solo correrán en la máquina HOST[1] . Todos estos requieren de ser soportados en un nuevo HOST que es relativamente simple para un programa servidor.

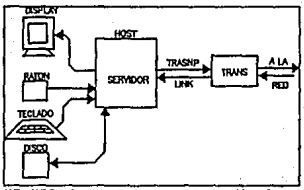


FIGURA 3.5 ESQUEMA DE SERVICIOS DE UN SISTEMA HOST

El primer sistema que fue capaz de hacer esto fue el INMOS Transputer Development System o TDS[2]. El TDS incluyó un servidor y un sistema integrado de OCCAM incluyendo un compilador , un editor de folders, un ligador y un configurador[2].

El servidor del TDS es un servidor más general llamado "iserver". El iserver correrá en cualquier sistema de computo, en el cual un transputer pueda ser interfaceado[2]. El servidor soporta un protocolo de manejo de pantalla, teclado y archivos del HOST. Este protocolo es usado para el manejo de varias herramientas que corran en el transputer. Solo el servidor , por ejemplo, necesitará leer un archivo que contenga una serie de rutinas en OCCAM las cuales darán la respuesta al servidor. El servidor localiza el archivo para llamar al S.O. del HOST y transmitir el programa por medio de links hasta el editor. El editor después usará otras rutinas de OCCAM , las que interactuarán con el servidor para desplegar caracteres en la pantalla del HOST y para tomar otros del teclado HOST.

El principal lenguaje provisto con el TDS fue el OCCAM , un lenguaje de alto nivel que fue diseñado para y con el transputer . Muchos documentos de INMOS describen a este como el mejor lenguaje posible para la programación del transputer y por algunos años este fue el único lenguaje capaz de aprovechar esto.

El nombre de OCCAM es derivado de William OCKAM (OCKAM 1270-1349), un filósofo inglés que creó el razonamiento de OCKAM el cual señala de una manera muy familiar "los derechos no deben de ser multiplicados sin razón" . El proverbio nunca fue dicho de esta forma por OCCAM ,el decía "es en vano hacer algo con más si puede ser hecho con menos"[2].

Como OCCAM fue diseñado para el transputer , su modelo de computo esta basado en procesos y sus relaciones de comunicación[4]. Las ideas básicas de comunicación entre procesos son varias , una de ellas es en la cual se tienen procesos sobre distintos espacios de memoria pero con la capacidad de alcanzar un nivel de memoria compartida en el momento en que dos procesos puedan leer el mismo espacio de memoria . Sin embargo , el espacio de memoria en el cual un proceso puede escribir, ningún otro puede escribir o leer sobre este mismo , puesto que, la única forma de comunicación de los procesos es por medio de canales. Estos canales actuarán solo cuando los procesos estén sincronizados.

Cuando un proceso manda un mensaje a otro , este último estará esperándolo hasta que lo reciba y si este mensaje no es mandado en forma adecuada o el proceso receptor no se encuentra listo en los tiempos adecuados , este último se quedará esperando indefinidamente.

Al igual que para un soporte explícito , para la programación concurrente en un solo transputer , el OCCAM incluye un lenguaje de configuración que soporta la capacidad de compilar y configurar el desarrollo de un sistema que correrá en una red de transputers.

Al nacer OCCAM del set de instrucciones del transputer , este tendrá una construcción formal y eficiente para el desarrollo de los sistemas que se puedan implementar con este tipo de elementos. No siendo así para otro tipo de lenguajes como por ejemplo FORTRAN ó C . Para este último el principal problema que se presenta es la construcción de una estructura que sea capaz de manejar en forma eficiente el paralelismo entre procesadores. Cabe señalar que las desventajas del OCCAM son principalmente en el manejo aritmético de estructuras algebraicas grandes , pues , como se verá mas adelante se requiere de cumplir con una serie de reglas que pueden llevar a cometer errores o generar procesos ineficientes en el tiempo.

Todo programa en OCCAM es estructurado en un SET de procesos secuenciales concurrentes con un estado interno de su propiedad el cual se comunica en instrucciones de entrada y salida por medio de canales de punto a punto[1]. Esta comunicación es sincronizada y no multiplexada. Los procesos que primero ejecutarán una instrucción de entrada y salida en un canal dado pasarán hasta que el patrón de comunicación ejecute el correspondiente comando de entrada y salida. El efecto de la ejecución de un par correspondiente de comandos de entrada y salida es la transferencia de algunos datos desde la salida a la entrada de procesos y la sincronización entre ellos. Todos los procesos ejecutan un cierto número de acciones y terminan. Esas acciones pueden ser ejecutadas secuencialmente como los lenguajes de programación convencional o en paralelo. En otras palabras, cada proceso puede ser estructurado como un set de procesos de comunicación concurrente o secuencial que están activados por un comando que se encuentra activo mientras estos trabajen.

Un programa en OCCAM tiene la estructura jerárquica paralela que está definida por la activación de procesos paralelos . Los procesos pueden ser construidos en un programa en OCCAM; el cual puede ser ejecutado en concurrencia en el mismo transputer, por medio de la capacidad de los recursos de tiempo compartido o en paralelismo real en una red de procesadores. Por otro lado, existen comandos especiales en OCCAM que pueden establecer una correspondencia entre la capacidad del hardware y la estructura lógica del programa , logrando una capacidad de ejecución en uno o mas transputers sin modificar el código fuente'.

En OCCAM todos los procesos están hechos de procesos "atómicos", llamados procesos primitivos . Los procesos primitivos son:

1) La asignación del valor de una expresión a una variable

variable := expresión

2) Una etiqueta de entrada, la cual esperará hasta que un valor sea recibido desde un canal y asignado a una variable

chanel ? variable

3) Una etiqueta de salida, la cual transmite el valor de una expresión al canal y espera hasta que el valor sea recibido

canal ! variable

4) Un proceso SKIP, el cual empieza y termina, sin realizar ninguna acción

5) Un proceso STOP el cual inicia, no realiza ninguna acción y nunca termina

Los procesos primitivos pueden ser combinados usando construcciones como secuencias (SEQ) , condicionales (IF), selecciones (CASE), loops (WHILE) , paralelismo (PAR), alternaciones (ALT). Mientras las primeras cuatro construcciones están presentes en casi todos los lenguajes convencionales, las construcciones paralelo (PAR) y alternación (ALT) son características de OCCAM y su existencia es el efecto de que en el modelo de programación del transputer ciertos procesos pueden ser ejecutados concurrentemente.

En una secuencia, los componentes del proceso son ejecutados uno tras otro , como una serie de reglas en cualquier otro lenguaje.

El condicional es parecido a una regla convencional IF. Este contiene un número de componentes , cada uno de los cuales es ejecutado por

una expresión booleana . Esas expresiones son evaluadas en secuencias y los procesos asociados a las primeras serán ejecutados si la expresión booleana es cierta. Si la expresión no es válida las expresiones que dependen de esta, no serán ejecutadas y se pasará a alguna otra expresión booleana para ser evaluada o en su defecto se saldrán de la construcción llamada IF. Un ejemplo de esto es el siguiente

```
IF
  X = 0
    Y := Y+1
  X <> 0
    SKIP
```

Una selección muy similar a la construcción llamada IF es el CASE. Ésta combina un número de procesos , cada uno de los cuales es asociado a algunos valores de una selección[1]. El proceso asociado a el valor de un selector es puesto y ejecutado conjuntamente con los procesos ligados a este. Por ejemplo , en la construcción

```
CASE número
  1,3,5,7,9
    even := FALSE
  0,2,4,6,8
    even := TRUE
```

La variable booleana even es cierta o falsa , dependiendo del valor que tome la variable entera llamada número.

Un loop es usado para repetir la ejecución de un determinado proceso hasta que una expresión booleana sea cierta[1]. Por ejemplo

```
WHILE  X > 5
  X := X-5
```

Los procesos que se componen de una construcción paralela serán ejecutados concurrentemente ; ellos empiezan juntos y pueden comunicarse por medio de operaciones de entrada y salida que son previamente definidas como canales. La construcción termina cuando todos los procesos han terminado.

```
PAR
  ch ! X
  ch ? Y
```

Una construcción puede ser priorizada y paralelizada (PRI PAR) . Ésto define una disciplina de jerarquía en los procesos componentes[1]. En las

implementaciones hechas por OCCAM es posible el uso de solo dos niveles diferentes de prioridad , es decir, un PRI PAR solo puede tener 2 procesos , el primero de ellos tendrá prioridad alta , mientras que el segundo prioridad baja .Los procesos de prioridad baja solo se ejecutarán cuando no se encuentre ningún otro proceso de alta prioridad ejecutándose . Ésto es, básicamente por la forma de trabajo de los transputers .

Ahora bien, una alternación (ALT) combina un número de procesos , cada uno de los cuales es guardado por un comando de entrada, una expresión booleana o una combinación de ellos . Un canal será visto para transmitir si el proceso de salida está listo para ejecutar la salida del canal . Una construcción ALT selecciona uno de los procesos el cual guarda una relación de activación compuesta de una expresión cierta y de una entrada a un canal cualquiera. Al encontrarse en estas condiciones se ejecuta la entrada también como los procesos guardados en la opción correspondiente. Si ningún proceso guardado esta listo, la alternación esperará hasta que alguno de ellos este preparado para ser ejecutado. La siguiente construcción

```
ALT
  ch1 ? X
    X := X+1
  ch2 ? Y
    Y := Y+1
```

será ejecutada hasta que ambos canales estén listos . Si solo uno de los dos canales esta listo , la entrada tomará en el canal que se encuentra preparado y se ejecutará el proceso asociado.

Se debe hacer notar que las entradas en una construcción ALT guardan algunas diferencias con otras construcciones en que inicialmente solo la condición del canal no es preguntada , mientras que aquí una (y solo una) de las entradas correspondientes ha de estar lista para guardar un valor al ser ejecutada[1]. Para la construcción paralela una construcción ALT puede ser priorizada (PRI PAR), definiendo el orden para la selección de alternativas listas al mismo tiempo.

Complementando todo lo anterior ,OCCAM provee los tipos de variables de datos mas comunes (BOOL, BYTE,INT,REAL32,REAL64,...). Todos estos tipos de datos primitivos pueden ser usados como componentes de arreglos multidimensionales . Las variables y arreglos deben de ser declaradas antes de ser usadas . Cada declaración aplica al proceso el mismo nivel de identificación que los inmediatamente siguientes. Las expresiones pueden ser construidas por combinación de variables y constantes lo cual significa un largo set de operadores lógicos, relacionales y aritméticos .

La más grande dificultad de OCCAM es probablemente la expresión de sintaxis . Cada expresión tiene un tipo y todas las variables y constantes en ésta ,deben de ser de éste tipo, porque la conversión implícita no es

provista en OCCAM[1] . Además, dado que no hay reglas de operadores precedentes cada par de operadores debe de ser puesto en brackets. Ésto significa , que para una expresión que contenga variables i(entera), x(real) las cuales en un lenguaje convencional deberán de ser escritas así

```
i:= (x*x+3.2)/2
```

en OCCAM tienen que ser

```
i := INT TRUNC(((x*x)+3.2(REAL32))/(2.0 (REAL32)))
```

Las desventajas son obvias . Si un programador no experto es forzado a considerar todos los tipos de conversiones es lógico que hará de una expresión un proceso no eficiente en la mayoría de los casos. La verosimilidad de la sintaxis es particularmente no deseable para la mayoría de la gente que conoce de álgebra.

Sin embargo, la gran ventaja de este lenguaje se haya en el uso de los canales y sus protocolos de comunicación. Rigurosamente hablando esos protocolos hacen posible el transmitir diferentes tipos de datos en el mismo canal físico. También debe de hacerse notar que las más recientes implementaciones del lenguaje hacen posible el uso de un protocolo anárquico (CHAN OF ANY) ,el cual da la capacidad de transmitir datos sin que haya una restricción sobre su tipo.

Al igual que otros microprocesadores , es posible, el acceso de un reloj procesador por medio de la declaración de TIMERS, los cuales son objetos del tipo primitivo[2]. Un timer trabaja en cualquier número de procesos concurrentes al accesar un reloj de solo lectura.

La entrada del timer , similar en forma, a la operación de entrada de un canal, regresa un valor de tipo INT que es representativo de un tiempo pasado . Dado que no se tiene una base de tiempo , se opta por trabajar en intervalos de lecturas , para que a la diferencia de estas, se pueda tomar una medida entre principio y fin de algún proceso en particular , como por ejemplo el hacer una serie de lecturas que después de haberse obtenido se hará una diferencia entre ellas para obtener así , un valor que será proporcional al tiempo transcurrido en la ejecución de un proceso o el tiempo que transcurrió entre la diferencia de timers.

Cabe señalar, que el timer no tiene un medida establecida , puesto que, trabaja en base a ticks generados por el procesador . Sin embargo, como se señalo anteriormente el timer esta íntimamente ligado con la selección de la prioridad de los procesos para la elección de la base con la cual se medirá el tiempo. Esto es, para un proceso de alta prioridad el timer generará un tick cada microsegundo ;mientras que; para un proceso de baja prioridad el timer generará un tick cada 64 microsegundos . Con base a este

marco de medidas y a la diferencia de lecturas se obtendrá el tiempo real del proceso que el usuario desea.

Dado que OCCAM es identificado como una programación de bajo nivel de transputers, éste contiene características que permiten al usuario especificar la posición que las variables ocuparán en la memoria del procesador y también de algún proceso en particular. A ésto se le conoce como localización (PLACE).

Como se verá posteriormente el mapa de memoria del hardware del transputer es BYTE direccionado , con direcciones signadas corriendo desde MimInt hasta MaxInt (Figura 3.6). El mapa de memoria de OCCAM es de forma diferente , las direcciones son palabras sin signo, corriendo desde cero hasta el tope del espacio de direccionamiento. Nosotros podemos usar la instrucción PLACE para asignar una dirección de memoria a una variable en base al mapeo de OCCAM y su contraparte en el hardware, como por ejemplo:

```
INT i:
PLACE i AT 300:
```

La localización de la variable entera en la memoria de hardware es la dirección 0x80000070 en un procesador de 32 bits o 0x8038 en un procesador de 16 bits[1]. Esto puede ser usado para la localización de variables en la memoria integrada al transputer o en la memoria externa a éste , claro está que ésto actuará guardando el compromiso de seguridad de OCCAM , en el momento en que dos procesos intenten accesar a la misma dirección sin ningún control.

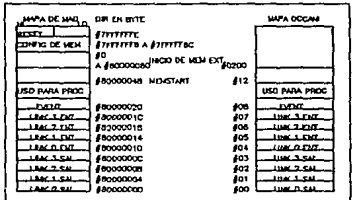


FIGURA 3.6 MAPAS DE MEMORIA

La localización de variables puede ser usada para el acceso de periféricos en memoria; sin embargo, esto es bastante inseguro ,dado que en cualquier momento más de algún proceso puede llamar a algún periférico de forma no controlada . Dado que el OCCAM no garantiza que el nombre de una variable no sea mapeado en alguna ocasión errónea en forma de lectura[2] , es mucho mejor trabajar con dispositivos mapeados en memoria, como puertos, los cuales son extensiones del concepto del canal en localidades de memoria[2]. Si un puerto es declarado y puesto en una dirección , este es leído y es escrito usando un canal de entrada y uno de salida:

```
PORT OF INT hola:
PLACE hola  AT #100000000:
INT i:
SEQ
  hola ? i
```

Este fragmento de código leerá el contenido de una localidad específica de memoria dentro de i. Esto garantiza que solo el acceso de lectura será por medio del canal hola.

Parte fundamental en el lenguaje OCCAM es la generación de redes de transputers con relativa facilidad (esto es conocido como configuración)[7]. Que es la localización de procesos en procesadores específicos. La configuración en el lenguaje OCCAM es en base a una extensión de la instrucción PAR, PLACED PAR. Ésta, solo puede ser usada en el nivel mas alto de un programa en OCCAM y especifica que un proceso es puesto en un procesador en particular, por ejemplo:

```
PLACE PAR
  PROCESSOR 0
    TAREA 0
  PROCESSOR 1
    TAREA 1
```

declara que la tarea 0 corre en el procesador 0 y la tarea 1 en el procesador 1 .

La construcción FOR puede ser usada con PLACE PAR , para asignar el mismo proceso a una serie de procesadores , como se muestra enseguida :

```
PLACE PAR i=0 FOR 10
  PROCESSOR i
    TAREA
```

Los links deben ser declarados fuera de la asignación de cualquier procesador. Este concepto es usado para atar procesos juntos al ser ejecutados en diferentes procesadores.

Si el proceso TAREA es definido con dos parámetros uno de entrada y uno de salida , entonces:

```
[11] CHAN OF INT32 c:
  PLACED PAR i=0 FOR 10
    PROCESSOR i
      TAREA(c[i],c[i+1])
```

Declara un proceso de TAREA de pipeline ejecutado en diferentes procesos, y pasando datos de uno a otro. Esos canales deben de ser hechos por su correspondiente link físico.

Con todo lo anterior, podemos dar la justificación del uso de OCCAM para el presente proyecto, en base a su capacidad para el manejo de transputer y su facilidad para el desarrollo de programación en paralelo.

3.3 ARQUITECTURA DEL TRANSPUTER

Hay dos fuertes influencias en la filosofía del diseño del transputer : el procesamiento paralelo y la tecnología RISC , conceptos que están ligados a los transputers al igual que las características de un microcontrolador teniendo un fuerte impacto sobre este dispositivo . La influencia del microcontrolador se presenta cuando el transputer puede darse de baja por si mismo, si el computo produce un error. Esto puede ser necesario si se produce un overflow o alguna anormalidad en el sistema. Otra influencia de los microcontroladores es que pueden ser capaces de servir a interrupciones externas rápidamente[7].

Las principales características del chip, sin embargo, son el soporte inherente para el procesamiento paralelo y la tecnología RISC. La influencia del procesamiento paralelo da al transputer mecanismos de comunicación que permiten que múltiples transputers sean como una sola máquina de cómputo con componentes necesarios para la transmisión de mensajes de un lado a otro. La tecnología RISC da un SET de instrucciones compacto pero bastante flexible en cualquier aplicación .

Esto significa que , debido al gran número de características específicas en el transputer (construcción interna de canales de comunicación llamados links , timers integrados , emsamblador para multiprocesamiento, etc), el set de instrucciones no es pequeño pero cada instrucción se ejercita solo en aspectos específicos de alguna parte del hardware. A diferencia de muchos microprocesadores de 32 bits, el transputer no provee un soporte para indexar el direccionamiento en una sola instrucción que se encargue de la dirección efectiva y sus contenidos de FETCH. El transputer tiene algunas instrucciones para computar la dirección

efectiva y el FETCH. En general, las instrucciones del transputer son menos funcionales que convencionales o CISC. La ventaja del RISC en el transputer es el aprovechamiento que se tiene sobre las instrucciones mismas , mientras menos poderosa se ejecuta mucho más rápido[7].

Además de las características orientadas al software ;el transputer tiene una interfase de memoria externa versátil con respecto al tipo de memoria que se quiera manejar. Para el caso de un sistema de multiprocesamiento es común usar memoria dinámica que como se verá en el siguiente capítulo su interfase con el transputer solo constará de unos cuantos circuitos integrados.

Las características atrás mencionadas son comunes para la familia de procesadores llamada transputers. Esta familia fue creada por INMOS en 1984 y consta de tres diferentes microprocesadores :

- EN 16 BITS T212
- EN 32 BITS T414
- EN 32 BITS T800

La Tabla 3.1, muestra las características generales de cada uno de los transputers:

TRANS	TAMAÑO PAL. (BIT)	MEM INT	DEBUGGER	RECON TEMP DE LINK	UNID. PTO. FTE	NUM. DE LINK	INTERRUP. EXT	BUS DIREC. DAT.
T805	32	4K	SI	SI	SI	4	SI	MUX
T800	32	4K	NO	SI	SI	4	NO	MUX
T425	32	4K	SI	SI	NO	4	SI	MUX
T414	32	2K	NO	NO	NO	4	NO	MUX
T400	32	2K	SI	NO	NO	2	SI	MUX

TABLA 3.1 FAMILIA DE TRANSPUTERS

Internamente un transputer consiste de una memoria , un procesador y un sistema de comunicaciones conectado vía un bus de 32 bits que también conecta a la interfase de memoria externa con capacidad de mapeo 4 gigabytes. El procesador , memoria y sistemas de comunicación ocupan cada uno cerca del 25% del total del área del silicio , el área restante es usada para la distribución de potencia , generación de relojes y conexiones externas[3].

El IMST800 , con su unidad de punto flotante , es solo 20% más grande en área que el sistema IMST414. El pequeño tamaño y alto rendimiento

vienen de un diseño el cual toma especial cuidado de la economía del silicio. Éste, es el principal contraste con los coprocesadores convencionales donde la unidad de punto flotante típicamente ocupa más área que un microprocesador y requiere un segundo chip.

EL diagrama a bloques (Figura 3.7) indica la forma en la cual están diseñados el IMST800 y el IMST414, mientras que, la Figura 3.8 muestra la distribución de servicios del IMST800 que será el elemento a usar en el diseño del sistema.

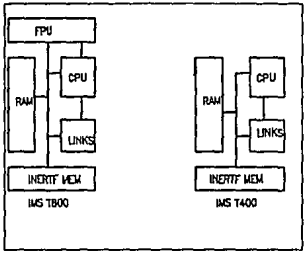


FIGURA 3.7 DIAGRAMAS A BLOQUES DE LAS ARQUITECTURAS DE LOS SISTEMAS T800 Y T414

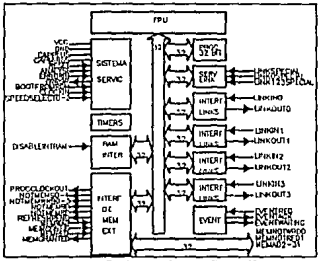


FIGURA 3.8 DIAGRAMA A BLOQUES DEL SISTEMA T800 CON TODAS SUS TERMINALES

El CPU de los transputers contiene 3 registros (A,B y C) usados para enteros y direcciones aritméticas , las cuales forman un stack de hardware cargando un valor en el stack y poniendo B dentro de C y A dentro de B , antes de cargar A, esto es, guardando un valor desde A. Leer un valor desde A, es poner B dentro de A y C dentro de B. De forma similar, la FPU incluye tres registros de evaluación de punto flotante contenidos en AF, BF y CF. Cuando los valores son guardados dentro o leídos desde, los registros de punto flotante actúan del mismo modo que los registros A,B y C. Este tipo de registros tiene como finalidad el dar flexibilidad al flujo de información entre los diferentes servicios del sistema y como fuente y destino de la mayoría de operandos aritméticos y lógicos.

Las direcciones de valores de punto flotante son formadas en el satck del CPU, y los valores están transferidos entre las localidades de direcciones de memoria y el satck del FPU sobre el control del CPU. Como el satck del CPU es usado solo en el hueco de las direcciones de los valores de punto flotante, el largo de la palabra del CPU es independiente de lo que se presenta en la FPU[7].

En suma el transputer tiene tres grandes registros:

1) APUNTADOR DE INSTRUCCIONES (PROGRAM COUNTER)

2) REGISTRO DE OPERANDOS.

3) APUNTADOR A ESPACIOS (SATCK POINTER)

El registro de operandos sirve como el punto focal para el procesamiento de instrucciones. Todas las instrucciones de transputer tienen el tamaño de un byte y típicamente se ejecutan en 1 o 2 ciclos de reloj. Los cuatro bits superiores del transputer contienen la operación en el rendimiento mientras que, los cuatro bits menores contienen un operando para la operación. Hay instrucciones especiales que forman grandes operandos esas son llamadas instrucciones de prefijo las cuales cargan sus cuatro bits de datos dentro del operando de registros y después corren al registro de operando cuatro lugares. Las instrucciones de prefijo negativo son similares a las anteriores, excepto que estas complementan al registro de operando antes de que este corra hacia arriba. Consecuentemente los operandos pueden ser extendidos tan largos como sean y el largo del registro de operando es capaz de construir una secuencia de instrucciones prefijas. Las instrucciones de prefijo consecutivas pueden ser usadas para llenar el registro de operando con cualquier valor nibble; a la vez se podrá usar el contenido del registro de operandos entero como un operando solo en el campo de operandos del flujo de la instrucción en cualquier valor de 32 bits con lo que podrá ser usado como operando de instrucciones .

3.3.1 NOTACION DE REGISTROS

Hay registros en el transputer que son usados en todos los programas; ellos son Iptr, Wptr,el stack A, B ,C y OREG.

Iptr.- Es un apuntador a instrucciones. Normalmente referido como un program counter, puesto que, apunta a la siguiente instrucción a ejecutar.

Wptr.- Apuntador a espacios . También es conocido como un program counter , dado que pone en un área especifica el almacenamiento de variables locales.

A,B,C .- Registros de propósito general

OREG .- Registro de operandos . Este registro es usado para ensamblar pequeños operandos de múltiples instrucciones dentro de un operando largo.

En síntesis nosotros podemos ver a los registros como lo muestra la Figura 3.9

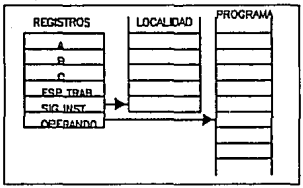


FIGURA 3.9 REGISTROS

3.3.2 BANDERAS

Hay tres tipos de banderas en el transputer Error,Halt on_Error y FP_Error

Error.- El transputer tiene una bandera de error similar a un overflow en microprocesadores convencionales , excepto que es "pegajoso", esto es, cada set mantiene su estado hasta que es específicamente borrado. La bandera de Error es puesta en activo al exterior por medio de una terminal externa del mismo nombre. Ésto es copiado de los microcontroladores logrando que otros transputers puedan detectar que alguno ha hecho un error de cálculo y tomar alguna acción apropiada.

Halt on_Error .- Esta bandera , cuando se activa, detiene al transputer si el error es detectado. Esta es probablemente una bandera de limpiado , pues se puede dar de baja al transputer y desactivar al error. Esta , también , es una influencia de los microcontroladores.

FP_Error .- Es la bandera que presentará un error de punto flotante.

3.3.3 UNIDAD DE PUNTO FLOTANTE

Existe la medición de soporte para la aritmética de punto flotante para IEEE en el T800 y en el T414 con estándar de 745-1985. El T414 tiene algunas instrucciones diseñadas de largo de 32 bits, las cuales soportan la aritmética de punto flotante . Esas instrucciones son implantadas en el microcódigo del CHIP.

El T800 no tiene esas instrucciones, tiene algo mucho más poderoso; una unidad de punto flotante . Los CPU más convencionales como son los intel 80X86 y MOTOROLA 680x0,tienen chips coprocesadores para aritmética de punto flotante (INTEL 80x87 y MOTOROLA 68881 y 68882) . INMOS a incorporado la unidad de punto flotante a un mismo chip en el T800.

La unidad de punto flotante T800 soporta aritmética de punto flotante IEEE para ambos tipos de valores (32 bits y 64 bits). Los registros de punto flotante son organizados en registros de tres pasos como los registros enteros. La máquina de punto flotante y la unidad entera pueden actuar autónomamente en cierto grado, mientras la unidad entera esta calculando una dirección ,la unidad de punto flotante puede trabajar en algún cálculo simultaneamente . Esta "ejecución de traslape" es una forma para maximizar el rendimiento del transputer.

3.3.4 FORMATO DE INSTRUCCIONES

Las instrucciones del transputer son de un solo byte de largo, sin embargo, pueden ser manejadas de varias formas hasta obtener un mejor rendimiento. Las instrucciones que son de un byte de largo son llamadas "instrucciones directas" y aquellas que sean de más de un byte serán llamadas "instrucciones indirectas" . Las instrucciones indirectas están compuestas de instrucciones prefijas que cargan un valor dentro del registro de operandos , seguidos por una instrucción que opera dentro de la ejecución actual.

Como se mencionó anteriormente, los cuatro bits más significativos de un byte de una instrucción determinan el código de esta, dando como resultado 16 posibles instrucciones. Solo 13 de estas instrucciones estarán comúnmente usadas por el programador del lenguaje ensamblador . Cada una de esas instrucciones requieren un operando , ésto es parcialmente arreglado por el campo del operando de 4 bits'.

De las otras 3 instrucciones , dos son instrucciones prefijas , las cuales son usadas para construir operandos más largos de 4 bits , mientras que, la instrucción de operación será usada para ejecutar el contenido del registro de operandos como una instrucción.

Muy comúnmente , 4 bits por operando son insuficientes , por esto es que el transputer tiene dos instrucciones especiales llamadas PFIX y NFIX que son usadas para manejar el registro de operandos . Cuando una de esas instrucciones es ejecutada, el registro de operandos es corrido 4 bits a la izquierda y después el operando de las instrucciones puesto en los cuatro

bits que fueron limpiados por el corrimiento . Usando esas cuatro instrucciones se puede usar el registro de operandos hasta con un valor de 32 bits.

3.3.5 ORGANIZACIÓN DE DATOS

La arquitectura del transputer provee un soporte inherente para enteros de 32 bits y valores más grandes , también como bytes o valores de caracteres; siendo más difícil de poder soportar valores cortos de 16 bits , como son en lenguajes de programación . Muchas instrucciones están generadas alrededor de palabras de 32 bits ; sin embargo; hay dos instrucciones LB (carga byte) y SB (guarda byte) que soportan palabras de 8 bits. Soportar cantidades de 16 bits debe de ser anulado para usar instrucciones de un byte y corrimientos. No hay niveles que soporten cantidades pequeñas de 16 bits[7].

Por ejemplo, el cargar una cantidad de 16 bits , primero se debe de cargar un byte, después correr a la izquierda 8 bits y sumarle el siguiente byte.

3.3.6 DIRECCIONAMIENTO

Las direcciones en el transputer son de una palabra larga , teniendo un espacio de direcciones continuas de 4 Gigabytes para el T414 y el T800. El bus de direcciones es multiplexado con el bus de datos. Una interesante característica del transputer es que las direcciones son signadas , esto es, si el bit mas significativo de un set de direcciones es 1 la dirección es negativa , mientras que , si el bit es cero la dirección será positiva.

El transputer tiene 2 formas básicas de direccionamiento de datos en memoria: (1) operaciones en el stack y (2) métodos de procesos aleatorios. Ambas técnicas son válidas , pero las operaciones en el stack son más rápidas , que los métodos de acceso aleatorio aunque son mas convencionales para el uso del programador . En cualquier caso el transputer es rápido de accesar palabras en memoria que en bytes. Importante: Muchas instrucciones requieren espacios de direccionamiento de memoria en palabras límites (múltiplos de 4)[7].

3.3.7 MICROCODIGO

Los procesos que corren en el transputer pueden tener cualquiera de los siguientes pasos de ejecución:

1.- RUNNING

2.- ESPERANDO UNA COLA A SER EJECUTADA

3.- ESPERANDO POR UNA ENTRADA

4.- ESPERANDO POR UNA SALIDA

5.- ESPERANDO POR UN CIERTO VALOR DEL TIMER

Si el estado es "corriendo" o "esperando en una cola" el proceso estará activo. Si el estado es cualquiera de los tres restantes , el proceso estará inactivo.

Los procesos del transputer pueden correr en dos prioridades ; alta y baja. En prioridad alta un proceso pueden interrumpir a un proceso de prioridad baja en cualquier manejo. Los procesos de prioridad alta continuarán corriendo ,bajo las siguientes condiciones:

1.- HASTA QUE ESTE COMPLETAMENTE TERMINADO

2.- HASTA QUE TENGA UN TIEMPO DE ESPERA PARA COMUNICACIÓN

3.- HASTA QUE TENGA UN TIEMPO DE ESPERA PARA UN CIERTO VALOR DE L TIMER

El cambio de procesos de uno a otro es comúnmente referido a un contexto de switcheo.

3.3.8 COMUNICACION

Un importante concepto para la programación paralela es la capacidad de conectar múltiples transputers juntos , en forma de una red de procesadores . La conexión entre procesadores conlleva a que los transputers se comuniquen información y compartan datos. Cada transputer tiene cuatro puertos seriales bidireccionales llamados links. Un link en un transputer puede ser conectado a un link de cualquier otro transputer. Aparte de esto , existe la ventaja de que cada proceso puede escribir o leer en un cierto espacio de memoria llamado canal[7]. Los canales pueden ser usados para la comunicación entre procesos de un solo transputer o pueden ser usados para comunicar por medio de LINKS a procesos entre transputers.

Esos canales internos son usados exactamente de la misma forma que los canales externos . La única diferencia es que cualquier localidad de memoria trabajará en lugares de localidades reservadas usadas por los links[7].

Los links han sido asignados a las localidades internas de la memoria del chip (Tabla 3.2). Al mandar un mensaje a otro transputer se debe usar una de esas localidades reservadas . Esas localidades corresponden directamente a terminales en el transputer que estarán conectadas físicamente a otro transputer o a circuitos de switcheo que finalmente se conectarán a otro transputer. Recuerde que cada entrada y salida es del tamaño de un byte.

Los programas de comunicación a través de un link ejecutarán una entrada o una salida. Los parámetros de esas instrucciones son las direcciones de inicio de memoria y el largo del bloque de los datos. Cuando se ejecuta la entrada o salida por medio de link se invoca a toda una máquina link.

LINK	DIRECCIÓN EN HEX
LINK 3 ENTRADA	0x8000001C
LINK 2 ENTRADA	0x80000018
LINK 1 ENTRADA	0x80000014
LINK 0 ENTRADA	0x80000010
LINK 3 SALIDA	0x8000000C
LINK 2 SALIDA	0x80000008
LINK 1 SALIDA	0x80000004
LINK 0 SALIDA	0x80000000

TABLA 3.2 LOCALIZACIÓN DE LINKS EN EL MAPA DE MEMORIA

Los links de hardware actúan autónomamente ,desde que el procesador toma un dato desde memoria pasa sobre el canal del link. Después de cada byte mandado , el transputer receptor mandará un reconocimiento y guardara el dato en el bus del proceso recibido . Si no hay ningún proceso recibido , el transputer receptor no reconocerá el byte, mientras que el transputer transmisor esperará por un reconocimiento.

Cuando el mensaje ha sido transferido ,los procesos de comunicación son puestos sobre un proceso de colas. Como resultado, la máquina del link interno roba ciclos desde el CPU.

Cada interfase de los links usa 3 registros:

1.- UN APUNTADOR A UN ESPACIO

2.- UN APUNTADOR A MENSAJE

3.- UN CONTADOR DE BYTES EN EL MENSAJE

En la Figura 3.10 los procesos P y Q ejecutados por diferentes transputers se comunican por medio de un canal C implementado por un link conectando 2 transputers. Poutput y Qinput.

Cuando P ejecuta su instrucción de salida , los registros en la interfase de link del transputer son inicializados y P es desprogramado. Similarmente ocurre cuando Q ejecuta su instrucción de entrada , los registros en la interfase de los links de los procesos que esta ejecutando Q son inicializados y Q es desprogramado para volverse a ejecutar con la instrucción de salida, Figura 3.11.

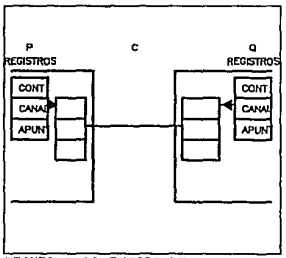


FIGURA 3.10 INICIO DE COMUNICACION

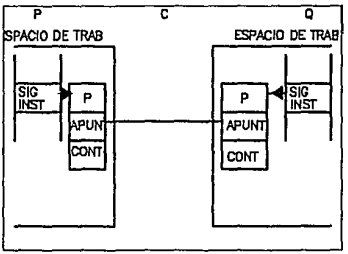


FIGURA 3.11 RECONOCIMIENTO FINAL DE COMUNICACION

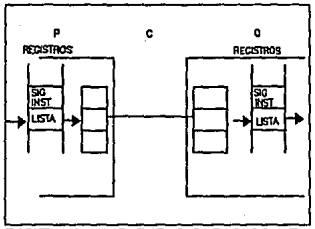


FIGURA 3.12 REINICIALIZACION DE LINKS PARA LA COMUNICACIÓN

El mensaje, es ahora copiado por el link , después los espacios P y Q están en las correspondientes listas de programación (Figura 3.12).El protocolo usado en P y Q define la no importancia de quienes están en las correspondientes listas de programación (Figura 3.12). El protocolo usado en P y Q define la no importancia de quién se encuentra listo en primera instancia.

Ahora bien , se requiere de un protocolo que multiplexe información y datos de control. Los mensajes son transmitidos como una secuencia de bytes , cada uno de los cuales deben de ser reconocido antes de que el siguiente sea transmitido. Un byte de datos es transmitido con dos bits de inicio seguidos por ocho bits de información y por un bit de parada (Figura 3.13a). El reconocimiento es transmitido con un bit de inicio seguido de un bit de parada . Un reconocimiento indica a ambos procesos que fue capaz de recibir los datos del byte mandado y que el bus es capaz de transmitir otro byte[6] (Figura 3.13b).

El protocolo permite un reconocimiento , que demuestra al transmisor que el receptor ha identificado un paquete de datos. De esta forma , el reconocimiento puede ser recibido por el transmisor antes de que todo el paquete de datos haya sido transmitido y el transmisor pueda enviar el siguiente paquete de datos. Algunos transputers no incluyen este traslape e incluyen suficiente bus en el hardware del link , la relación puede ser más que doblada al archivar 1.8 MBYTES/SEG en una dirección y 2.4 MBYTES/SEG cuando los links acarrean datos en ambas direcciones . La Figura 3.14 muestra las señales que deben de ser observadas en las líneas de links cuando el paquete de datos es traslapado con un reconocimiento.

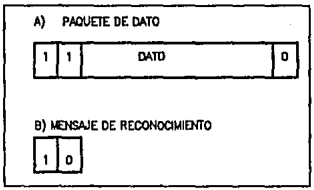


FIGURA 3.13 PROTOCOLO DE COMUNICACION ENTRE CANALES

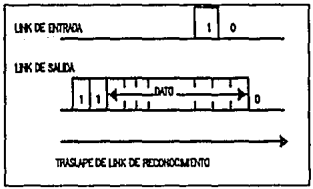


FIGURA 3.14 FORMA DE TRASLAPE EN LA COMUNICACION

3.3.9 CANAL EVENT

Además de los links antes mencionados , hay otro mecanismo de hardware que actúa como un canal . La entrada externa event utiliza en el transputer la localidad 0x80000020 . Siempre que event reciba una señal , cualquier proceso esperará que este canal se deshabilite . Este es el equivalente a una interrupción de un microprocesador normal. En el sentido en el que el canal deberá esperar la ejecución de entrada mientras la bandera de event esté dada.

3.3.10 PROCSPEEDSELECT 0-2

La velocidad del procesador IMS T800 es variable en pasos discretos. La velocidad puede ser seleccionada en base a la máxima relación para un componente particular. Para las tres líneas de selección de velocidad Procspeedselect 0-2 las terminales manejan los siguientes parámetros de velocidad como lo indica la Tabla 3.3.

PROCSPEEDSELECT2	PROCSPEEDSELECT1	PROCSPEEDSELECT0	RELOJ EN MHz	RELOJ EN ns	NOTAS
0	0	0	20.0	50.0	
0	0	1	22.5	44.4	
0	1	0	25.0	40.0	
0	1	1	30.0	33.3	
1	0	0	35.0	28.6	
1	0	1			INVALIDA
1	1	0	17.5	57.1	
1	1	1			INVALIDA

TABLA 3.3 SELECCIÓN DE LA VELOCIDAD

Solo seis posibles combinaciones están siendo usadas . Ahora bien la frecuencia de reloj de entrada debe de ser de 5MHZ para que se obtenga la velocidad de operación mostrada en la Tabla 3.3.

3.3.11 RESET

El RESET debe de darse en alto con VCC pero no debe de exceder el voltaje específico llamado VIH. Después de que VCC es válido clockin debe de estar corriendo con un período mínimo TDCVRL antes del final del RESET[6]. Al momento de caída del RESET se inicializará el transputer ; se disipará la secuencia de configuración de memoria e inicia la rutina de bootstrap . Los links de salida están forzados en bajo durante el RESET ; los links de entrada y EVENREQ deberán estar en bajo. Memory Request (DMA) no debe ocurrir mientras RESET este en alto pero puede ocurrir antes del BOOTSTRAP.

Después del final del RESET habrá un retraso de 144 períodos de reloj de entrada (Figura 3.15). Siguiendo esto MemWrD0 , MemRfD1 y Mem A0-31 serán escaneados al checar la existencia de una memoria programada en la interfase de configuración . Después de revisar la configuración interna y externa , ocurrirá un retraso de tiempo, que su período dependerá de la configuración actual. Finalmente ocho ciclos completos de refresco ocurrirán ,para después inicializar la RAM dinámica , usando la nueva configuración de memoria . Si la configuración de memoria no necesita refresco de la RAM dinámica los ciclos de refresco serán reemplazados por un retardo equivalente sin actividad de memoria externa.

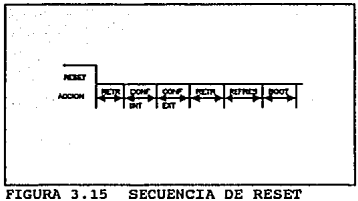


FIGURA 3.15 SECUENCIA DE RESET

3.3.12 BOOTSTRAP

El transputer puede ser inicializado por un link o por una memoria externa . La facilidad del manejo de BootFromRom permite esto, logrando un cambio dinámico pero se debe de tener cuidado con las restricciones de los tiempos. Esta inicialización solo ocurre para el transputer antes de que la primera instrucción sea ejecutada y después de que el reset este en bajo.

Si BootFromRom está conectado en alta el transputer empezará a ejecutar el código proveniente de dos bytes de la memoria externa , en las direcciones #7FFFFFFE .Esta localidad deberá contener un respaldo de un programa en ROM. Siguiendo este acceso, BootFromRom debe ser tomado bajo si se requiere teniendo al procesador en prioridad baja y el registro de operando en MemStart.

3.4 REFERENCIAS

1.- CARLINI DE U , VILLANO DE U, TRANSPUTER AND PARALLEL ARCHITECTURE, CHESHIRE SIGMA, 1988

2.- GRAHAM IAN, KING TIM, THE TRANSPUTER HANDBOOK,PRENTICE HALL, SEGUNDA EDICION, 1989

3.- HARP GORDON, TRANSPUTER APLICATIONS, PITMAN, COMPUTER SYSTEMS SERIES,1989

4.- HOARE C.A.R., COMUNICATING SEQUENTIAL PROCESSES, PRENTICE HALL,1989

5.- HWANG KAI, PARALLEL PROCESSING FOR SUPERCOMPUTERS AND ARTIFICIAL INTELIGENCE, MC GRAW HILL, 1989

6.- INMOS,TRANSPUTER DATABOOK, INMOS,1990

7.- ROBERTS JOHN, THE TRANSPUTER ASSEMBLY PROGRAMMING, PRENTICE HALL, 1990.

CAPITULO IV

DISEÑO DEL SISTEMA DE PROCESAMIENTO PARALELO

4.1 INTRODUCCION

La familia de microprocesadores llamada transputers desarrollada por INMOS ha formado un campo estándar dentro de la industria del multiprocesamiento. La familia consiste en un rango de poderosos dispositivos VLSI que tienen la misma arquitectura básica como cualquier otro microprocesador, además de una interfase de memoria y 4 puertos seriales de comunicación llamados links (ligas).

Los sistemas de multiprocesamiento pueden ser construidos en base a un serie de transputers operando concurrentemente y comunicándose a través de links.

Para este propósito INMOS creó el concepto de TRAM que fue introducido en 1987 y que buscaba explotar al máximo los beneficios que pudieran dar sus puertos de comunicación. La arquitectura TRAM ofrece una serie de ventajas sobre sistemas de configuración convencional, como se verá en el siguiente punto.

En el presente capítulo daremos una explicación del proceso de diseño del sistema en base al concepto TRAM. El sistema constará de dos módulos de transputers (FIGURA 4.19) conectados con un sistema monitor que será a su vez la máquina HOST. El sistema monitor estará compuesto por la PC y un sistema de desarrollo llamado TEK (véase Apéndice A) , el que tendrá como función trabajar como transputer maestro (véase Apéndice C) de las aplicaciones que se ejecutan en el sistema.

El sistema TEK (TRASNPUTER EDUCATION KIT) ha sido usado desde varios puntos de vista, en este trabajo aún cuando su principal aplicación es servir como sistema monitor de los módulos de procesamiento. Inicialmente el TEK presenta una serie de herramientas en su software (OCCAM y C paralelo) que en conjunto con el sistema de desarrollo nos permite simular cualquier programa antes de ser probado en los módulos, además este sistema puede ser usado como parte de una red de transputers sin necesidad de que este sea el maestro. Existen diversas aplicaciones que nos presentan este tipo de sistemas pero como se menciono anteriormente será usado con la finalidad de un sistema monitor.

4.2 DESCRIPCION DE UN SISTEMA LLAMADO TRAM

En el capítulo anterior se señalaron las ventajas y desventajas del Transputer; ahora bien para que este dispositivo actúe como parte de un sistema, se requiere de una serie de elementos que le permitan interactuar de una forma más eficaz y estándar en una serie de aplicaciones encaminadas al paralelismo. La necesidad del desarrollo de un estándar , trajo consigo la definición de un tipo de tarjetas que cumplan con una serie de requerimientos , para la construcción de redes en paralelo, las cuales se agrupan en módulos de transputers (o sus siglas en inglés TRAM).

Este tipo de tarjetas; a grandes rasgos cumplen con los siguientes lineamentos:

1.- DIMENSIONES
2.- ESTANDAR DE TERMINALES
3.- DESIGNACION DE VELOCIDADES
4.- CAPACIDAD DE MEMORIA

Para el primer punto se definió el siguiente estándar (Tabla 4.1)

TAMAÑO	DISTANCIA ENTRE TERMINALES (IN)
1/4	0.6
2/4	0.6
1/2	1.5
1	3.66
2	3.66
4	3.66
8	3.66

TABLA 4.1 ESTANDAR DE DIMENSIONES DE SISTEMAS TIPO TRAM

Mientras que, para el estándar de terminales , se definió , aquel que se muestra en la tabla 4.2;

NOMBRE DE TERMINALES	POSICION ESTANDAR	NOMBRE DE TERMINALES	POSICION ESTANDAR
LINK2OUT	1	ANALYSE	9
LINK2IN	2	RESET	10
VCC	3	NOTERROR	11
LINK1OUT	4	LINK0OUT	12
LINK1IN	5	LINK0IN	13
LINKSPEEDA	6	GND	14
LINKSPEEDB	7	LINK3OUT	15
CLOCKIN	8	LINK3IN	16

TABLA 4.2 ESTANDAR DE TERMINALES DE LOS SISTEMAS TIPO TRAM

que es completamente independiente del tamaño de la tarjeta.

Como se puede ver, las señales que se presentan a la salida son las mínimas posibles; VCC, GND, RESET, ANALYSE, NOTERROR, CUATRO LINKS (bidireccionales) y LinkSpeedA/LinkSpeedB. Las dos últimas señales tendrán dos modos de operación; si es bajo, los links bidireccionales operarán a 10MBps y si los links A y B están en alto los bidireccionales operarán a 20 MBps. Otras combinaciones están reservadas para futuros crecimientos , mientras que, la velocidad de 5 MBps ha sido desechada.

Por último, tenemos que mencionar a la capacidad de memoria , que podrá ser independiente del tamaño de los TRAMs y solo dependerá de las aplicaciones que se deseen ejecutar en una red de paralelismo.

Ahora bien, este tipo de módulos constituyen el soporte principal de cualquier sistema paralelo en base a transputers, sin importar la versión de transputers que podrá montarse en este tipo de tarjetas, pues, la compatibilidad entre los transputers (IMS T414,T800,T805) es 100% posible, dado que, sus terminales son las mismas y en aquellos casos, donde una terminal no tenga uso para un transputer de una generación baja, para el siguiente, será ocupada de alguna forma que no afecte al sistema y en cambio, lo beneficie.

Este avance , permitió implementar el desarrollo en forma modular no importando el tipo de transputer, el tamaño o la capacidad de memoria en que se trabajará. Habrá que tomar en cuenta que este tipo de módulos se tomaron como base para el diseño que a continuación se explica.

Habrá que señalar que el uso de un reloj externo en un TRAM es indispensable, mientras que, para el sistema propuesto el reloj estará integrado en el diseño de cada medio.

4.3 DESCRIPCION DEL SISTEMA DISEÑADO

Antes que nada hay que establecer los alcances y expectativas del proyecto, en la búsqueda por paralelizar procesos tomando como base un sistema de memoria distribuida que actué como esclavo de la máquina HOST.

Al definir a este sistema como esclavo, tendremos que , todos los módulos que lo integran estarán , sujetos a la máquina HOST (en este caso el TEK), sin que ninguno pueda tener control sobré algún otro módulo externo.

Para los requerimientos del proyecto se necesita de una capacidad cercana a los 256k BYTES y una velocidad de 20 MHz, por lo que, podemos suponer que las memorias requeridas serán del tipo DRAM (Dada la velocidad).

4.4 INTERFASE DE MEMORIA

Los transputers T414 y T800 tienen la capacidad de configuración de la interfase de memoria de una forma bastante flexible la cual les permite soportar una serie de elementos de memoria , desde ROMs hasta DRAMs sin importar sus velocidades de respuesta.

El T414 y T800 tienen 32 bits multiplexados de direcciones y datos dando un espacio total de 4Gbytes. Hay, también, cuatro señales de escritura , una señal de lectura, una señal de refresco , 5 señales configurables, una señal de entrada de espera, una señal de entrada de configuración de memoria, una señal de entrada de chequeo del bus y una salida de validación del bus[2]. La Figura 4.1 muestra las entradas y salidas , mientras que la Figura 4.2 nos muestra un bosquejo sobre el diagrama de tiempos que vamos a trabajar.

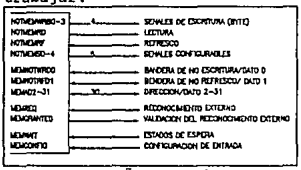


FIGURA 4.1 SEÑALES GENERADAS POR EL TRANSPUTER

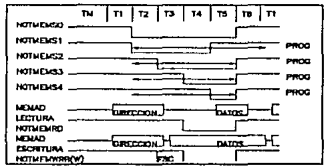


FIGURA 4.2 SINCRONIZACION DE SEÑALES PARA MANEJO DE MEMORIA

Como se menciono en el capitulo anterior , los transputers manejan un espacio de memoria signado que tiene una compatibilidad estricta con su lenguaje de programación OCCAM. Como se muestra en la Figura 4.3.

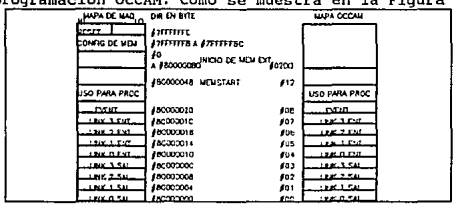


FIGURA 4.3 MAPA DE MEMORIA DEL TRANSPUTER Y OCCAM

Retomando lo señalado en los capítulos II y III el T414 tiene 2Kbytes de memoria interna, mientras que, el T800 tiene 4 Kbytes . Esto nos conduce ha decir que si existe memoria externa y memoria interna en la misma dirección , el transputer se decidirá por la memoria interna.

4.5 TIEMPOS DE INTERFASE DE MEMORIA

Los ciclos de interfase de memoria del T800 tiene 6 estados de tiempos referidos a Tstates y a un período de Clockout. Los Tstates tienen las siguientes funciones nominales:

TSTATE	
T1	APARACION DE LA DIRECCION ANTES DE LA SEÑAL DE VALIDACION
T2	TIEMPOS DE ESPERA DESPUES DE LA SEÑAL DE VALIADACION
T3	CICLO DE LECTURA /TRIESTADO/ CICLO DE ESCRITURA
T4	ESTADOS DE ESPERA
T5	LECTURA O ESCRITURA DE DATOS
T6	FINAL DE TRISTATE

La duración de cada Tstate puede ser configurable a la respuesta da cada dispositivo de memoria usado y puede ser desde uno hasta 31 Tm . Un Tm es la mitad del ciclo del tiempo del procesador ; la mitad del período de clockout; Figura 4.4 Tabla 4.3 . Si la cantidad total configurada de períodos Tm es un número impar , un período extra Tm será adicionado al final de T6 forzando a iniciar al siguiente T1 para coincidir con el inicio de Proclockout[1].

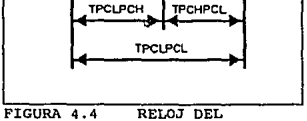


FIGURA 4.4 RELOJ DEL TRANSPUTER CLOCKOUT

SIMBOL.	PARAM.	MIN	NOM.	MAX	UNIDAD	NOTAS
TPCLPCL	PERIOD DE CLK	a-2	a	a+2	ns	1,3
TPCHPCL	PULSO ALTO	b-11	b	b+3	ns	2,3
TPCLPCH	PULSO BAJO		c		ns	3,3

TABLA 4.3 TIEMPOS DE INTERFASE DE MEMORIA DEL TRANSPUTER

NOTAS
1.- a es TDCLDCL/PROCCLOCKOUT
*2.- b es 0.5*TPCLPCL (mitad del periodo de reloj)*
3.- c es TPCLPCL-TPCHPCL
4.- Este parámetro es muestreado y no es 100% probado

Para facilitar el control de diferentes tipos de memoria y dispositivos periféricos, la interfase de memoria externa esta provista por 5 señales de salida (NotmemS0-4) , cuatro de las cuales pueden ser configuradas por el usuario. Las señales son convencionalmente asignadas a las funciones de los ciclos de diagramas de lectura y escritura.

Notmems0 es una señal de formato; iniciando con T2 y siempre terminado con T5. El inicio de Notmems1 siempre coincide con T2 y su final puede ser configurado hasta 31 períodos de Tms . NotmemS2, NotmemS3 y NotmemS4 son idénticos en operación . Todos ellos terminan al final de T5 , pero el inicio de cada uno de ellos puede ser retrasado desde uno hasta 31 períodos Tm. Esta programación puede llegar a tal grado que no se activaran las señales, como se muestra en la Figura 4.5.

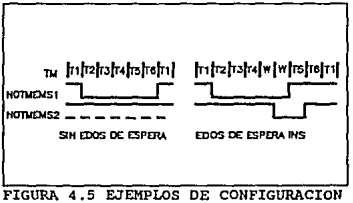


FIGURA 4.5 EJEMPLOS DE CONFIGURACION DE TIEMPOS DEL TRANSPUTER

Se pueden manejar ciclos de espera con bastante facilidad dependiendo solo, de la terminal Memwait. Esta señal, cuando se va a alto se insertarán tantos tiempos de espera como permanezca en alto[2].

La configuración de la interfase con memoria tiene 17 posibles configuraciones pre-programadas (para el T800) que son mostradas en la Tabla 4.4 que dan como resultado los ejemplos mostrados en la Figura 4.6.

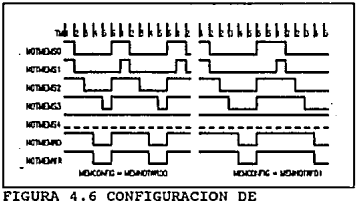


FIGURA 4.6 CONFIGURACION DE MEMORIA POR MEMCONFIG

Esta configuración depende de la terminal memconfig y de su conexión con alguna de las señales de direccionamiento mostradas en la Tabla 4.4 [3].

	DURACION DE CADA PERIOD EN TM						CICLO DE ESCRITURA	INTERVALO DE REFRESCO
TERMINAL	T1	T2	T3	T4	T5	T6	TIPO	CICLO DE RELOJ
MEMNOTWRDO	1	1	1	1	1	1	RAPIDO	72
MEMOTRFD1	1	2	1	1	1	2	LENTO	72
MEMAD2	1	2	1	1	2	3	LENTO	72
MEMAD3	2	3	1	1	2	3	LENTO	72
MEMAD4	1	1	1	1	1	1	RAPIDO	72
MEMAD5	1	1	2	1	2	1	RAPIDO	72
MEMAD6	2	1	2	1	3	1	RAPIDO	72
MEMAD7	2	2	2	1	3	2	RAPIDO	72
MEMAD8	1	1	1	1	1	1	RAPIDO	72
MEMAD9	1	1	2	1	2	1	RAPIDO	+
MEMAD10	2	2	2	2	4	2	LENTO	72
MEMAD11	3	3	3	3	3	3	LENTO	72
MEMAD18	2	1	2	3	3	3	RAPIDO	72
MEMAD31	4	4	4	4	4	4	LENTO	72

+ SOLO PARA MEMORIA ESTATICA

TABLA 4.4 CONFIGURACION DE TIEMPOS PARA EL TRANSPUTER

Como se puede ver, la ventaja que se obtiene de esto es la facilidad en el manejo de las señales de configuración sin la necesidad de circuitos externos.

Con todo lo manejado anteriormente se pueden proponer una serie de diagramas para lectura y escritura dinámica como los mostrados en las Figuras 4.7 y 4.8 a reserva de los tiempos que marquen las propias memorias y las velocidades del transputer como lo muestra la Tabla 4.6 correspondiente a las Figuras 4.7 y 4.8. A partir de lo señalado en el inicio de este capítulo, el proyecto requiere de una capacidad de memoria de 256K bytes y lo suficientemente rápida para responder a la velocidad de 20 MHz del transputer. En base a esto y tomando encuentra las facilidades de configuración que puede tener el transputer , para su interfase de memoria, se eligió la familia de memorias dinámicas mostrada en la Tabla 4.5.

TIPO	TIEMPO DE ACCESO RAPIDO	DISIPACION DE BAJA POTENCIA
MCM514256A-70	70ns	440mW
MCM514256A-80	80ns	385mW
MCM514256A-10	100ns	330mW

TABLA 4.5 SELECCION DE TIPO DE MEMORIAS A USAR

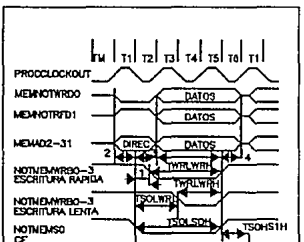


FIGURA 4.7 CICLO DE ESCRITURA

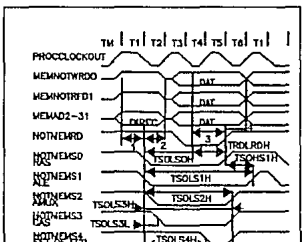


FIGURA 4.8 CICLO DE LECTURA

PARAMETRO	SIMBOLO	MIN	MAX	UNIDAD	NOTA
INIC. CAT ANTES LEER	TDVRDH	25		ns	
FIN DAT DESP LEER	TRDHDX	0		ns	
NOTMEMS0 ANTES INIC LEER	TSOLRDL	a-4	a+4	ns	1
FIN LEER DESDE FIN NOTMEMS0	TSOHRDH	-4	4	ns	
PERIODO LEER	TRDLRDH	B-3	B+3	ns	2
PULSO BAJO NOTMEMS0	TSOLSOH	C-5	C+6	ns	3
NOTMEMS1 Y NOTMEMS0	TSOLS1L	-4	4	ns	
NOTMEMS1 FIN DES NOTMEMS0	TSOLS1H	D-1	D+9	ns	4,6
NOTMEMS1 FIN FIN NOTMEMS0	TSOHS1H	E-8	E+8	ns	5,6
RET NOTMEMS1 DES NOTMEMS0	TSOLS2L	F-6	F+5	ns	7
NOTMEMS1 FIN FIN NOTMEMS0	TSOHS2H	-4	7	ns	
FIN NOTMEMS1 DES NOTMEMS0	TSOLS2H	C-5	C+7	ns	3
RET NOTMEMS1 DES NOTMEMS0	TSOLS3L	F-6	F+5	ns	7
FIN NOTMEMS1 DES NOTMEMS0	TSOLS3H	C-5	C+7	ns	3
FIN NOTMEMS1 FIN NOTMEMS0	TSOHS3H	-4	7	ns	
RET NOTMEMS1 DES NOTMEMS0	TSOLS4L	F-6	F+5	ns	7
FIN NOTMEMS1 DES NOTMEMS0	TSOLS4H	C-5	C+7	ns	3
FIN NOTMEMS1 FIN NOTMEMS0	TSOHS4H	-4	7	ns	
INIC DAT ANT ESCRIBIR	TDVWRH	D-7	D+10	ns	8,12
FIN DAT DESP ESCRIBIR	TWRHDX	A-10	A+5	ns	8,9
NOTMEMS0 INIC DE ESC. TEMP	TSOLWRL	B-5	B+5	ns	8,10
NOTMEMS0 INIC DE ESC. TARD	TSOLWRL	C-5	C+5	ns	8,11
FIN DE ESC DES FIN NOTMEMS0	TSOHWRH	-5	4	ns	8

TABLA 4.6 TIEMPOS DE ESCRITURA Y LECTURA DEL TRANSPUTER

NOTAS

1.- a es T1 donde T1 puede ser desde uno hasta 4 periodos Tm de largo

2.- b es T2 donde T2 puede ser desde uno hasta 4 periodos Tm de largo

3.- c es la suma de T2+T3+T4+Tespera+T5 donde T2,T3,T4,T5 pueden ser desde uno hasta cuatro periodos Tm, mientras que, Tm pueder ser de cualquier número de periodos Tm

4.- d puede ser desde cero hasta 31 periodos Tm de largo

5.- e puede ser desde -27 hasta +4 periodos Tm de largo

6.- si la configuración causara una llamada a los tiempos de espera, ésta dará al final de cada ciclo T6 y reiniciará con un estado alto en T1

7.- f puede ser desde cero hasta 31 periodos Tm de largo. Si el largo del ciclo sobre pasara a T6, el final de este se daría al fina del próximo T5 con un estado alto

8.- g es un ciclo completo de memoria externa comprendido por T1+T2+T3+T4+Tespera+T5+T6 donde T1,T2,T3,T4,T5 pueden ser desde una hasta cuatro periodos Tm de largo, T6 puede ser desde una hasta 5 periodos Tm de largo y Tespera puede ser cero o cualquier número de periodos Tm

9.- tiempo para que todas las señales de escritura se den notMemwrb0-3

10.- a es T6 donde T6 puede ser desde una hasta 5 periodos Tm

11.- b es T2 donde T2 puede ser desde una hasta 4 periodos Tm

Estas memorias se distinguen por su rapidez de acceso y su baja potencia de disipación y tienen como principal característica el manejo de datos por medio de matrices, es decir, por medio del direccionamiento de renglones y columnas como lo muestra la Figura 4.9[5].

PARAMETRO	SIMBOLO	MCM514256A-70		MCM514256-80		MCM514256-10		UNIDAD
		MIN	MAX	MIN	MAX	MIN	MAX	
CICLO DE TIEMPO DE ESC O LEC. ALEAT.	TRC	130	-	150	-	180	-	ns
TIEMPO DE ACC. RAS	TRAC	-	70	-	80	-	100	ns
TIEMPO DE ACC. CAS	TCAC	-	20	-	20	-	25	ns
TIEMPO DE ACC. DIR COLUM.	TAA	-	30	-	40	-	50	ns
SALIDA BUFFER RETARD. T.OFF	TOFF	0	20	0	20	0	20	ns
TIEMPO PRECARGA RAS	TRP	50	-	60	-	70	-	ns
PULSO RAS	TRAS	70	10000	80	10000	100	10000	ns
PULSO CAS	TCAS	20	10000	20	10000	25	10000	ns
RETRASO CAS A RAS	TRCD	20	50	20	60	25	75	ns
TIEMPO RETRAS DIR COLUM A RAS	TRAD	15	35	15	40	20	50	ns
INIC DIREC RENGLON	TASR	0	-	0	-	0	-	ns
INIC DIREC COLUMNA	TASC	0	-	0	-	0	-	ns
DIREC COLUM REFER DE RAS	TAR	55	-	60	-	75	-	ns
COMAND ESC REFER DE RAS	TRWL	20	-	20	-	25	-	ns
COMAND ESC REFER DE CAS	TCWL	20	-	20	-	25	-	ns
INIC DE DAT	TDS	0	-	0	-	0	-	ns
INIC COMAND ESC	TWCS	0	-	0	-	0	-	ns
ACCS DE G	TGA	-	20	-	20	-	25	ns
COMAND READ REFER DE RAS	TRRH	0	-	0	-	0	-	ns

TABLA 4.7 TIEMPOS REQUERIDOS POR LAS MEMORIAS TIPO DRAM

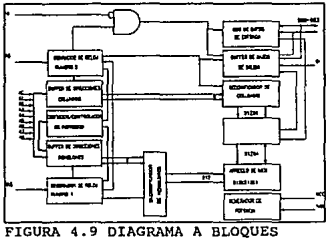


FIGURA 4.9 DIAGRAMA A BLOQUES DE LA MEMORIA PROPUESTA

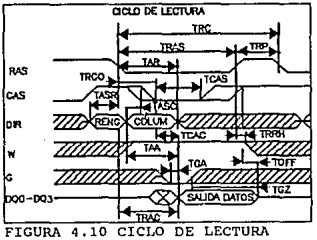


FIGURA 4.10 CICLO DE LECTURA

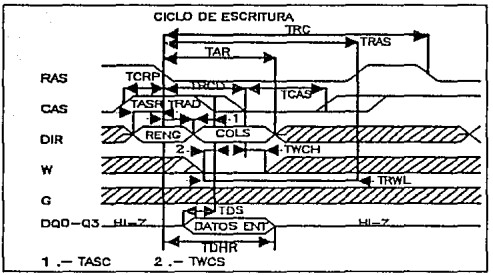


FIGURA 4.11 CICLO DE ESCRITURA

Teniendo como base este tipo de elementos podemos analizar sus diagramas de tiempos mostrados en las Figuras 4.10 y 4.11 Para ajustar la configuración del transputer con el fin de que responda en forma adecuada a su banco de memoria externa.Lo primero que se determino fue la configuración interna en base a la Tabla 4.7 y a las Figuras 4.7 y 4.8 con lo que se obtuvo MemAD5 como la mejor opción por lo mostrado en las Tablas 4.3 y 4.5 con respecto a los tiempos de las memorias propuestas.Con esto, se designo a la señal notmems3 como la señal CAS de las memorias y a notmems1 como señal RAS.

Ahora bien, como se señaló en el Capítulo pasado , el transputer tiene el bus de datos multiplexado con el bus de direcciones ; este último , tiene las direcciones columnas colocadas en la parte inferior del bus y las direcciones renglones colocadas en la parte superior, mostrando al mismo tiempo las columnas y los renglones, por lo que se requiere un retraso tiempo entre columnas y renglones para poder funcionar con las memorias dinámicas. En base a las Figuras 4.10 y 4.11 vemos que existen dos condiciones; la primera que las direcciones renglones deben de estar un cierto tiempo antes que las direcciones columnas y la segunda que estas dos ocupan un mismo bus de direccionamiento.

Retomando al los diagramas de tiempos del transputer podemos ver que notMems0 se activa al mismo tiempo que notMems1 (RAS para las memorias) cuando las direcciones "columnas" están presentes , con lo que notMems0 puede activar por flanco a un circuito LATCH que sirva para almacenar , y por lo tanto retrasar, a las direcciones columnas como lo muestra la Figura 4.12. Hay que recordar que el bus de direccionamiento de las memorias se encuentra multiplexado para renglones y columnas , con lo que el bus externo de direcciones del transputer también debe de estar en forma multiplexada.

De nueva cuenta, en los diagramas de tiempo del transputer, vemos que existe un retraso de tiempo entre las señales notMems2 y notMemS1 que nos es suficiente para multiplexar de forma eficiente los renglones y columnas de las memorias. Para que esto funcione se requiere de un multiplexor veloz que responda al retraso de tiempo TSOL2L (para la Figura 4.7) de forma que muestre las direcciones columnas antes de que el tiempo TCLZ (para la Figura 4.10) haya transcurrido, como el mostrado en la Figura 4.13. Revisando a los Diagramas 4.12 y 4.13 ,tenemos que señalar que en ambos casos las salidas de los mismos se encuentran abiertas, mostrando su contenido sin que esto afecte al desempeño del sistema.

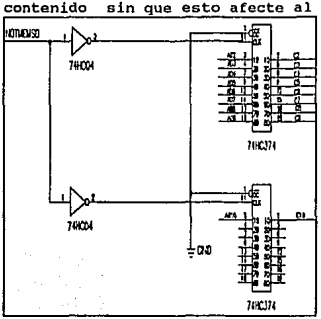


FIGURA 4.12 CIRCUITO DE LATCHEO

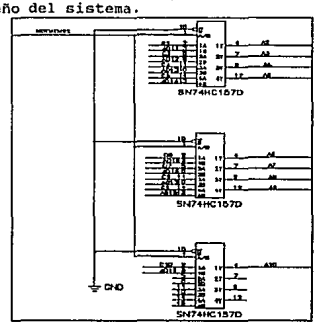
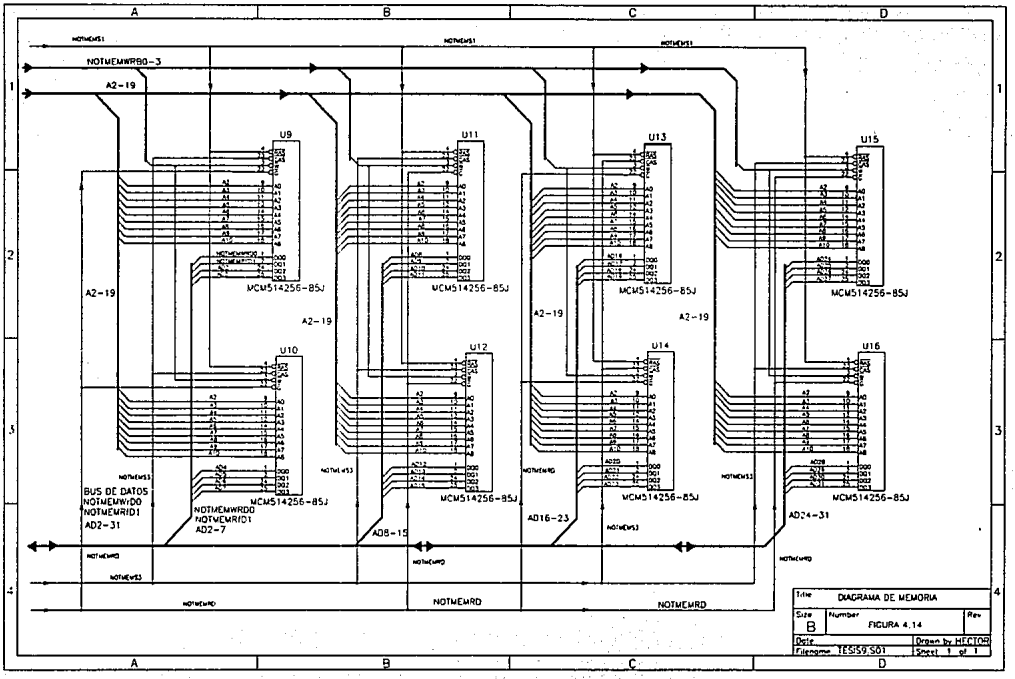


FIGURA 4.13 CIRCUITO DE MULTIPLEXAJE

Hasta el momento hemos visto el direccionamiento de las memorias y su selección de direcciones renglones y direcciones columnas , faltaría por señalar la selección de escritura y de lectura. La selección de lectura se lleva acabo por medio de la señal NotmemRd (Figura 4.7) que se habilita en el flanco de subida y es única para todas las memorias , dado que, el transputer leerá solamente palabras de 32 bits. La selección de escritura es mucho más flexible que su contraparte de lectura , pues, es manejada por medio de 4 señales (notMemWrB0-3) que se activarán al mismo tiempo por el flanco de subida como se ve en la Figura 4.8, pero se diferenciarán de NotmemRd porque solo se activarán en función del largo de la palabra que se quiera guardar , por ejemplo un BYTE , una palabra , una palabra larga, etc., por lo que la conexión de las señales esta en función del nivel de la división del bus de datos, que esta desde la parte más alta (bits más significativos) hasta la parte más baja (bits menos significativos) como lo muestra el Diagrama final 4.14.


NOTMEMWRB0-3
A2-19
U9
U10
U11
U12
U13
U14
U15
U16
MCM514256-85J
A2-19
BUS DE DATOS
NOTMEMWD0
NOTMEMRD1
AD2-31
NOTMEMWRD0
NOTMEMRD1
AD2-7
AD8-15
AD16-23
AD24-31
NOTMEMRD
Title
DIAGRAMA DE MEMORIA
Size
B
Number
FIGURA 4.14
Rev
Date
Drawn by HECTOR
Filename TESIS9.SCI
Sheet 1 of 1

4.6 LINKS DE COMUNICACIÓN

El concepto de link es fundamental para el transputer y el OCCAM. Un link es la implementación en hardware de un canal en OCCAM , cada link bidireccional posee un par de canales de OCCAM , uno en cada dirección. Un link provee una comunicación de datos seriales entre dos dispositivos de la familia de los transputers a velocidades entre 10 Mbps y 20 Mbps[3].

Un link entre dos transputers es implantado por la operación de la interfase de operación de un transputer a otra interfase de operación de otro transputer por dos señales unidireccionales. Cada señal envía datos e información de control. Eléctricamente las señales de los links son compatibles con TTL y permiten la comunicación a corta distancia (menos de 0.3 metros) . Los links son diseñados para comunicación local . Sin embargo, es posible su uso para grandes distancias teniendo encuentra otro tipo de dispositivos de comunicaciones.

Cada mensaje es transmitido como una secuencia de comunicaciones (paquetes) de un solo byte, requiriendo solo la presencia de un solo byte en el bus del transputer receptor . Como se vio en el capítulo anterior, cada byte es transmitido con un bit de inicio ,un bit de anuncio de paquete seguido por ocho bits de información y un bit de fin. Después de transmitir un byte de dato, el receptor espera hasta que el reconocimiento sea recibido Figura 4.15a. Este consiste de un bit de inicio seguido por un cero Figura 4.15b. El reconocimiento indica para ambos que el proceso fue capaz de recibir el byte y que el proceso receptor es capaz de recibir otro byte.

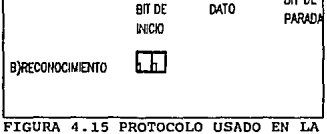


FIGURA 4.15 PROTOCOLO USADO EN LA TRANSFERENCIA DE INFORMACION

Los bytes de datos y de reconocimiento pueden ser multiplexados bajo cada señal de comunicación conjunta. En una implementación, los reconocimientos saldrán del receptor después de que los once bits del paquete hayan sido enviados.

Recordando que los links pueden ser conectados de forma muy simple para distancias cortas (menores a 0.3 metros). Para lo cual no es requerido ningún tipo de circuito en estas distancias dando como resultado el circuito mostrado en la Figura 4.16, sin embargo, para distancias mayores se presentarán los siguientes problemas:

1.- EFECTOS DE LINEAS DE TRANSMISION
2.- RUIDO
3.- ATENUACION DE LINEAS
4.- DISPERSION DE PULSOS
5.- PROPAGACION DE RETARDOS

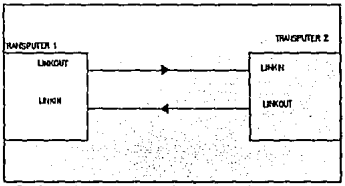


FIGURA 4.16 COMUNICACION SENCILLA ENTRE TRANSPUTERS

En este tipo de casos se opta por colocar circuitos drivers conforme al standard EIA422[4], y cuando se usa en el par de transmisiones se obtiene una máxima inmunidad al ruido. La señal será enviada en modo común diferencial obteniendo así una máxima eliminación de ruido. La Figura 4.17 muestra una implementación de un sistema EIA422 para la comunicación de dos links . Para nuestro caso, la utilización de drivers (Figura 4.18) será necesaria solo para la comunicación con la PC, puesto que, como se verá en el apéndice A el sistema que se utilice como máquina HOST tendrá como salida/entrada un funcionamiento diferencial. Con lo que podemos concluir que solo un link de los transputers será necesario para efectuar la comunicación con la máquina HOST.

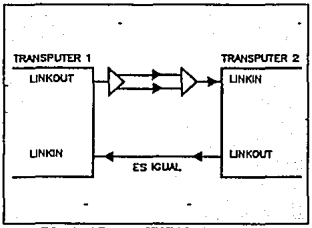


FIGURA 4.17 COMUNICACION A DISTANCIA

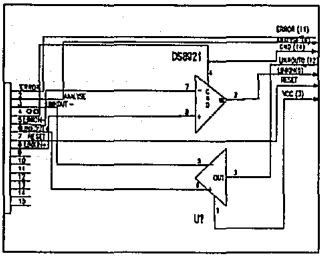


FIGURA 4.18 INTERFASE ENTRE EL HOST Y EL SISTEMA DISEÑADO

Es necesario mencionar que , los links por si mismos no presentarán problemas, sin embargo, se requiere una protección para sobrecargas y para balanceo de impedancias en la línea. En este sentido, se utilizaron 3 elementos, dos para la entrada y uno para la salida .Los elementos de la entrada están configurados de la forma como se muestra en la Figura 4.19, teniendo como fin evitar sobrevoltajes y descargas electrostáticas. El elemento de salida es una resistencia de 47 ohms (Figura 4.20) , que sirve para evitar sobrecorrientes debidas a desacoplo de impedancias.

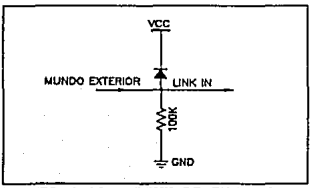


FIGURA 4.19 LINK DE ENTRADA

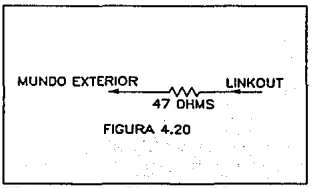


FIGURA 4.20 ELEMENTO DE PROTECCION PARA EL LINK DE SALIDA

El reloj de entrada se decidió colocar a una velocidad de 5MHz para lograr una velocidad en el transputer de 20MHz , logrando con esto, la posibilidad de probar una serie de aplicaciones en tiempo real sin tener limitantes de tiempo.

Las señales de control y comunicación con el mundo exterior se decidió colocarlas en base al estándar de un TRAM de tamaño dos[1]. Cabe señalar, que el sistema propuesto tiene un reset integrado que permite dar un master reset , el cual tiene como finalidad el activar analyse y reset al mismo tiempo logrando con esto el reset deseado, su configuración es la mostrada en la Figura 4.21. Así mismo existen una serie de capacitores de acoplamiento colocados entre la polarización de cada elemento con a finalidad de eliminar el ruido blanco que se pudiera presentar en el sistema.

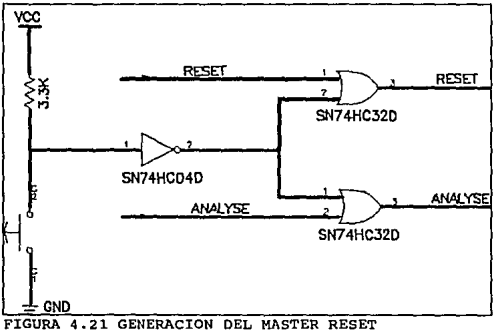


FIGURA 4.21 GENERACION DEL MASTER RESET

Lo señalado anteriormente abarca todo el diseño requerido para un módulo de procesamiento y queda configurado como lo muestran los Diagramas finales 4.24, 4.14 y 4.18. Cabe señalar que los siguientes módulos tendrán el mismo contenido con la variación de que sus cuatro links se presentarán en forma TTL y no en forma diferencial , como el Diagrama 4.18.

4.7 INTERCONECCION DEL SISTEMA

Como se señaló en la introducción de la tesis , el diseño buscado era la implementación e integración de módulos de procesamiento en base a transputers. Lo mostrado anteriormente nos da una visión del diseño de un módulo de procesamiento, lo que falta ahora es la forma de conexión entre ellos. Como se vio en el Capítulo dos, existen varias formas de construcción de una red de procesamiento.

1.- ESTATICAS
2.- SEMI ESTATICAS
3.- DINAMICAS

Para efectos de aplicación se determino utilizar una red de forma estática que originalmente trabaje con dos procesadores; por lo que se decidió utilizar una conexión tipo pipeline como la mostrada en la Figura 4.22. Ahora bien , el sistema de servicios como RESET, ANALYSE, NOTERROR, es necesario que sea leído por el procesador HOST que en este caso es el TEK (TRANSPUTER EDUCATION KIT), puesto que, el sistema que integran estos procesadores deben de ser esclavos del HOST. Lo cual nos lleva a pensar en que el sistema , requiere de un reset general que va estar dado por el TEK y se propaga por toda la red al interconectarlo físicamente entre todos los módulos de procesamiento. Lo mismo ocurrirá al conectar la señal NOTERROR y ANALYSE. Todo lo anterior es posible observarlo en el Diagrama 4.23. Las pruebas del sistemas fueron realizadas en base a los programas mostrados en el Apéndice B.

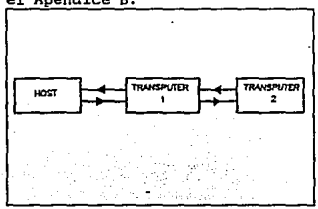


FIGURA 4.22 SISTEMA FINAL

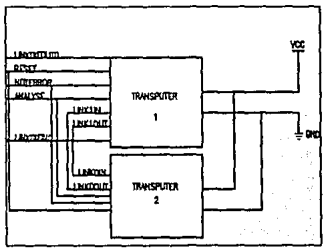
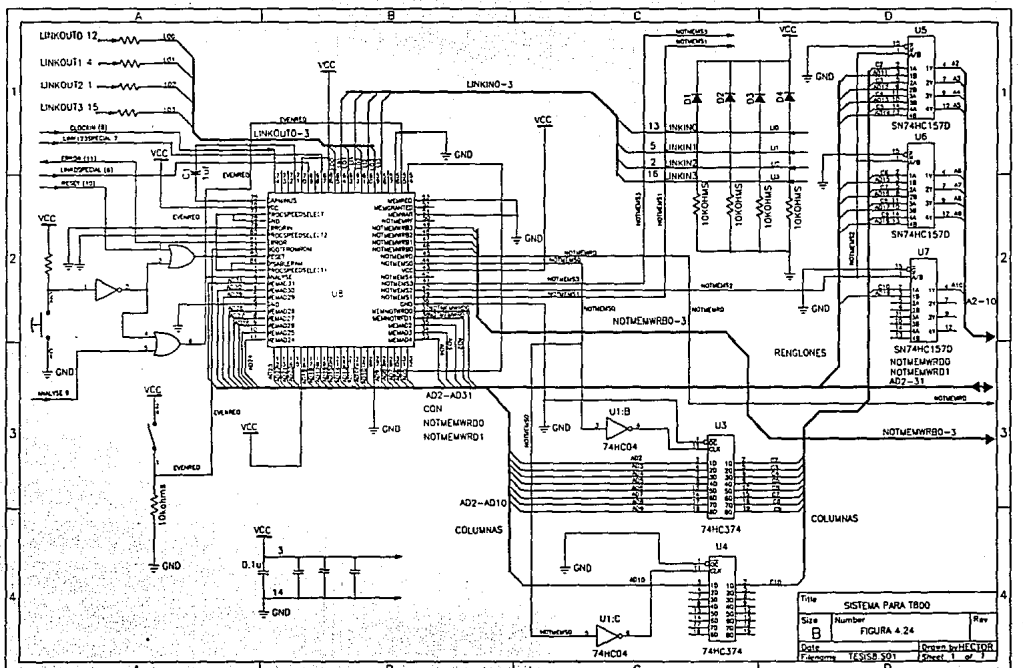


FIGURA 4.23 DIAGRAMA DE CONFIGURACION


LINKOUT0 12
LINKOUT1 4
LINKOUT2 1
LINKOUT3 15
LINKOUT0-3
LINKIN0-3
VCC
GND
U5
SN74HC157D
U6
SN74HC157D
U7
SN74HC157D
NOTMEMWRB0-3
RENGLONES
NOTMEMWRD0
NOTMEMWRD1
AD2-31
A2-10
AD2-AD31
CON
NOTMEMWRD0
NOTMEMWRD1
U1:B
74HC04
U3
74HC374
AD2-AD10
COLUMNAS
U4
74HC374
U1:C
74HC04
0.1u
10KOHMS
SISTEMA PARA TBDD
FIGURA 4.24
TESIS8.SCH
HECTOR

4.8 SUMARIO

En el presente Capítulo se han analizado una serie características del transputer buscando obtener un diseño eficiente que cumpla con las expectativas de procesamiento buscadas en la aplicación (véase Capítulo 5).

Dos fueron los campos más importantes en el desarrollo de este proyecto; por un lado el diseño de la interfase de memoria y por otro la comunicación con el mundo exterior. Para el primero se plantearon una serie de alternativas que se fundamentaban en el tipo de memoria a usar y la velocidad de procesamiento buscada, en este sentido la flexibilidad presentada por el transputer fue de gran apoyo para la elección de la memoria externa. Para el segundo campo se buscó darle protección al sistema contra diferentes tipos de descargas que en un momento dado pudieran aparecer entre el mundo exterior y el transputer, dañando a este último.

Así mismo, una de las ventajas presentadas por diseños como este es la capacidad de conexión entre dispositivos de su mismo tipo, por lo que el sistema puede incrementar poder de procesamiento de forma casi lineal solamente afectado por el tiempo de comunicación entre procesadores.

Uno de los puntos más relevantes en el desarrollo del sistema fue la tarjeta de conexión entre los módulos, pues, se buscó conformar un sistema que pudiera ser manejado por el sistema HOST sin necesidad de tener que usar otro dispositivo de comunicación más que un link y sus señales de subsistema (véase Apéndice A). Por último, podemos señalar que esta tarjeta de interconexión es solo una fase en el desarrollo de este proyecto, ya que se planea que sea reconfigurable por medio de software. Este es uno de los proyectos futuros a desarrollarse a partir de la conclusión de esta tesis.

4.9 REFERENCIAS

1.- GRAHAM, IAN AND KING, TIM, THE TRANSPUTER HANDBOOK, PRENTICE HALL, 1989, SEGUNDA EDICION

2.- INMOS, THE TRANSPUTER APLICATIONS NOTEBOOK, INMOS, 1989, SEGUNDA EDICION

3.- INMOS, THE TRANSPUTER DATABOOK, INMOS, 1989, SEGUNDA EDICION

4.- MOTOROLA, LINEAR AND INTERACE INTEGRATED CIRCUITS, MOTOROLA, 1989, PRIMER EDICION

5.- MOTOROLA, MEMORY DATAM, MOTOROLA, 1989, PRIMER EDICION

CAPITULO V

APLICACION - PROCESAMIENTO PARALELO DE SEÑALES DOPPLER DE ULTRASONIDO

5.1 INTRODUCCION

Técnicas de ultrasonido Doppler son utilizados para la detección de problemas cardiovasculares, esta se lleva acabo mediante el uso de técnicas de estimación espectral en la medición de la velocidad del flujo sanguíneo y en el monitoreo turbulencias. Un instrumento útil para el diagnóstico y monitoreo de padecimientos cardiovasculares es el detector pulsatil de flujo sanguíneo Doppler. Este instrumento determina la velocidad del torrente sanguíneo y detecta además turbulencias en su flujo. Un incremento en el rango de frecuencias Doppler, como resultado de algún tipo de turbulencia en el flujo sanguíneo, es usado para detectar lesiones escleróticas. Debido a que la velocidad de la sangre dentro de las arterias es periódica, La señal Doppler es ciclo estacionaria. Por tal razón, el espectro Doppler varía en la frecuencia media y en la forma a través del ciclo cardíaco. Esto hace necesario el tener que realizar el análisis espectral de la señal en intervalos de tiempo pequeños (5-10 ms), en los que puede considerarse que la señal es estacionaria.

Los instrumentos hasta ahora disponibles hacen uso del algoritmo FFT para calcular el espectro de la señal Doppler generada por el flujo sanguíneo, extrayendo de este modo la información diagnóstica cuantitativa. Moderados y severos padecimientos pueden ser detectados de esta manera; sin embargo, con esta técnica es difícil distinguir problemas de estenosis en estado inicial, debido a limitantes de la FFT asociadas a resolución y segmentación[4].

Recientes trabajos de investigación han conducido a la conclusión de que los métodos paramétricos de estimación espectral ofrecen una importante mejora en la resolución de frecuencia[8,9].

Estudios previos realizados para investigar el desempeño de un número de estimadores espectrales en señales Doppler, han identificado el método paramétrico AutoRegresivo denominado Método Modificado de Covariancia como costo-efectivo. El costo esta asociado a su complejidad computacional, ya que este método está formado de un número de procesos que incluyen cálculos matriciales, solución de ecuaciones lineales y cálculo de la densidad de potencia espectral para obtener las componentes de frecuencia de la señal. La efectividad está asociada a su resolución ya que es susceptible de implantarse en tiempo real con técnicas de procesamiento paralelo.

El objetivo de este trabajo es la implantación con transputers de la versión paralelizada de los algoritmos paramétricos, como el Método Covariancia Modificada, y no paramétricos, como los basados en la FFT, para ejecutarlos en tiempo real y demostrar la potencia del procesamiento paralelo así como el eficiencia del Método paramétrico sobre su contraparte no paramétrica.

Ambas técnicas son usadas para calcular, en tiempo real, el contenido de frecuencias de la señal Doppler generada por un instrumento Doppler pulsado.

5.2 ULTRASONIDO DOPPLER

El presente capítulo muestra los fundamentos teóricos de la aplicación , así como los procedimientos seguidos para el diseño de los programas mostrados en el apéndice C.

La señal Doppler proviene de una señal ultrasónica que consiste de disturbios mecánicos en el medio (gas,líquido o sólido) la cual pasa a través del éste a una velocidad constante. Las ondas del sonido consisten de disturbios de las moléculas en el aire, las vibraciones pasan de molécula a molécula desde la fuente hasta el destino. Note que las moléculas por si mismas no se mueven entre la fuente y el destino, requieren de un disturbio. La relación en la cual las partículas en el medio vibran es la frecuencia o velocidad del sonido y es medida en Hertz. Cuando la velocidad se incrementa mas haya de una frecuencia cercana a los 20 KHz el nombre que reciben este tipo de disturbios es el de ultrasónicos[1].

En el ultrasonido médico, el disturbio es caracterizado por un cambio en la presión local o el movimiento en la distancia de las partículas del medio desde su posición inicial, originando un movimiento electromecánico en algún transductor cualquiera. El transductor (operando como transmisor) cambia señales eléctricas por movimientos mecánicos. Las vibraciones mecánicas pueden ser cambiadas por señales eléctricas por el mismo transductor trabajando en forma inversa (como receptor). El disturbio se propaga a través de un medio a una cierta velocidad la cual depende de la compresibilidad y densidad del medio.

Dentro del desarrollo de la instrumentación médica con técnicas de ultrasonido encontramos dos áreas de investigación;la relacionada con instrumentos de onda continua y la que trabaja en base a ecos de pulsos[1]. En nuestro trabajo haremos referencia generalmente al instrumento de onda continua, este funciona en base al diagrama mostrado en la Figura 5.1.

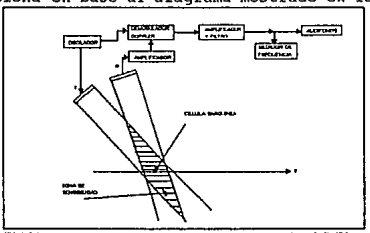


FIGURA 5.1 DIAGRAMA DE LA MEDICION DEL EFECTO DOPPLER POR METODO CONTINUO

Básicamente la forma de operación es la siguiente: El transductor transmisor es acoplado a un oscilador (si es necesario a un modulador para que se alcance una distancia más grande)de onda continua. El reflector de ultrasonido está acoplado a la misma punta pero el circuito receptor es diseñado y acoplado de forma diferente al sistema que se maneja en el transmisor . La señal de salida eléctrica para el transductor receptor es amplificada y desmodulada; posteriormente es pasada por un filtro y un amplificador, para así tener una señal en la cual se pueda distinguir con claridad el efecto doppler o señal doppler. Cabe señalar que la señal doppler tiene un rango de frecuencias correspondiente al rango de velocidades del flujo sanguíneo y al resultado de cierto tipo de ruido. Este último no será analizado en este trabajo pues no corresponde a las expectativas de éste.

La velocidad del flujo sanguíneo, en los vasos varía con el tiempo según las condiciones físicas del cuerpo humano presentando dos tipos, la velocidad en las arterias y la velocidad en las venas siendo la principal diferencia la forma de bombeo[1]. La velocidad en las arterias tiene una regulación de pulsación correspondiente a la del corazón y la velocidad en las venas comúnmente varía con la respiración, por lo que requerimos medir el flujo sanguíneo que se presenta en las arterias.

La forma de medición con respecto a un plano de trabajo y siguiendo la filosofía del método de onda continua es mostrada en la Figura 5.2.a mientras que en las Figuras 5.2.b y 5.2.c se muestra la forma de onda Doppler para una vena y una arteria respectivamente.

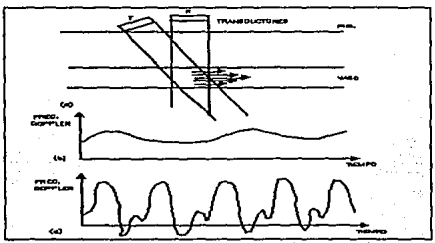


FIGURA 5.2 FUNCIONAMIENTO DE UN APARATO BASADO EN EL METODO CONTINUO

Por que el ultrasonido debe de ser transmitido continuamente y el mismo transductor no debe de ser usado para la recepción, existen dos transductores y la sensibilidad del instrumento en el flujo sanguíneo debe de estar en la región de sobrecarga de los dos rayos. El ángulo entre el rayo receptor y el rayo transmisor es usualmente pequeño, resultando en la medida usual de la sensibilidad del rayo[1].

La mayor limitación que presenta el instrumento de doppler continuo[1], es su sensibilidad al flujo en la región de traslape de los dos rayos ultrasónicos, si esto no sucede, este se verá limitado por la atenuación. Esto significa que no hay separación de señales de dos o más vasos en el rayo. Por lo que , al manejar un solo rayo se evita el problema tener que manejar una región muy particular dependiendo del ángulo manejado por los transductores, a este nuevo sistema se le conoce como Doppler pulsado.

El diagrama de bloques de este instrumento es mostrado en la Figura 5.3 . Revisándolo, tenemos que, el transductor que es usado como transmisor es también usado como receptor . El transductor es usado durante la transmisión por una pequeña "explosión " ocurrida en la señal de transmisión; esto significa; que la señal transmitida será un impulso el cual excitará al transductor para que emita una señal ultrasónica.

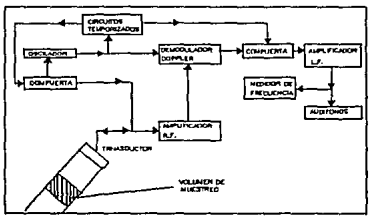


FIGURA 5.3 DIAGRAMA A BLOQUES DE UN INSTRUMENTO DE DOPPLER PULSADO

Cabe señalar que el impulso es en el dominio de la frecuencia una recta paralela al eje de las frecuencias, lo cual quiere decir que con un impulso, teóricamente se barren todas las frecuencias. Al barrer todas las frecuencias,

tenemos que tocar la frecuencia central del transductor y con esto ocasionar su excitación.

En el momento que el transductor termine de enviar el pulso éste será switcheado para actuar como receptor y quedará en estado de espera hasta que se tenga que enviar otro pulso. La señal, al ser capturada por el receptor, será amplificada por un dispositivo logarítmico para tener un mejor control de la ganancia, después, será filtrada y de ahí pasará a la plataforma paralela para ser procesada. El instrumento es sensible al flujo solo con pequeños volúmenes de paquetes, mientras es localizado cualquiera de los paquetes el tiempo que el transductor permanecerá como receptor será mayor . Es importante señalar que, el circuito pulsador solo tiene diferencias importantes con respecto al circuito de onda continua en el desarrollo del transmisor. El operador puede examinar diferentes profundidades del flujo alterando el tiempo de retraso de la relación transmisión muestreada, además de poder alterar el espesor de los paquetes cambiando la duración de la transmisión y/o la duración de muestreo del receptor.

Lo anterior nos permite darnos una idea general sobre los métodos e instrumentos utilizados para obtener la señal Doppler sobre el flujo sanguíneo. Estableciendo así las bases necesarias para el entendimiento de la aplicación que es el procesamiento de una señal Doppler por métodos

paramétricos y no paramétricos buscando la obtención de su frecuencia central así como su ancho de banda. Puntos básicos para el análisis del flujo sanguíneo que como ya se ha dicho no pertenecen a los objetivos de esta tesis.

5.3 IMPLANTACION DE UN ESTIMADOR BASADO EN EL METODO DE LA FFT

El principal análisis que se busca realizar durante el trabajo es en base a una señal discreta que tendrá características muy particulares en el Doppler. Como se menciono, utilizaremos como primer método a la transformada discreta de fourier (DFT) . Pero como este método es poco eficiente tendremos que utilizar un método optimizado llamado transformada rápida de fourier (FFT).

Para empezar definiremos a la DFT como un set de N muestras [x(k)] de la transformada de fourier X(w) para una secuencia de duración finita [x(n)] de largo L menor o igual que N^2. El muestreo de X(w) ocurre en N espacios iguales de frecuencias $w_k=2(pi)k/N$, k=0,1,2,...,N-1. Nosotros demostraremos que las N muestras [X(k)] únicamente representa la frecuencia de [x(n)] en el

dominio de la frecuencia . Debemos tomar como base que la DFT y la IDFT están declaradas como

$$DFT: X(k)=\sum_{n=0}^{N-1} x(n)W_N^{kn} \quad k=0,1,..,N-1 \tag{5.2.1}$$

$$IDFT: \quad x(n)=\frac{1}{N}\sum_{k=0}^{N-1} X(k)W_N^{-kn} \quad n=0,1,...,N-1 \tag{5.2.2}$$

donde W_N es conocido como

$$W_N=\exp^{\frac{-2\pi}{N}} \tag{5.2.3}$$

y constituye una función compleja básica , o twidle factor de la DFT[5] . El twidle factor es periódico y define sus puntos en base a un círculo unitario en el plano complejo.

Dado que la transformada DFT e IDFT envuelven el básicamente el mismo tipo de cálculos , cualquier discusión de la eficiencia en el cálculo de la transformada DFT también es válido para la IDFT[7].

Para cada valor de k , el cálculo directo de X(k) envuelve N multiplicaciones complejas (4N multplicaciones reales) y N-1 sumas complejas (4N-2 sumas reales). Consecuentemente, el cálculo de todos los valores de la DFT requieren N^2 multiplicaciones complejas y (N^2-N) sumas complejas.

Los cálculos directos de la DFT son francamente ineficientes porque no explotan la simetría y las propiedades de periodicidad de la fase del factor W_N.

Cuando N es un número de grande, puede ser descompuesto en productos de números primos de la forma

$$N=r_1r_2...r_v$$

entonces la descomposición puede ser repetida (v-1) veces . De este procedimiento resultan pequeñas DFT's , las cuales provocan una mayor eficiencia en el cálculo del algoritmo.

De particular importancia es el caso en el que $r_1=r_2=...=r_v=r$, esto es, $N=r^v$. En el caso en que el tamaño de las transformadas es de r, el cálculo del enésimo punto tiene un patrón regular . El número r es llamado radio del algoritmo de la FFT. Para nuestros objetivos , nosotros describiremos a la transformada de radio 2.

Debemos de considerar para el cálculo de la DFT a $N=2^v$ puntos ; con lo cual podemos hacer una división de N/2 puntos y dos secuencias de datos ; de tal forma que, $f_1(n)$ y $f_2(n)$ (secuencias propuestas) correspondan a las muestras pares y a las muestras nones de x(n), respectivamente

$$f_1(n)=x(2n)$$
$$f_2(n)=x(2n+1) \qquad n=0,1,...,N/2-1$$

f_1 y f_2 son obtenidas por decimación de x(n) en un factor de 2 , de aquí el resultado del algoritmo FFT es llamado algoritmo en decimación en tiempo[7].

La decimación de la secuencia de datos puede ser repetida una y otra vez hasta que el resultado de las secuencias estén reducidas a su mínima expresión como lo marca r . Es decir para $N=2^v$ esta decimación debe de ser referida a $v=\log_2 N$ veces. Entonces el número total de multiplicaciones complejas requeridas son reducidas a $(N/2)\log_2 N$. El número de sumas complejas es $N\log_2 N$[4].

Para ilustrar este propósito la Figura 5.4 describe el cálculo de una transformada DFT de 8 puntos. Nosotros vemos que el cálculo es realizado en tres estados , partiendo de cálculos de cuatro DFT de dos puntos , después dos DFT de cuatro puntos, y finalmente una DFT de 8 puntos. La combinación de pequeñas DFT para formar una gran DFT es ilustrada en la Figura 5.5 para N=8.

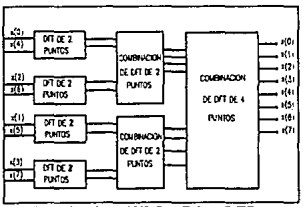


FIGURA 5.4 CALCULO DE LA DFT EN 8 PUNTOS

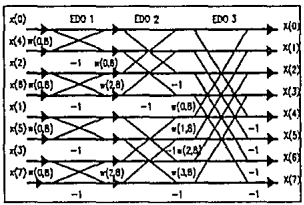


FIGURA 5.5 COMBINACION DE DFTS PARA FORMAR LA DFT FINAL

Obsérvese que el cálculo básico realizado en cualquier estado como el ilustrado en la Figura 5.5 es el tomar dos números complejos (a,b) , multiplicar b por W^{r}_{N}, y después sumar el producto a "a" para formar dos nuevos números complejos (A,B) . Este cálculo básico , que es mostrado en la Figura 5.6 es llamado mariposa porque el diagrama de flujo representa una mariposa.

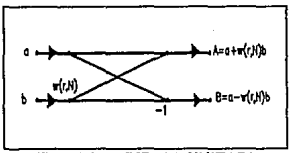


FIGURA 5.6 MARIPOSA UNITARIA

En general, cada mariposa envuelve una multiplicación y dos sumas complejas. Para $N = 2^{v}$, hay N/2 mariposas por estado del proceso de cálculo y $\log_2 N$ estados . De ahí, el número total de multiplicaciones es $(N/2)\log_2 N$ y las sumas complejas son $N\log_2 N$, como se indico previamente[7].

Cada operación de mariposa es realizada en un par de números complejos (a,b) produciendo (A,B), no hay necesidad de guardar el par de entrada (a,b). Entonces, nosotros podemos guardar el resultado (A,B) en las mismas localidades de (a,b). Consecuentemente, nosotros requerimos un corrimiento del almacenamiento , llamado, 2N registros de almacén, que sirven para guardar los resultados de los cálculos de cada estado.

Otra consideración es en el orden de la entrada de la secuencia de datos de entrada después de que es decimada (v-1) veces.

Existe un "desordenamiento" de la secuencia de datos de entrada que tiene un orden bien definido como puede ser visto al observar la Figura 5.7, que ilustra la decimación de una secuencia de ocho puntos. Para expresar el índice n, en la secuencia x(n), en forma binaria nosotros vemos que la secuencia de datos decimada es fácilmente obtenida por la lectura de la representación binaria de el índice n en orden reverso. Esto es , el dato en su posición original x(3)= x(011) es puesto en la posición m=110 o m=6 en el arreglo decimado. Nosotros vemos que el arreglo x(n) después de la decimación es guardado en orden "bit-reversed".

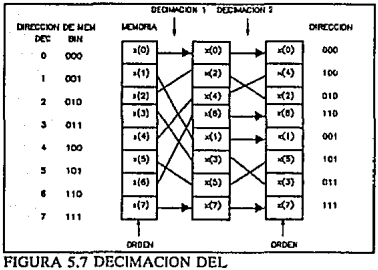


FIGURA 5.7 DECIMACION DEL "BIT-REVERSED"

5.3.1 DESCRIPCION DEL ALGORITMO PROPUESTO

Con la secuencia de datos de entrada guardada en bit-reversed y los cálculos de las mariposas realizados en el momento , el resultado de la transformada es obtenido en forma natural. De otra forma, debemos indicar que en el arranque del algoritmo de FFT la entrada es puesta en forma natural y el resultado de la transformada será en bit-reversed.

Ahora bien, para la aplicación requerida nosotros necesitamos conocer la densidad de potencia espectral, para lo cual, tenemos que obtener el módulo cuadrado de todos los puntos generados por la transformada.

Para el caso de la investigación, nosotros realizamos el algoritmo antes señalado con el nombre de FFT de radix dos en base al lenguaje OCCAM, para efectos de prueba y futura comparación con un método paramétrico (covariancia modificada).

El algoritmo esta dividido en tres bloques funcionales que se distribuyen a través de la plataforma paralela siendo dos para la captura y almacenamiento de datos y el tercero tendrá como función el cálculo de la FFT y su Densidad de Potencia Espectral. El proceso inicia con el cálculo de los

coseno y senos de los factores de corrimiento "twidle factors", almacenandolos en una serie de arreglos que después serán llamados en el momento del cálculo de la transformada; el siguiente paso es el ordenar a los datos de forma bit-reversed por medio del cálculo de la ubicación de su valor espejo en el campo binario volviéndolos a guardar en el mismo arreglo, con los resultados obtenidos de los pasos anteriores se procesa a la transformada que solo será la sumatoria de pares de datos con sus respectivos factores de corrimiento los cuales serán elegidos por el número de mariposas en que se encuentre el cálculo. Después de aplicar la FFT a los datos iniciales se obtiene un arreglo de datos de tipo complejo del cual se tomará solo la mitad, dado que la FFT arroja sus resultados de forma normal y de forma a espejo. Al tener este último arreglo se calculará su módulo cuadrado de cada uno de los datos complejos obteniendo así su Densidad de Potencia Espectral.

La otras dos partes del programa propuesto, como se menciono anteriormente, son el manejo de datos con respecto del sistema. La primer parte es un proceso que se encuentra en el procesador maestro y tendrá como función leer el archivo que el usuario le determine y transmitirlo por medio de un canal físico hacia el procesador esclavo que tendrá la función de

calcular la FFT. La segunda parte recibirá los resultados obtenidos del cálculo realizado por el procesador esclavo y los almacenará en un archivo que indique el usuario.

De la misma forma en que se realizó el programa para un solo procesador se realizó otro programa que repartiera el arreglo de datos en dos procesadores de forma tal que se procesan dos ventanas de datos consecutivas, esto con el objeto de poder demostrar la eficiencia del procesamiento paralelo sobre el procesamiento secuencial.

El programa en su versión paralela solo contiene un proceso más, el cual manejará la distribución de los datos para que los procesadores calculen la transformada de Fourier de sus respectivos arreglos. Al igual que el programa anterior éste puede ser leído en el apéndice C.

Cabe hacer mención que el programa en la versión paralela deberá procesar el doble de información que el requerido por el programa en su versión serie en el mismo lapso de tiempo más un período necesario para la comunicación entre procesadores. Sobre esto, se tiene que tomar una serie de tiempos que nos expresen el desempeño del algoritmo de forma secuencial para un arreglo de N datos cargado en un solo procesador y para un algoritmo de forma secuencial para un arreglo de 2N datos copiado en dos procesadores.

5.3.2 DESEMPEÑO

Como se ha dicho, una de las razones mas importantes para implementar el algoritmo de la FFT en serie y en paralelo es el demostrar el incremento en la eficiencia a partir de su paralelización, para lo cual se tomaron una serie de tiempos que servirán para mostrar la eficiencia del sistema. Dichos tiempos fueron tomados en base a un esquema de alta prioridad (véase capítulo 3) por lo que cada tic equivale a 1 micro segundo y su conversión a segundos es directa. Estos resultados fueron utilizados para calcular la eficiencia en base a lo visto en el capítulo 2 y son mostrados en las Figuras 5.8, 5.9 y 5.10.

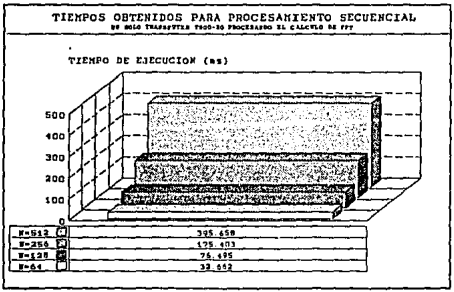


FIGURA 5.8 TIEMPOS OBTENIDOS EN BASE AL ALGORITMO FFT EN UN SOLO TRASPUTER

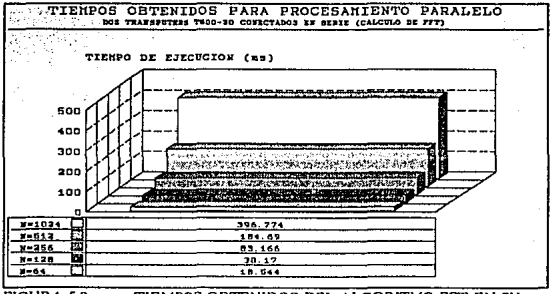


FIGURA 5.9 TIEMPOS OBTENIDOS DEL ALGORITMO FFT EN SU VERSION PARALELA EN BASE A DOS TRANSPUTERS DE FORMA PIPELINE

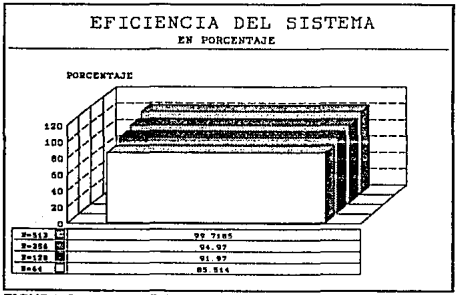


FIGURA 5.10 EFICIENCIA DEL SISTEMA EN BASE AL ALGORITMO NO PARAMETRICO

En base a estos resultados se observa que la eficiencia se incrementa de forma casi lineal al ir creciendo el número de datos que se utilizan para el cálculo de la PSD. Es importante aclarar que en la gráfica 5.10 se realizaron los cálculos de la eficiencia sobre una ventana de N datos obtenida del algoritmo secuencial contra una ventana 2N datos obtenida del algoritmo paralelo dado que en este algoritmo cada procesador trabaja con N datos al igual que en el algoritmo secuencial. En base a lo anterior podemos anticipar que el desempeño de este sistema puede mejorar si un número adicional de procesadores son utilizados.

5.4 IMPLANTACION DE UN ESTIMADOR ESPECTRAL BASADO EN EL METODO DE COVARIANCIA MODIFICADA

En esta parte nosotros explicaremos el aprovechamiento alternativo para describir un proceso aleatorio basado en una modelación paramétrica. Los parámetros del modelo son obtenidos desde la función de autocorrelación (ACF siglas en inglés) por medio de la estimación paramétrica en base a un set limitado de datos. El ventaneo de datos o valores ACF hace que se asuma a los valores no observados o los valores fuera de la ventana de ACF como cero, lo cual es una predicción no real. Pero comúnmente nosotros tenemos mejor conocimiento de los procesos desde que las muestras de datos son tomadas , o de otra forma, nosotros debemos ser capases de hacer una mas razonable extrapolación que el asumir los datos fuera de la ventana con un valor de cero. El uso de esa información nos permite la selección de un modelo más exacto para el proceso que el generado por los datos muestreados , o al menos una muy buena aproximación. Con lo que , es usualmente posible obtener una mejor estimación del espectro basados desde las propias observaciones del modelo.

La estimación espectral realiza su función en tres pasos . El primero es seleccionar el modelo . El segundo, es el estimar los parámetros del modelo usando la capacidad de los datos muestreados. El tercero, es el obtener el estimador espectral por substitución de los parámetros del modelo dentro de la teoría implicada por la densidad de potencia espectral (PSD siglas en inglés) dentro del modelo.

Una motivación por aprovechar la modelación en la estimación espectral es la aparente alta resolución de esta técnica sobre la obtenida por las técnicas clásicas discutidas anteriormente.

La clasificación de métodos dentro de paramétricos y no paramétricos es relativamente sencilla. La filosofía que será adoptada, es en base a una

técnica paramétrica que asume el conocimiento acerca de la PSD sobre otros que se validan en el teorema de Wiener-Khinchin[5]. Con esta definición la transformada de Fourier entra en la clasificación de los estimadores espectrales no paramétricos, mientras que, los autorregresivos (siglas en inglés AR), los de movimiento promedio (siglas en inglés MA) y los autorregresivos de movimiento promedio (siglas en inglés ARMA) son paramétricos[5]. Como lo muestra el diagrama 5.11[2]

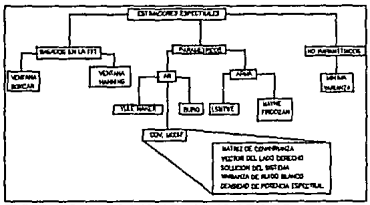


FIGURA 5.11 CLASIFICACION DE LOS ESTIMADORES ESPECTRALES

La modelación paramétrica para estimación espectral consiste en escoger un modelo apropiado , estimar los parámetros del modelo, y entonces substituir esos valores estimados dentro de las expresiones teóricas de la PSD. Los modelos a ser discutidos son series de tiempo o modelos de funciones de transferencia rotacional. Ellos son los modelos autorrgresivos (AR), los modelos de movimiento promedio (MA), y los modelos de movimiento promedio autoregresivos (ARMA). Estos modelos son comúnmente usados para distintos tipos de señales , como por ejemplo, el modelo AR es apropiado para señales con picos muy pronunciados y no para valles muy profundos , por el contrario , el modelo MA es común utilizarlo para señales con valles muy profundos , mientras que el modelo ARMA es el más general para representar ambos extremos.

Ahora bien, si todos los coeficientes b(k) excepto b(0)=1 son cero en el modelo ARMA, entonces

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

y el proceso es estrictamente un proceso AR de orden p . El proceso es llamado de autorregresión en que la secuencia x(n) es una regresión lineal de si misma

con una entrada u(n) representando el error .

Con este modelo, el valor presente del proceso es expresado como una suma de valores pasados mas un termino de ruido .la PSD es

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

Este modelo es llamado modelo de todos los polos y es mostrado en la Figura 5.12 es reconocido como un proceso AR(p)[3]

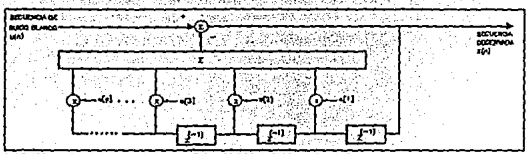


FIGURA 5.12 DIAGRAMA DE FLUJO DEL MODELO AR

Como apoyo podemos tomar el teorema de WOLD5[4],el cual dice que si la PSD es puramente continua,cualquier proceso ARMA o AR puede ser representado por un modelo MA de orden infinito.Además,el teorema Komolgorov señala de forma similar que para un proceso ARMA o MA puede ser representado por un proceso infinito AR[3].

Al ilustrar esos teoremas,nosotros podemos mostrar como el proceso ARMA(1,1) puede ser modelado por un AR(infinito) o por un MA(infinito).Tomando a un modelo ARMA(1,1) el proceso será

$$H(z)=\frac{1+b(1)z^{-1}}{1+a(1)z^{-1}}$$

Al representar el proceso ARMA como un proceso AR de orden infinito , tenemos que

$$H(z)=\frac{1}{1+c(1)z^{-1}+c(2)z^{-2}+\ldots}$$

Ahora bien, al estimar la PSD usando un modelo AR nosotros calculamos los parámetros del modelo [5]. La teoría de PSD es dada por

$$P_{AR}(f)=\frac{\sigma^2}{|1+a(1)\exp(-2j\pi f)+\ldots+a(p)\exp(-2j\pi f)|^2}$$

La estimación de la PSD es formada por el reemplazo de los parámetros AR por sus correspondientes valores estimados

$$\hat{P}_{AR}(f)=\frac{\sigma^2}{|1+\hat{a}(1)\exp(2j\pi f)+\ldots+\hat{a}(1)\exp(2j\pi f)|^2}$$

5.4.1 DESCRIPCION DEL ALGORITMO

Se eligió el método de Covariancia Modificada , dado que será quien nos presente una mayor aproximación a la señal y esto debido a sus cálculos hacia atrás y hacia adelante del error promedio de la señal. Además hay que recordar que bajo ciertas circunstancias un modelo ARMA puede ser representado por un modelo AR que es el caso que nos atañe.

Ahora bien, el método de covariancia es un estimador de máxima probabilidad[5] . Del cual para evaluar la predicción de potencia de error se toman los datos de la siguiente forma

$$\hat{\rho}=\frac{1}{N-p}\sum_{n=p}^{N-1}|x(n)+\sum_{k=1}^{p}a(k)x(n-k)|^2$$

Cabe señalar que la única diferencia entre el método de covariancia y el método de autocorrelación es el rango de sumatoria en la estimación de la potencia del error predictivo. En el método de covariancia todos los puntos necesarios para calcular ρ deben de ser observados . No es necesario declarar los datos con un valor de cero.

La minimización de la ecuación anterior puede ser efectuada aplicando un gradiente complejo sobre los parámetros estimados de AR como solución de las siguientes ecuaciones[3]

$$\begin{vmatrix} c_{xx}(1,1) & c_{xx}(1,2) & \dots & c_{xx}(1,p) \\ c_{xx}(2,1) & c_{xx}(2,2) & \dots & c_{xx}(2,p) \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ c_{xx}(p,1) & c_{xx}(p,2) & \dots & c_{xx}(p,p) \end{vmatrix} \begin{vmatrix} \hat{a}(1) \\ \hat{a}(2) \\ \cdot \\ \cdot \\ \hat{a}(p) \end{vmatrix} = - \begin{vmatrix} c_{xx}(1,0) \\ c_{xx}(2,0) \\ \cdot \\ \cdot \\ c_{xx}(p,0) \end{vmatrix} \qquad (5.3.1.1)$$

por lo que

$$c_{xx}(j,k)=\frac{1}{N-p}\sum_{i=p}^{N-1} x^*(n-j)x(n-k)$$

siendo el ruido blanco la variancia y estimada como

$$\hat{\sigma}^2=\hat{p}_{min}=c_{xx}(0,0)+\sum_{k=1}^{p}\hat{a}(k)c_{xx}(0,k)$$

La matriz 5.3.1.1 es hermitiana y semidefinitiva positiva (simétrica) . Sin embargo, cualquier observación de ruido puede causar que la matriz sea no singular . El sistema de ecuaciones puede ser resuelto utilizando la descomposición de Cholesky[3].

El uso de este método presenta varias ventajas sobre otros como el autorregresivo , dado que, covariancia en $c_{xx}(j,k)$ usa la suma de solo N-p productos al estimar la ACF , mientras que, los métodos autorregresivos utilizan los N datos para producir N productos , haciendo al algoritmo menos eficiente[3].

Por otro lado existe un método que en principio tiene el mismo concepto que el método de covariancia.El método al que nos referimos es el

de covariancia modificada. Este método consiste en optimizar el cálculo de AR por medio de un predictor hacia atrás y hacia adelante , el primero es

$$\hat{x}(n) = -\sum_{k=1}^{p} a(k)x(n-k)$$

mientras que el predictor hacia atrás es

$$\hat{x}(n) = -\sum_{k=1}^{p} a^*(k)x(n+k)$$

donde los a(k) son los parámetros del filtro AR. En cualquiera de los dos casos la predicción del error es por medio de la varianza del ruido blanco, sigma cuadrada. El método de covariancia modificada estima los parámetros AR por minimización del promedio del estimador hacia adelante y hacia atrás.

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

donde

$$\hat{\rho}^f = \frac{1}{N-p}\sum_{n=p}^{N-1} |x(n) + \sum_{k=1}^{p} a(k)x(n-k)|^2$$

$$\hat{\rho}^b = \frac{1}{N-p}\sum_{n=0}^{N-1-p} |x(n) + \sum_{k=1}^{p} a^*(k)x(n+k)|^2$$

En el método de covariancia las sumatorias son solo sobre los errores de predicción que envuelven los datos muestreados. Note que un camino

alternativo para visualizar este observador es el reconocer que ro cuadrada de b es el estimador del error de potencia de predicción obtenido por mover sus cálculos alrededor del archivo de parámetros en base a la conjugación de los mismos y a la aplicación de una predicción hacia adelante de este archivo[5].

Si tomamos todos los parámetros desde l = 1,2,...,p tenemos que

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

puede ser escrita en forma de matriz

$$\begin{vmatrix} c_{xx}(1,1) & c_{xx}(1,2) & \dots & cxx(1,p) \\ c_{xx}(2,1) & c_{xx}(2,2) & \dots & c_{xx}(2,p) \\ . & . & . & . \\ . & . & . & . \\ c_{xx}(p,1) & c_{xx}(p,2) & \dots & c_{xx}(p,p) \end{vmatrix} \begin{vmatrix} \hat{a}(1) \\ \hat{a}(2) \\ . \\ . \\ \hat{a}(p) \end{vmatrix} = - \begin{vmatrix} c_{xx}(1,0) \\ c_{xx}(2,0) \\ . \\ . \\ c_{xx}(p,0) \end{vmatrix}$$

y la estimación de la variancia de ruido blanco o sigma cuadrada es

$$\hat{\sigma}^2=\hat{p}\min=\frac{1}{2(N-p)}\left(\sum_{n=p}^{N-1}(x(n)+\sum_{k=1}^{p}\hat{a}(k)x(n-k))x^*(n)+\sum_{n=0}^{N-1-p}(x^*(n)+\sum_{k=1}^{p}\hat{a}(k)x^*(n+k))x(n)\right)$$

o finalmente

$$\hat{\sigma}^2=c_{xx}(0,0)+\sum_{k=1}^{p}\hat{a}(k)c_{xx}(0,k)$$

Hay que señalar que la covariancia modificada es idéntica al método de covariancia excepto por la definición de $c_{xx}(j,k)$ y que su resolución dependerá del número de parámetros que se estén tomando en cuenta.

No hay que olvidar que la Covariancia Modificada es una optimización de un método autorregresivo por lo que el cálculo de su densidad de potencia espectral es el mismo que se dio con anterioridad.

5.4.2 IMPLANTACION SECUENCIAL Y PARALELA

Al igual que el algoritmo implantado para la transformada de Fourier, en esta sección se presentan dos programas que permitan demostrar la eficiencia del procesamiento paralelo sobre el procesamiento secuencial, todo esto sobre la base de la plataforma paralela diseñada en el Capítulo 4.

El primer programa es la implantación del algoritmo de covariancia modificada en su versión secuencial. Éste se encuentra dividido en tres procesos y cargado en una red de dos procesadores(véase apéndice C). Dos de los procesos son ejecutados en paralelo en el procesador maestro mientras que el tercer proceso es ejecutado en el procesador esclavo. Aquellos que son ejecutados en el procesador maestro tienen como función, por un lado, leer el archivo de datos con que se trabajará y adquirir una serie de datos requeridos, como es el orden del modelo, para su procesamiento y a su vez transmitirlos al procesador esclavo por medio de un canal físico, por otro lado, el segundo proceso tiene como función recibir la información proveniente del procesador esclavo, desplegarla en pantalla y almacenarla en memoria.

El proceso que se encuentra en el procesador esclavo tiene como función el construir el modelo a partir del algoritmo de covariancia modificada y obtener su densidad de potencia espectral. Para lo cual calculará la matriz de covariancia y la matriz de términos independientes, para después resolver el sistema por el método de CHOLESKY obteniendo con esto los parámetros del modelo buscado; el siguiente paso es el cálculo de la variancia de ruido blanco. Con el modelo encontrado y en base a la ecuación 5.3.1.1 resta transformar a éste en sus componentes de frecuencia por lo que se implemento la trasnformada rápida de Fourier logrando así el modelo en el dominio de las frecuencias. Por último se calculó el módulo cuadrado de cada uno de los datos de la trasnformada con lo que se obtuvo la densidad de potencia espectral.

Por lo que respecta al siguiente programa (véase apéndice C) este tiene la misma función que su versión secuencial solo que el primero podrá procesar el doble de la información en el mismo tiempo tan solo con un sobrepaso dado por las comunicaciones entre los procesadores. Cabe señalar que la información al ser manejada por ventanas de tiempo se dirá que cuando se ejecute el programa en paralelo tendrá la capacidad de procesar dos ventanas al mismo tiempo.

El programa en paralelo tiene una estructura pipeline con dos procesadores esclavos y un procesador maestro; los procesadores esclavos contendrán dos procesos idénticos en los que se ejecutará el cálculo de la PSD por el método de la covariancia modificada, por lo que y como se señalo anteriormente, esta estructura podrá procesar dos ventanas datos al mismo tiempo, tal y como se realizo para la FFT.

El siguiente diagrama 5.13 muestra los pasos a seguir en el cálculo de la Densidad de Potencia Espectral.

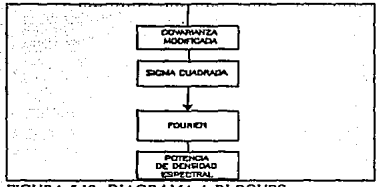


FIGURA 5.13 DIAGRAMA A BLOQUES DEL ALGORITMO PROPUESTO

Del mismo modo que en los programas basados en la FFT se midieron los tiempos de procesamiento, aquí también se llevo al cabo este proceso. Para los dos programas (secuencial y paralelo), los procesos que realizaron el cálculo básico de la PSD estuvieron asignados como procesos de alta prioridad (véase capítulo 3) dando con esto un reloj en tiempo real sobre una base de 100ms . Los resultados fueron utlizados para demostrar la eficiencia del procesamiento paralelo sobre el procesamiento secuencial

5.4.3 ANALISIS DE DESEMPEÑO

Después de haber diseñado el sistema y planteado la aplicación, resta mostrar los resultados que justifican al presente trabajo como una mejor opción para el procesamiento digital de señales en comparación con el desarrollo secuencial.

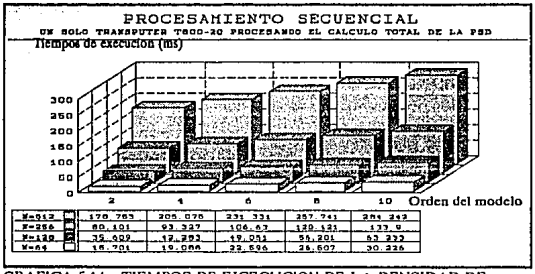
Para lo cual , tomaremos al proceso de covariancia modificada de forma serie y de forma paralela conforme a los programas mostrados en el apéndice C demostrando en sus respectivos tiempos de ejecución (Gráfica 5.14, Gráfica 5.15), que el procesamiento paralelo es más eficiente que su contraparte secuencial como lo muestra la Gráfica 5.16 correspondiente a la eficiencia del sistema.

Recordando lo dicho en el capítulo 2 la eficiencia del sistema se obtiene a partir de la siguiente ecuación

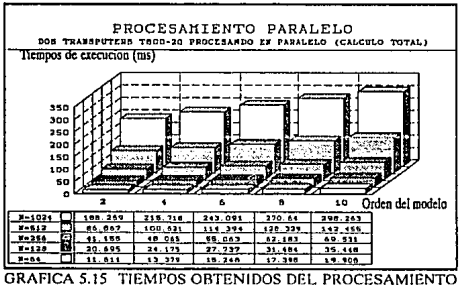
$$ef = \frac{p * t[1]}{t[p] * p} = \frac{t[1]}{t[p]}$$

siendo p el número de procesadores, t[1] el tiempo consumido por un sólo

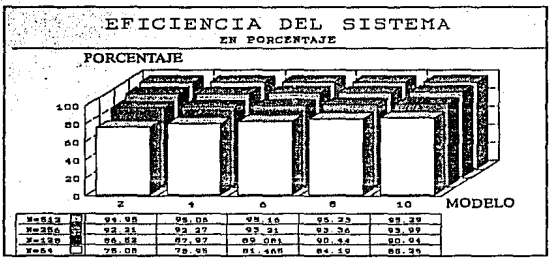
procesador , t[p] el tiempo consumido por p procesadores y la eficiencia escalada.



GRAFICA 5.14 TIEMPOS DE EJCECUCION DE LA DENSIDAD DE POTENCIA ESPECTRAL POR MEDIO DEL METODO DE COVARIANCIA MODIFICADA EN SU VERSION SECUENCIAL



GRAFICA 5.15 TIEMPOS OBTENIDOS DEL PROCESAMIENTO PARALELO DE LA SEÑAL POR MEDIO DEL METODO DE COVARIANCIA MODIFICADA Y SU OBTENCION DE LA DENSIDAD DE POTENCIA ESPECTRAL



GRAFICA 5.16 PORCENTAJES DE EFICIENCIA DEL SISTEMA PARALELO

Como ejemplo tomaremos una ventana de 512 datos y un modelo 6; obteniendo con respecto de la Gráfica 5.14 (procesamiento secuencial) un tiempo de 231.331 milisegundos de tiempo de ejecución , mientras que, para la Gráfica de procesamiento paralelo debemos de tomar una ventana de 1024 datos, porque cada transputer está procesando una ventana de 512 datos . Logrando un tiempo de 215.718 milisegundos . Con lo que su eficiencia será de 95 %.

Habrá que visualizar del anterior ejemplo, que para el procesamiento paralelo cada transputer está procesando 512 datos; con lo que se puede decir que cualquiera de estos están procesando el mismo cúmulo de información que su contraparte secuencial.

En base a éste punto de vista obtenemos para la ventana de 512 datos en el procesamiento paralelo, un tiempo de pérdida de 10 milisegundos, aproximadamente , con respecto procesamiento secuencial el cual es referente al tiempo que se lleva el sistema para comunicarse.

Con estas bases, se puede concluir que el procesamiento paralelo tuvo una eficiencia del 95 % sobre el procesamiento secuencial ; es decir; el procesamiento paralelo logró procesar el doble de información en el mismo tiempo del que lo hizo el procesamiento secuencial.

5.5 ANALISIS COMPARATIVO

Como se señalo en el Capítulo 5 existen varios métodos para obtener la densidad de potencia espectral de cualquier señal, siendo dos los de particular interés en el presente trabajo. Los estimadores espectrales paramétricos y los estimadores espectrales no paramétricos, de los cuales tomaremos un ejemplo de cada uno y los compararemos a partir de los resultados sobre una ventana.

Los ejemplos a tomar serán por el lado de los estimadores espectrales paramétricos el algoritmo de Covariancia Modificada y por el lado de los estimadores no paramétricos la FFT de radio 2. Los programas que se realizaron a partir de la teoría de estos dos ejemplos , se muestran en el apéndice C.

Antes de analizar sus correspondientes resultados hay que partir de cuales son las características que se desean obtener da la Densidad de Potencia Espectral de la señal adquirida. Ya en la introducción de este capítulo se señalo cuales son aquellas que se requieren para hacer un examen del efecto Doppler que para el caso; sería sobre el flujo sanguíneo. Dos son los de mayor importancia; la frecuencia central y el ancho de banda, las demás revisten menor importancia en el análisis del flujo sanguíneo y su efecto Doppler.

Se tomará como muestra base a una ventana de 512 datos que para el caso del método de Covariancia Modificada será sobre todos los ordenes del modelo. En este punto no nos interesa el desarrollo del paralelismo de tal o cual método si no demostrar que la técnica del estimador espectral paramétrico es ma eficiente para obtener la densidad de potencia espectral que los resultados predecidos por la Transformada Rápida de Fourier, sin importar que el modelo obtenido por la covariancia modificada requiera de un mayor número cálculos, pues su espectro será mas definido en cuantos a las características buscadas.

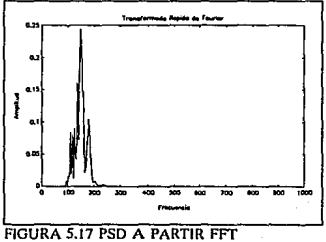


FIGURA 5.17 PSD A PARTIR FFT DE UNA VENTANA DE 512 DATOS

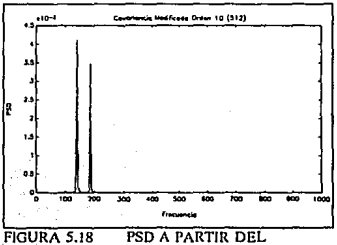


FIGURA 5.18 PSD A PARTIR DEL MODELO DE ORDEN 10 DE COVARIANCIA

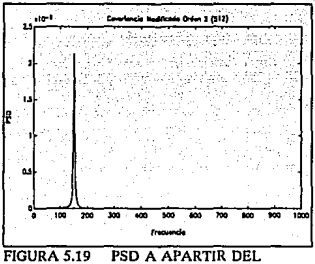


FIGURA 5.19 PSD A APARTIR DEL MODELO DE ORDEN 2 DE COVARIANCIA

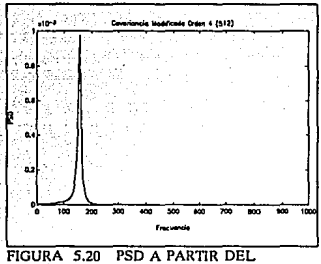


FIGURA 5.20 PSD A PARTIR DEL MODELO DE ORDEN 4 DE COVARIACIA

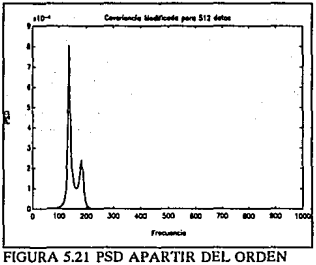


FIGURA 5.21 PSD APARTIR DEL ORDEN 6 DE MODELO DE COVARIANCIA MOD.

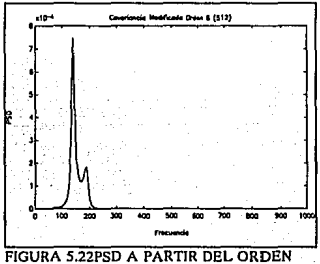


FIGURA 5.22PSD A PARTIR DEL ORDEN 8 DEL MODELO DE COVARIANCIA MOD.

Como se puede observar y en base a las escalas propuestas se obtiene una mayor resolución conforme se va aumentando el orden del modelo hasta obtener mas información en la otorgada por el proceso de covariancia modificada que la obtenida por el modelo no paramétrico.

Observando la Figura 5.18 vemos que se obtienen dos picos prominentes, uno es el referente a la PSD de la ventana analizada , mientras que el segundo es el ruido blanco proveniente de la misma señal[5].

5.5 SUMARIO

En el presente capítulo se analizaron y demostraron las ventajas que se tienen en el procesamiento paralelo sobre el procesamiento secuencial en un campo específico como es el procesamiento digital de señales. Cabe señalar que se puede ir incrementando el rendimiento de forma casi lineal al ir aumentando el número de procesadores en la plataforma, claro está, que se tendrá una perdida de tiempo debida al proceso de las comunicaciones entre la plataforma. Sin embargo, hay que tener en cuenta que en los dos casos de estudio el algoritmo solo se reprodujo en otro procesador, por lo que su concepto como un proceso secuencial no fue atacado siendo esto último un trabajo futuro dentro del proyecto.

Aún cuando el procesamiento requerido en el método paramétrico es mas intenso que su contraparte no paramétrica los resultados obtenidos demuestran que la frecuencia central y el ancho de banda buscados pueden ser analizados de una forma más sencilla que los obtenidos en cálculo del método no paramétrico.

5.7 REFERENCIAS

1.- FISH, PETER; MEDICAL ULTRASOUND, PRENTICE HALL, PRIMER EDICION, 1988

2.- FISH P.J., FLEMING P.J., GARCIA D.F., RUANO M.G., PARALLEL IMPLEMENTATION OF A MODEL BASED SPECTRAL ESTIMATOR FOR DOPPLER BLOOD FLOW INSTRUMENTATION, 8TH INTERNATIONAL PARALLEL PROCESSING SYMPOSIUM (IPPS 94), IEEE, CANCUN, MEXICO, ABRIL 1994, PAGINA 810

3.- FLEMING P.J., GARCIA D.F., PARALLEL PROCESSING IN DIGITAL CONTROL, SPRINGER-VERLAG,1992

4.- GARCIA D.F., PROCESAMIENTO PARALELO EN ESTIMACION ESPECTRAL, XV CONGRESO NACIONAL ACADEMICO DE INGENIERIA ELECTRONICA, INSTITUTO TECNOLOGICO DE CHIHUAHUA, CHIHUAHUA, MEXICO, OCTUBRE 1993,PAGINA 53

5.- KAY, M. STEVEN, MODERN SPECTRAL ESTIMATION, PRENTICE HALL, 1989

6.- LYNN, A PAUL, AN INTRODUCTION TO ANALYSIS AND PROCESSING OF SIGNALS, SEGUNDA EDICION

7.- MANOLAKYS, PROAKIS, DIGITAL SIGNAL PROCESSING, MACMILLAN, SEGUNDA EDICION, 1989

8.- MARPLE S.L., "DIGITAL SPECTRAL ANALYSIS". PRENTICE HALL, ENGLEWOOD CLIFFS, N.J., 1987

9.- VAITKUS P., COBBOLD R. - A COMPARATIVE STUDY AND ASSESSMENT OF DOPPLER ULTRASOUND SPECTRAL ESTIMATION TECHNIQUES PART I: ESTIMATION METHODS, ULTRASOUND IN MEDICINE AND BIOLOGY, 14,661-672,(1988)

CAPITULO VI

CONCLUSIONES

6.1 CONCLUSIONES GENERALES

En este trabajo se ha presentado el diseño de un sistema de procesamiento paralelo basado en la arquitectura del Transputer y en el concepto de modularidad denominado TRAM . Para este sistema se desarrollaron dos módulos iguales que mediante una tarjeta de conexión se comunican entre ellos. A partir de la modularidad se obtiene una de las ventajas más importantes en el desarrollo de sistemas de procesamiento paralelo, que es la capacidad de conexión de diferentes módulos sin la necesidad de modificar al sistema para la comunicación entre ellos.

Dos fueron los campos de mayor importancia en el desarrollo de este proyecto; por un lado, el diseño de la interfase de memoria y por otro, la comunicación con el mundo exterior. Para el primero se plantearon una serie de alternativas que se fundamentaban en el tipo de memoria a usar y la velocidad de procesamiento buscada. Para el segundo campo se buscó darle protección al sistema contra diferentes tipos de descargas que en un momento dado pudieran aparecer entre el mundo exterior y el transputer.

Este sistema se ha utilizado para la implantación de algoritmos de estimación espectral de señales Doppler de Ultrasonido, con técnicas de procesamiento paralelo. En particular dos algoritmos han sido desarrollados; el primero es un método no paramétrico basado en la FFT. El segundo es un método paramétrico basado en el algoritmo de Covariancia Modificada.

Inicialmente estos dos métodos tienen como propósito el demostrar la eficiencia del procesamiento paralelo sobre el procesamiento secuencial, para lo cual se han desarrollado dos versiones de cada método; una de forma secuencial y otra de forma paralela midiendo en ellas los tiempos tomados para el procesamiento de la señal. A partir de los tiempos obtenidos se midió la eficiencia comparando la información obtenida de forma paralela y la de forma secuencial. Los resultados arrojados de esta comparación muestran que en los dos casos (método paramétrico y no paramétrico) la eficiencia de procesamiento paralelo es de alrededor del 90 % para el caso de ventanas de 512 datos.

Ambos métodos han sido evaluados con respecto a su resolución, velocidad de ejecución y eficiencia al ser implantados en su sistema de procesamiento paralelo. Del mismo modo, se ha presentado un análisis cualitativo y cuantitativo de las técnicas de estimación espectral utilizadas. Aún cuando el procesamiento requerido en el método paramétrico es más intenso que su contraparte no paramétrica, los resultados obtenidos

demuestran que la frecuencia central y el ancho de banda buscados pueden ser identificados de una forma más sencilla que los obtenidos en el cálculo del método no paramétrico.

6.2 TRABAJO FUTURO

La presente tesis fue desarrollada dentro de un trabajo de investigación; por lo que en base a los resultados obtenidos aquí se han planteado una serie de alternativas para buscar formas de procesamiento más eficientes que las ya planteadas. Dentro de este marco se proponen diferentes espectos de desarrollo que van desde la implantación de nuevos algoritmos hasta la expansión del sistema. A continuación se muestra una serie de puntos que plantea trabajo futuro asociado a este proyecto.

1.- Se planea incrementar el número de procesadores hasta ahora disponible en este sistema con objeto de incrementar la capacidad del mismo.

2.- La configuración de la Topología de las tarjetas actualmente es realizada por medio de "jumpers". Sin embargo la estrategia más adecuada es el utilizar una matriz de switches programables por medio de software , como la ofrecida por INMOS (crossbar switch IMSC004) por lo que es posible diseñar todo una estructura de comunicación para generar un cierto tipo de topología variable con el software que se este ejecutando.

3.- Se pretetende investigar otras técnicas de implantación de los algoritmos de estimación espectral utilizados que ofrezcan el potencial de un mejor desempeño y mayor eficiencia.

4.- Se probarán algoritmos alternativos al de Covariancia Modificada como es el método de Burg o el método de Máxima Estimación de Probabilidad que también ofrecen el prospecto de una mayor resolución comparada con la obtenida por el método de FFT.

APENDICE A

TRANSPUTER EDUCATION KIT

APENDICE A

El presente apéndice muestra una visión general de la tarjeta que se utilizo como interface entre la PC y la plataforma. No pretendiendo dar una explicación amplia de sus características puesto que no es el objetivo del presente trabajo.

A.1 GENERALIDADES

El TEK (transputer education kit) esta diseñado para la experimentación con transputers y el multiprocesamiento. Todos los elementos activos están puestos en bases para la fácil reparación en el caso que ellos llegáran a dañarse en la experimentación con hardware. La distribución de componentes de esta tarjeta es la mostrada en la Figura A.1 que como se ve, esta distribuida en las 6 areas marcadas por la propia Figura A.1 y la tabla 1 .

AREA	CARACTERISTICA
A1	Aquí se puede comunicar el transputer con el mundo exterior por medio de sus links y enbase a un protocolo que posteriormente se explicara
A2	Al igual que en A1 el trans. utiliza el manejo de un link para comunicarse con la PC .Esta comunicación serie paralela se realiza por medio del COI2 que es un controlador , convertor entre las señales de la PC y el transputer
A3	En esta parte se especifica la dirección en donde se encontrará el transputer en base al mapa de memoria de la PC
A4	Esta parte es una serie de jumpers en la cual se controla la velocidad de la memoria (Mspd) el tamaño de esta (MSz) , la velocidad del transputer (XpuSpd) y la velocidad de los links (LSpd)
A5	Esta área solo contiene el transputer
A6	La última parte muestra la memoria externa de esta tarjeta y una circuitería necesaria para su comunicación con el transputer.

TABLA 1 UBICACION DE COMPONENTES CON RESPECTO A LA TARJETA TEK

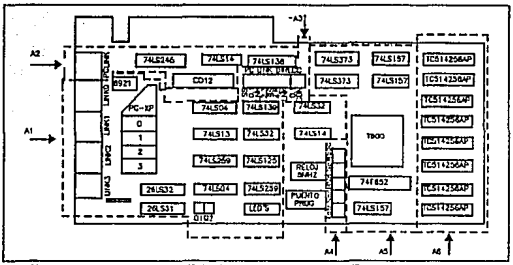


FIGURA A.1 DIAGRAMA FISICO DE LA TARJETA DE DESARROLLO TEK

Ahora bien , acorde con el mapa de memoria del Transputer se construyó el mapa de memoria del TEK como lo muestra la Figura A.2

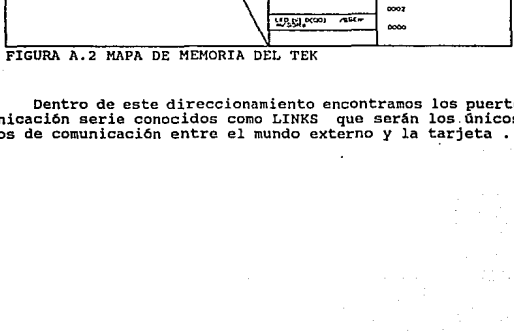


FIGURA A.2 MAPA DE MEMORIA DEL TEK

Dentro de este direccionamiento encontramos los puertos de comunicación serie conocidos como LINKS que serán los únicos medios de comunicación entre el mundo externo y la tarjeta .

Por lo anteriormente dicho es obvio pensar, que esta tarjeta generará todas las palabras paralelas de la PC en paquetes seriales con cierto protocolo. Esto es realizado en el Area 2 de la figura A1 por medio del CO12 el cual es un link adaptador que ofrece una interface con un microprocesador , convirtiendo la transmisión serial en paquetes paralelos y viceversa . El CO12 muestra los Bytes desde la PC a el link del transputer y de forma análoga con los bytes del PCLINK. Este PCLINK puede comunicarse con cualquiera de los canales de comunicación del transputer pero por sencillez es preferible comunicarse con el link cero , dado que , es este el elegido por CSA para que internamente y por medio de jumpers se pueda adaptar la comunicación entre ambos sistemas. La salida que muestra el PCLINK al igual que los otros 4 links están compuestos por una salida y una entrada diferencial así como una serie de señales de protocolo que ya fueron analizadas en el capítulo 4 son : el reset, el error y el analyse .

Por último nos resta hacer incapie en la Figura A.3 donde se muestra un diagráma a bloques de la distribución del equipo aquí utilizado.

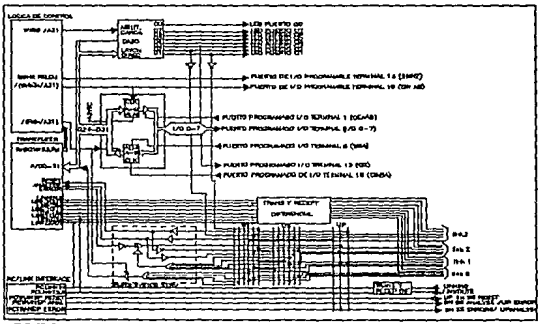


FIGURA A.3 DIAGRAMA A BLOQUES DEL SISTEMA TEK

APENDICE B

PROGRAMAS DE PRUEBA

B.1 PRUEBA DE COMUNICACION Y DE MEMORIAS DE LA PRIMER TARJETA EXTERNA

El siguiente programa es una prueba a la primer tarjeta del sistema, con esta prueba a la primer se verifica el canal de comunicación, el mapeo interno de memoria y el mapeo externo de memoria. La forma de prueba del sistema es en base al envio de paquetes de información los cuales se ubicarán en zonas especificas de la memoria de la tarjeta externa. Para el desarrollo de este programa se realizaron tres procesos ; dos cargados en la tarjeta maestra y uno en la tarjeta esclava. Los dos primeros tienen como objetivo el recibir la información proveniente de diferentes partes del sistema base (la PC) para después transmitirlos en forma serial a través de uno de sus links hacia el TRAM esclavo, además de recibir la información procesada por el TRAM externo y desplegarla en la pantalla para una comparación con los resultados deseados. El proceso que estará alojado en el TRAM externo tiene como recibir información del TRAM maestro y copiarla en la parte de memoria que se desea analizar, después esta información será leída por otra variable completamente diferente para ser transmitida de regreso a la tarjeta externa

PRUB2.PGM

```
-- CONFIGURACION DE LA RED EN BASE A DOS PROCESADORES, UN
-- PROCESADOR MAESTRO Y UN PROCESADOR ESCLAVO

#INCLUDE "hostio.inc"
#INCLUDE "linkaddr.inc"

-- LOS SIGUIENTES PROCESOS SON LLAMADOS PARA SER DISTRIBUIDOS A
-- TRAVES DE LA RED POR MEDIO DE LIBRERIAS QUE ESTAN ALMACENADAS
-- EN UNIDADES TIPO #USE Y QUE SE DISTRIBUIRAN POR MEDIO DEL

-- PROGRAMA QUE CONFIGURA A LA RED

#USE "sortce.clb"
#USE "inoutt.clb"

-- DECLARACION DE CANALES FISICOS PARA SER USADOS EN LA

-- INTERCONEXION DE LA RED

CHAN OF SP fs, ts:
CHAN OF ANY to.pipe,to1.pipe:

-- DECLARACION DE LOS PROCESADORES Y SU FORMA DE INTERCONEXION
-- ASI COMO LA FORMA EN QUE SE DISTRIBUIRAN LOS PROCESOS DENTRO
-- LA RED

PLACED PAR
  PROCESSOR 0 T800
    PLACE ts AT link0.out:
    PLACE fs AT link0.in:
    PLACE to.pipe AT link1.out:
    PLACE to1.pipe AT link1.in:
    inoutt(fs,ts,to.pipe,to1.pipe)
  PROCESSOR 1 T800
    PLACE to.pipe AT  link0.in:
    PLACE to1.pipe AT  link0.out:
    sortce(to.pipe,to1.pipe)
```

SORTEE.OCC

```
-- DECLARACION DE LAS LIBRERIAS ; EN ESTE CASO PARA EL
-- USO DE FUNCIONES MATEMATICAS
#USE"englmath.lib"
-- DECLARACION DEL PROCESO Y DE LOS CANALES DE COMUNICACION
-- CON OTROS PROCESOS
PROC sortce(CHAN OF ANY to.pipe,to1.pipe)
  -- DECLARACION DE LAS VARIABLES A UTILIZAR
  -- EN ESTE CASO SE HAN DECLARADO UNA SERIE
  -- DE VARIABLES TIPO PUERTO QUE SON UTILIZADAS
  -- PARA PROBAR CIERTAS REGIONES DE LA MEMORIA
  PORT OF [100000]REAL32 T0,T1 :
  REAL32 f,ll:
  INT K:
  [100000]REAL32 t00,t11,t01:
  PLACE T0 AT #399:
  PLACE T1 AT #399:
-- CON LA SIGUIENTE ETIQUETA "SEQ" SE INICIA EL PROGRAMA
-- EL CUAL TIENE COMO FUNCION PROBAR LA COMUNICACION Y PROBAR
-- EL BUEN FUNCIONAMIENTO DE LAS MEMORIAS DEL SISTEMA
  SEQ
    -- SE ASIGNA A UNA VARIABLE TIPO REAL UN VALOR, EN
    -- ESTE CASO 0.0
    ll:= 0.0 (REAL32)
    -- SE RECIBE INFORMACION POR MEDIO DEL CANAL DE ENTRADA
to.pipe
    -- Y ES ASIGNADA A UNA VARIABLE TIPO ARREGLO REAL
    to.pipe ? t00
    -- LA VARIABLE DE ARREGLO A LA QUE LE FUE ASIGNADA EL VALOR
DE
    -- t00 AHORA ES ASIGNADA A LA VARIABLE TIPO PUERTO PARA SER
    -- ALMACENADA DENTRO DE UNA REGION EN ESPECIFICO
    T0 ! t00
    -- DESPUES DE HABER SIDO ALMACENDA LA INFORMACION, ESTA ES
    -- LEIDA POR OTRA  VARIABLE TIPO ARREGLO POR MEDIO DE OTRA
    -- VARIABLE TIPO PUERTO
    T1 ? t11
--POR ULTIMO LA INFORMACION RECIBIDA POR t11 ES TRANSMITIDA POR
--MEDIO DE UN CANAL TIPO ANY QUE FUE DECLARADO EN EL INICIO
DEL
```

```
  -- PROGRAMA
  to1.pipe 1 tl1
-- AL FINALIZAR EL PROGRAMA SE REQUIERE QUE ESTE TERMINE CON DOS
-- PUNTOS ":"
:
```

INOUTTT.OCC

```
-- LOS SIGUIENTES "USE" E "INCLUDE" SON USADOS PARA LLAMAR
-- A UNA SERIE DE LIBRERIAS QUE DARAN SOPORTE AL MANEJO DE
-- DE ENTRADA Y/O SALIDA DE INFORMACION CON RESPECTO DEL
-- SISTEMA BASE QUE PARA EL PRESENTE TRABAJO ES LA PC
#INCLUDE "hostio.inc"
#USE "hostio.lib"
-- SE DECLARA EL NOMBRE DEL PROCESO POR MEDIO DE LA ETIQUETA
-- "PROC" AL IGUAL QUE LOS TODOS CANALAES QUE SE VAN A USAR PARA
-- COMUNICARSE CON EL PROCESADOR ESCLAVO O CON EL SISTEMA BASE
-- HAY QUE RECALCAR QUE LA COMUNICACION CON RESPECTO AL
-- PROCESADOR ES POR MEDIO DE UN CANAL TIPO ANY LOGRANDO CON
-- ESTO LA TRANSMISION DE INFORMACION SIN NECESIDAD DE
PROTOCOLOS
PROC inoutt(CHAN OF SP fs, ts, CHAN OF ANY to3.pipe,to4.pipe )
  -- EL SIGUIENTE PASO ES EL DECLARAR LAS VARIABLES A USAR
  REAL32 line, line1:
  [100000]REAL32 acu,kcu:
  BOOL error:
  -- AL IGUAL QUE EL PROCESO SORTEE.OCC Y QUE EN CUALQUIER OTRO
  -- SE INICIARA ESTE CON LA ETIQUETA "SEQ"
  SEQ
    -- ES LEIDO UN VALOR TIPO REAL DESDE EL TECLADO Y ASIGNADO A LA
    -- VARIABLE line QUE TAMBIEN ES DE TIPO REAL

    so.read.echo.real32(fs,ts, line, error)
    --EL VALOR ES ASIGNADO 100000 VECES A UN ARREGLO DE ESTE TAMAÑO

    SEQ i=0 FOR 100000
      SEQ
      acu[i]:= line
    so.write.string(fs,ts,"                ")
-- EL ARREGLO QUE HA SIDO FORMADO ES AHORA TRANSMITIDO A TRAVES
-- DEL CANAL to3.pipe HACIA EL PROCESADOR ESCLAVO
    to3.pipe ! acu
    so.write.string(fs, ts,"                ")
    so.write.real32(fs, ts,acu[9],0,0)
    -- POR MEDIO DE OTRA VARIABLE TIPO ARREGLO ES RECIBIDO EL
    -- RESULTADO DEL PROCESO EJECUTADO EN EL SEGUNDO PROCESADOR
    to4.pipe ? kcu
    -- PASO SIGUIENTE ES MOSTRAR LOS RESULTADOS DEL PROCESAMIENTO
    -- QUE PARA ESTE CASO ES REGRESAR LA INFORMACION TAL Y COMO SE

    -- MANDO POR LO QUE SE MOSTRARA EL ARREGLO DE RESULTADOS AL

    -- MISMO TIEMPO QUE EL ARREGLO QUE SE FORMO AL INICIO DEL
    -- PROGRAMA
    SEQ i=0 FOR 100000
      SEQ
        so.write.string(fs,ts,"*c*n*c                ")
        so.write.real32(fs,ts,acu[i],0,0)
        so.write.string(fs,ts,"                ")
        so.write.real32(fs,ts,kcu[i],0,0)
  -- EL PROCESO INOUTTT.OCC DADO QUE ESTA EJECUTANDOSE DE EN EL
  -- PROCESADOR HOST, TIENE LA FUNCION DE TRABAJAR COMO
  -- SERVIDOR DEL RED QUE SE HA PLANTEADO, POR LO QUE LA FINALIZAR
  -- ESTE PROCESO FINALIZARA TODO PROGRAMA DE PRUEBA, PARA QUE
  -- SUCEDA ESTO DE FORMA EXITOSA SE REQUIERE DEL PROCESO
  -- so.exit
    so.exit(fs,ts,sps.success)
--COMO SE SEÑALO ANTERIORMENTE CUALQUIER PROCESO DEBE
FINALIZAR
-- CON ":"
:
```

B.2 PRUEBA DE COMUNICACION Y DE MEMORIA DE LA SEGUNDA TARJETA EXTERNA

El siguiente programa cumple los mismos objetivos que el programa anterior, solo que este probará a la segunda tarjeta, para lo cual usará a la primer tarjeta como nodo de comunicación entre la tarjeta maestra y ella.

Al igual que en programa anterior la tarjeta maestra ejecutará dos procesos; el primero de recepción de datos provenientes de la PC y la transmisión de estos hacia el sistema de prueba y el segundo de recepción de datos del sistema de prueba así como almacenamiento de los mismos en un disco de la PC. Cabe señalar que estos procesos son exactamente los mismos que los utilizados en el programa anterior por lo que no serán repetidos en esta ocasión.

La primer tarjeta esclava, como se menciono servirá como nodo enlace entre la transmisión del sistema maestro y la segunda tarjeta esclava, por lo que utilizará el proceso " TRANSMIT.OCC" el cual sólo recibirá información a partir de uno de uno de los canales y transmitira a través de otro.

La segunda tarjeta tendrá el mismo proceso que la primer tarjeta en el programa "PRUB2.PGM" por lo que guardará información en su memoria externa para despues ser leída y transmitida de regreso.

PRUB3.PGM

```
-- EN PRINCIPIO DE CUENTAS SE DECLARAN TODAS HERRAMIENTAS
-- Y LIBRERIAS REQUERIDAS PARA CONSTRUIR LA RED
#INCLUDE "hostio.inc"
#INCLUDE "linkaddr.inc"
-- A DIFERENCIA DE PRUB2.PGM AQUI SE VAN A UTILIZAR TRES
-- PROCESADORES UNO DE ELLOS SERVIRA SOLO COMUNICADOR ENTRE
-- LOS OTROS DOS, LA PRUEBA QUE SE REALIZARA SERA EXACTAMENTE
-- IGUAL A LA HECHA EN PRUB2 SOLO QUE PROBARA AL TERCER
-- PROCESADOR
#USE "sastoc.c4h"
#USE "inoutc.c4h"
#USE "transmit.c4h"
-- LA DECLARACION DE CANALES DE COMUNICACION DEBE DE SER PARA
-- TODOS AQUELLOS CANALES FISICOS QUE SE VAN A USAR ADEMAS DE
-- DETERMINAR EL TIPO DE DATOS QE SE TRANSMITIRAN A TRAVES DE
-- ELLOS. PARA ESTE CASO SE USARAN CANALES TIPO ANY CON LO CUAL
-- SE PUEDEN TRANSMITIR CUALQUIER TIPO DE DATOS
CHAN OF SP fs, ts:
CHAN OF ANY to.pipe,to1.pipe,to2.pipe,to3.pipe:
-- LA ASIGNACION DE PROCESOS EN LOS PROCESADORES SE LLEVA A CABO
-- BAJO LA ETIQUETA DE "PLACED PAR" AL IGUAL QUE LA DECLARACION
-- DE CANALES Y SU FUNCION PARA EL PROCESADOR (ENTRADA O SALIDA)
PLACED PAR
  PROCESSOR 0 T800
    PLACE ts AT link0.out:
    PLACE fs AT link0.in:
    PLACE to.pipe AT link1.out:
    PLACE to1.pipe AT link1.in:
    inoutc(fs,ts,to.pipe,to1.pipe)
  PROCESSOR 1 T800
    PLACE to.pipe AT link0.in:
    PLACE to1.pipe AT link0.out:
    PLACE to2.pipe AT link1.in:

    PLACE to3.pipe AT link1.out:
    HP50o(to.pipe,to1.pipe,to2.pipe,to3.pipe)
  PROCESSOR 2 T800
    PLACE to3.pipe AT link0.in:
    PLACE to2.pipe AT link0.out:
    sortc(to3.pipe,to2.pipe)
```

TRANSMIT.OCC

```
-- DADO QUE ESTE PROCESO SOLO SERVIRA DE ENLACE ENTRE EL
-- PROCESADOR MAESTRO Y EL SEGUNDO PROCESADO ESCLAVO NO
-- SE REQUIERE NINGUN TIPO DE LIBRERIA
PROC HP50(CHAN OF ANY P1,P2,P3,P4)
  -- LA DECLARACION DE VARIABLES SE REALIZO DE FORMA SIMILAR
  -- A LOS PROCESOS ANTERIORMENTE SEÑALADOS
  [10000]REAL32 X.A:
  INT MODE,IP.N:
  REAL32 EP:
  SEQ
    -- COMO SE PODRA VER LA UNICA FUNCION QUE SE REALIZO AQUI
    -- ES EL RECIBIR INFORMACION A TRAVES DE UN CANAL TIPO ANY
    -- Y TRANSMITIRLA POR OTRO DE SU MISMO TIPO
    P1 ? X
    P4 ! X
    P3 ? X
    P2 ! X
```

APENDICE C

PROGRAMAS DE APLICACION

C.1 PROGRAMA PARA CALCULAR LA DENSIDAD DE POTENCIA ESPECTRAL EN BASE A LA TRANSFORMADA DE FOURIER-DESARROLLADO EN UN MODULO MONITOR Y EN UN MODULO TRABAJADOR

El presente programa está desarrollado en base a la teoría expuesta en el Capítulo 5 , es decir , este programa está basado en un algoritmo de la FFT en decimación en tiempo de radix 2 , no optimizado. Consta de 4 partes básicas:

1.- La configuración de la red (FPS.PGM)

2.- Proceso captador de datos y transmisor de estos al modulo trabajador(modulo monitor EDFPS.OCC)

3.- Proceso desarrollado para calcular la FFT y la densidad de potencia espectral (modulo trabajador CFPS.OCC)

4.- Proceso receptor de datos del modulo trabajador y generador del archivo final (modulo monitor SDFPS.OCC)

Como podemos ver en la parte uno, el sistema está compuesto de dos módulos los cuales procesarán la información de forma paralela y se comunicarán entre ellos por medio de los canales P1 y p2. Por otro lado el procesador monitor se transmitira con el sistema base , que en este caso es la PC, por medio de los canales fs y ts.

Es importante señalar que la transmisión realizada por los canales P1 y P2 es declarando a estos de la forma anárquica dado que , esto nos permitirá la transmisión de cualquier tipo de datos sin necesidad de la verificación de los protocolos de los canales. Sin embargo, habrá que tener cuidado en la utilización de los canales anárquicos , pues, se tiene que buscar una sincronía entre el tipo de datos en la transmisión y el tipo de datos en la recepción , porque de no lograrse esto , no se podrán comunicar los módulos.

El proceso transmisor de datos, es un proceso de tipo secuencial que toma los datos de un archivo que es especificado por el usuario al igual que otro tipo de elementos necesarios para el proceso de la transformada. Ya habiendo recopilado dichos elementos el proceso envía los datos através del canal P1 con rumbo del proceso trabajador , después de esto termina su ejecución.

Ya en el proceso trabajador lo primero que se realiza es el cálculo de los TWIDLE FACTORS para después llevar el reacomodo de los datos de la forma bit reverse ; logrado esto se procede al cálculo de la transformada de fourier en base a las mariposas requeridas. Por último se

realiza el cálculo de la densidad de potencia espectral que no es más que el módulo cuadrado de cada uno de los datos obtenidos de la FFT. Después de realizar este proceso resta transmitir los datos resultantes al procesador monitor por medio del canal P2.

De regreso en el procesador monitor , está tendrá un segundo proceso corriendo, el cual se avocará a recibir los datos transmitidos por el procesador trabajador por medio del canal P2. Posteriormente mostrará estos resultados en la pantalla y generará un archivo, que también deberá ser especificado por el usuario, para contener los resultados finales de este programa.

Por último se muestra un diagráma a bloques de los pasos seguidos en el anterior programa y la red configurada.

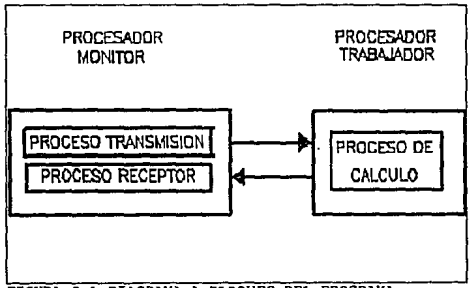


FIGURA C.1 DIAGRAMA A BLOQUES DEL PROGRAMA

FPS.PGM

```
#INCLUDE "hostio.inc"
#INCLUDE "streamio.inc"
#INCLUDE "mathvals.inc"
#INCLUDE "linkaddr.inc"
#USE "cdfps.clb"
#USE "cfps.clb"
#USE "sdfps.clb"
CHAN OF SP fs,ts:
CHAN OF ANY P1,P2:
PLACED PAR
  PROCESSOR 0 T800
    PLACE ts AT link0.out:
    PLACE fs AT link0.in:
    PLACE P1 AT link1.out:
    PLACE P2 AT link1.in:
  SEQ
    H1(fs,ts,P1)
    H4(fs,ts,P2)
  PROCESSOR 1 T800
    PLACE P1 AT link0.in:
    PLACE P2 AT link0.out:
    H2(P1,P2)
```

EDFPS.OCC

```
--{{{  INCLUDEs
#INCLUDE "hostio.inc" --
#INCLUDE "mathvals.inc"
#INCLUDE "streamio.inc" --
--}}}
--{{{  USEs
```

```
#USE "hostio.lib"
#USE "streamio.lib"
#USE "string.lib"
#USE "convert.lib"
--}}}
PROC HI(CHAN OF SP fs, ts, CHAN OF ANY PI)
 --{{{  PROC lectura de archivo
 PROC lec.arch.fich(VAL INT len, VAL []BYTE filename,
                    INT j, []REAL32 data, BYTE bres)
  --{{{  principal
  -- Toma los datos desde un archivo
  SEQ
    CHAN OF K5 fickeys:
    CHAN OF K5 kryboard IS fickeys:
    PAR
      ------------------------------------------------------
      so.KeyStr.ext.from.file (fs, ts,
                               keyboard,[filename FROM 0 FOR len], bres)
      ------------------------------------------------------
      REAL32 x:
      INT kchar:
      SEQ
        j := 0  --90
        --{{{  lee una secuencia de numeros reales
        kchar := 0
        x := 1.0(REAL32)
        WHILE (kchar <> ft.terminated)
          SEQ
            ks.read.char (keyboard, kchar)
            IF
              kchar < 0
                SKIP
              TRUE  --100
                ks.read.real32 (keyboard, x, kchar)
            IF
              kchar = ft.terminated
                SKIP
              TRUE
                SEQ
                  IF
                    kchar = ft.number.error
                      data[j] := INFINITY
                    TRUE  --110
                      SKIP
                  data[j] := x
                  j := j + 1
        --}}}
        --{{{  consume el resto del archivo
        IF
          (kchar >= 0) OR (kchar = ft.number.error)
            ks.keystream.vis8 (keyboard) -- consume  el  resto del
                                         -- archivo
          TRUE
            SKIP -- el archivo ha terminado
        --}}}
    IF
      bres <> spr.ok
        SKIP
      TRUE
        SEQ
          so.write.string(fs, ts, "  fallo la lectura  ")
          so.write.string.nl(fs, ts,[filename FROM 0 FOR len])
      ------------------------------------------------------
  --}}}
: --131
--}}}
INT index1,M,index2, LL,IP,N,H:
REAL32 C,SUM,TT1,TT2,SIG2,TTT,TT3,TT4:
REAL32 EP:
[640]REAL32 X,A1:
INT MODE,largo,largo1,N,L :
BOOL ERR,error :
INT IFLAG,KK,K:
INT32 streamid,str2:
VAL []BYTE sig IS "signal" :
BYTE result,res2:
BYTE tes :
[60]BYTE filename: --170
INT len,TT,PP,nomen,nonono:
BYTE res:
SEQ
  --{{{  lectura de  datos del archivo
  --{{{  tomar datos del archivo
  so.write.string.nl(fs, ts,
   "c*n*c nombre del archivo de entrada  ? ")
  so.read.echo.line(fs, ts, len, filename, res)
  so.write.nl(fs, ts)  --180
  so.write.string(fs,ts," c*n*c*n*c DAR EL NUMERO DE MARIPOSA  ")
  so.read.echo.int(fs,ts,M,error)
  --}}}
  lec.arch.fich(len,filename,N,X,res)
  --{{{  despliegue de datos si estan  bien
  IF
    res = spr.ok
      SEQ
        so.write.string(fs, ts, "el archivo de entrada  ")
        so.write.string(fs, ts, [filename FROM 0 FOR len])
        so.write.string(fs, ts, "  tiene  ")
        so.write.int(fs, ts, N, 0)
        so.write.string(fs, ts, " valores  reales  ") --190
        so.write.nl(fs, ts)
        SEQ i = 0 FOR N
          SEQ
            so.write.string(fs, ts, "[")
            so.write.int(fs, ts, i, 0)
            so.write.string(fs, ts, "]= ")
            so.write.real32(fs, ts, X[i], 0, 0)
            so.write.nl(fs, ts)
    TRUE
      SKIP
  --}}}
  --}}}
  PI ! X
  PI ! M
```

CFPS.OCC

```
#USE "occam3b.lib"
#USE "mgtmath.lib"
PROC H2(CHAN OF ANY PI,PS1)
 {{{  declracion de variables
 INT index1,index2, LL,IP,N,H:
 REAL32 W11,C,SUM,TT1,TT2,SIG2,TTT,TT3,TT4:
 REAL32 EP, PI, ARG,SAVE1,SAVE3,SAVE2:          --60
 [640]REAL32 A,XY,Y1,Y2: --10
 BYTE res:
 [10][640]REAL32  W1,W2:
 INT L,MJ,NUM,LK,LDFT,NDFT,NP,NQ:
 TIMER clock:
 INT IFLAG,KK,K:
 INT  dif,tiempo,tiempo1:
```

```
INT len,TT,PP:
}}}
PRI PAR
  SEQ
    P1 ? A
    P1 ? M
    C := POWER (2.0 (REAL32),REAL32 ROUND M)
    N:= INT TRUNC (C)
    PI := 3.141592654 (REAL32)
    {{{  calculo de los twiddle factor
    LDFT: =1
    NDFT: =N                    -30
    SEQ K=1 FOR M
      SEQ
        LDFT := (2* LDFT)
        NDFT := (NDFT/2)
        SEQ I=1 FOR NDFT
          SEQ
            SEQ J=1 FOR (LDFT/2)
              SEQ
                NP := J + (LDFT*(I-1))-40
                ARG: = 0.0 (REAL32)-(((2.0(REAL32)*PI)*((REAL32ROUND J)- 1.0
(REAL32)))/(REAL32 ROUND  (LDFT)))
                W1[K][NP]: = COS(ARG)
                W2[K][NP]: = SIN(ARG)
    }}}
    SEQ I=1 FOR N
      SEQ
        XY[I]: =A[I-1]
    Y1[1]: =XY[1]
    {{{  reacomodo de  datos en bit reverse
    SEQ I=2 FOR N-1
      SEQ
        NUM: = I-1                  -50
        J: =0
        L: =N
        SEQ K=1 POR M
          SEQ
            L:= (L/2)
            IF
              NUM  > = L
                SEQ
                  TT2: =POWER(2.0 (REAL32),REAL32 ROUND (K-1))
                  J:= J  + (INT TRUNC (TT2))  -60
                  NUM: = NUM-L
              NUM < L
                SKIP
        Y1[J]: =XY[J + 1] -- SIGUE EL CALCULO DE FFT LINEA 188
    }}}
    SEQ I=1 FOR N
      Y2[I] := 0.0 (REAL32)
    clock ? tiempo
    W11 := 0.0 (REAL32)
    {{{  calculo de las mariposas de la  fft
    LDFT := 1
    NDFT := N
    SEQ K=1 FOR M -- SE CALCULA LOS ESTADOS DE MARIPOSA
      SEQ
        LDFT := 2*LDFT
        NDFT := (NDFT/2 )
        SEQ I=1 FOR NDFT -- SE CALCULA LAS COMBINACIONES EN CADA
ESTADO
          SEQ
            SEQ J=1 FOR (LDFT/2) -- COMBINACIONES ENTRE ELEMENTOS
              SEQ
                NP:= J + (LDFT*(I-1))          -80
                NQ: =NP+(LDFT/2)
                SAVE1: = Y1[NP]+((W1[K][NP]*Y1[NQ])-(W2[K][NP]*Y2[NQ])
                SAVE2: = Y2[NP]+((W1[K][NP]*Y2[NQ])+(W2[K][NP]*Y1[NQ])
                SAVE3: = Y1[NP]+((W2[K][NP]*Y2[NQ])-(W1[K][NP]*Y1[NQ])
                Y2[NQ]: = Y2[NP]-((W1[K][NP]*Y2[NQ])+(W2[K][NP]*Y1[NQ]))
                Y1[NP]: =SAVE1
                Y2[NP]: =SAVE2
                Y1[NQ]: =SAVE3
    }}}
    {{{  calculo de la DSP
    SEQ I=1 FOR (N/2)
      XY[I] : = ((Y1[I]*Y1[I])+(Y2[I]*Y2[I]))
    }}}
    clock ? tiempo1
    dif := (tiempo1 - tiempo )
    W11 := (REAL32 ROUND dif)
    {{{  trasmision de informacion
    PS1 ! W11

    PS1 ! (N/(2 (INT)))
    PS1 ! XY
    }}}
  SKIP
:
```

SDFPS.OCC

```
--{{{  INCLUDEs
#INCLUDE "hostio.inc" --
#INCLUDE "streamio.inc" --
--}}}
--{{{  USEs
#USE "hostio.lib"
#USE "streamio.lib"
#USE "string.lib"
--}}}
PROC H4(CHAN OF SP fs, ts, CHAN OF ANY PS1)
  --{{{  PROC escritura de datos al fichero
  PROC esc.arch.fich(VAL INT len, VAL [][BYTE filename,
                     INT jdata, [][REAL32 data, BYTE bres)
    --{{{  principal
    INT32 id:
    SEQ
      so.open(fs,ts,[filename FROM 0 FOR len],spt.text,spm.output,id,bres)
      IF  --140
        bres = spr.ok
          SKIP
        TRUE
          so.write.string nl(fs, ts, " el archivo no fue abierto  ")
      SEQ I = 1 FOR jdata
        SEQ
          so.fwrite.real32(fs, ts, id, data[i], 0, 8, bres)
          so.fwrite.nl(fs, ts, id, bres)
      so.close(fs, ts, id, bres)
      IF
        bres = spr.ok
          SKIP
        TRUE
          so.write.string.nl(fs, ts, " el archivo no fue cerrado ")
    --}}}
:
--}}}
INT index1,index2, LL,IP,N:
REAL32 C,SUM,TT1,TT2,SIG2,H,TTT,TT3,TT4:
REAL64 EP:
```

```
[640]REAL32 CC,A,A1;
INT MODE,largo,largo1,N,L :
BOOL error,error :
INT IPLAO,KK,K,tiempo:
INT32 streamid,str2:
VAL []BYTE sig IS "signal" :
BYTE result,res2:
BYTE tres :
[60]BYTE filename:
INT len,TT,PP:
BYTE res:
SEQ
--{{{ proceso de recuperacion de informacion de la red
  PSI ? IP
  PSI ? TT3
  PSI ? A
--}}}
  --{{{  muestra de la informacion obtenida

  so.write.string(fs,ts,"*c*n*c      el tiempo es :  ")
  so.write.string(fs,ts,"*c*n*c*n fourier      ")
  so.write.real32(fs,ts,TT3,0,0)
  so.write.string(fs,ts,"    *n*c*n*c    ")
--}}}
  --{{{  escritura de datos en un archivo
  --{{{  nombre del archivo de salida
  so.write.string.nl(fs, ts, "dar archivo de salida ? ")
  so.read.echo.line(fs, ts, len, filename, res)
  so.write.nl(fs, ts)
  --}}}
  esc.arch.fich(len,filename,IP,A,res)
  --{{{  despliegue de datos
  SEQ
    so.write.string(fs, ts, "archivo  de salida ")
    so.write.string(fs, ts, [filename FROM 0 FOR len])
    so.write.string(fs, ts, " tiene ")
    so.write.int(fs, ts, IP, 0)
    so.write.string(fs, ts, " valores reales  ")
    so.write.nl(fs, ts)
    SEQ i = 1 FOR IP
      SEQ
        so.write.real32(fs, ts, A[i], 0, 0)
        so.write.nl(fs, ts)
  --}}}
  --}}}
  so.exit (fs,ts, sps.success)
:
```

C.2 PROGRAMA PARA CALCULAR LA DENSIDAD DE POTENCIA ESPECTRAL EN BASE A LA FFT-DESARROLLADO EN UN MODULO MONITOR Y DOS MODULOS TRABAJADORES

El presente programa está desarrollado en base a la teoría expuesta en el Capítulo 5 , es decir , este programa está basado en un algoritmo de la FFT en decimación en tiempo de radix 2 , no optimizado. Consta de 4 partes básicas:

1.- La configuración de la red (FPP.PGM)
2.- Proceso captador de datos y transmisor de estos al módulo trabajador(módulo monitor EDFPS.OCC); este programa fue ya expuseto con anterioridad por lo que no será mostrado en esta ocasión
3.- Proceso desarrollado para calcular la FFT y la densidad de potencia espectral (modulo trabajador 1 HP2FFT2.OCC)
4.- Proceso desarrollado para calcular la FFT y la densidad de potencia espectral (modulo trabajador 2 HP2FFT1.OCC
5.- Proceso captador de la información proveniente del procesador maestro distribuidor de ésta hacia los procesadores trabajadores
6.- Proceso receptor de datos del modulo trabajador y generador del archivo final (modulo monitor HP4FFT2.OCC)

Al igual que el primer programa este cumplira con las mismas funciones solo que tendrá la ventaja de procesar el doble de información en el mismo tiempo en que lo hace el programa anterior.

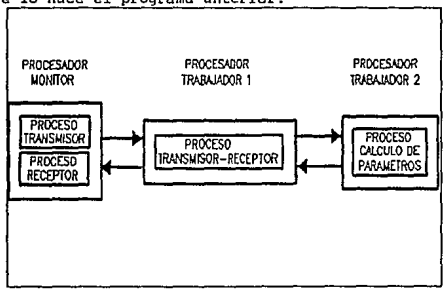


FIGURA C.2 DIAGRAMA A BLOQUES DEL PROCESO

FPP.PGM - DECLARACION DE LIBRERIAS USADAS EN LOS DIFERENTES PROCESOS

FPP.PGM

```
-- DECLARACION DE LIBRERIAS USADAS EN LOS DIFERENTES PROCESOS
#INCLUDE "hostio.inc"
#INCLUDE "streamio.inc"
#INCLUDE "mathvals.inc"
#INCLUDE "linkaddr.inc"
-- DECLARACION DE PROCESOS
#USE "hP1FPT.c8h"
#USE "HP2FPT2.C8H"
#USE "hP2FPT1.c8h"
#USE "hP4fft2.c8h"
#USE "hp3fft1.c8h"
-- DECLARACION DE CANALES DE COMUNICACION
CHAN OF SP fs,ts:
CHAN OF ANY P1,P2,P3,P4:
--  DISTRIBUCION DE PROCESOS Y PROCESADORES ASI COMO
DECLARACION DE CANALAES
-- DE COMUNICACION
PLACED PAR
  PROCESSOR 0 T800
    PLACE ts AT  link0.out:
    PLACE fs AT link0.in:
    PLACE P1 AT link1.out:
    PLACE P2 AT link1.in:
    SEQ
      H1(fs,ts,P1)
      H4(fs,ts,P2)
  PROCESSOR 1 T800
    PLACE P1 AT link0.in:
    PLACE P2 AT link0.out:
    PLACE P3 AT link1.in:
    PLACE P4 AT link1.out:
    CHAN OF ANY P5:
    PAR
      --HP5(P1,P4,P5)
      H22(P1,P3,P2,P4)
  PROCESSOR 2 T800
    PLACE P4 AT link0.in:
    PLACE P3 AT link0.out:
    H2(P4,P3)
```

Los siguentes dos procesos son exactamanete iguales y siguen la misma secuencia que el proceso cfps.occ por lo que los comentarios quedan designados al proceso ya mencionado

HP2FFT2.OCC

```
#USE "occamFh.lib"
#USE "maplmath.lib"
PROC H2([CHAN OF ANY P1,P5!)
  INT index1, index2, L1, IP, N, H:
  REAL32 W11,C,SUM,TT1,TT2,SIG2,TTT,TT3,TT4:
  REAL32 EP, PI, ARG,SAVE1,SAVE3,SAVE2:          --60
  [517]REAL32 A.XY,Y1,Y2: --10
  BYTE rrx:
  [10][517]REAL32  W1,W2:
  INT L,M,J,NUM,I,K,LDPT,NDPT,NP,NQ:
  TIMER clock:
  INT IFLAG,KK,K,11,I1:
  INT  dif,tiempo,tiempo1:
  INT kn,TT,PP:
  PRI PAR
    SEQ
      PI ? A
      PI ? M
      clock ? I1
      W11 := 0.0 (REAL32)
      C:= 1.0 (REAL32)
      SEQ  K=1  FOR M
        SEQ
          C:= ((2.0 (REAL32))*C)
      N:= INT TRUNC (C)
      PI := 3.141592654 (REAL32)
      LDPT:=1
      NDPT:=N                                    --30
      SEQ K=1 FOR M
        SEQ
          LDPT := (2* LDPT)
          NDPT := (NDPT/2)
          SEQ I=1 FOR NDPT
            SEQ
              SEQ J=1 FOR (LDPT/2)
                SEQ
                  NP := J+(LDPT*(I-1))--40
                  ARG := 0.0 (REAL32)-((2.0(REAL32)*PI)*((REAL32ROUND J)-1.0
(REAL32))/(REAL32 ROUND  (LDPT)))
                  W1[K][NP]:= COS(ARG)
                  W2[K][NP]:= SIN(ARG)
      --C:= 1.0 (REAL32)
      --SEQ  I=1  FOR (M+1)
      --  SEQ
      --    C:= ((C* (2.0 (REAL32)))
      --K:= INT TURNC (C)
      SEQ I=1 FOR N
        SEQ
          XY[I]:=A[N+(I-1)]
      Y1[1]:=XY[1]
      SEQ I=2 FOR N-1
        SEQ
          NUM:= I-1                              --50
          J:=0
          L:=N
          SEQ K=1 FOR M
            SEQ
              L:= (L/2)
              IP:
                NUM > = L
                  SEQ
                    TT2:=POWER(2.0 (REAL32),REAL32 ROUND (K-1))
                    J:= J + (INT TRUNC (TT2))  --60
                    NUM:= NUM-L
                NUM < L
                  SKIP
          Y1[I]:=XY[J+1] -- SIGUE EL CALCULO DE FFT LINEA 188
      SEQ I=1 FOR N
        Y2[I] := 0.0 (REAL32)
      LDPT := 1
      NDPT := N
      SEQ K=1 FOR M -- SE CALCULA LOS ESTADOS DE MARIPOSA
        SEQ
          LDPT := 2*LDPT
          NDPT := (NDPT/2 )
```

```
      SEQ I=1 FOR NDFT  -- SE CALCULA LAS COMBINACIONES EN CADA
ESTADO
        SEQ
          SEQ J=1 FOR (LDFT/2) -- COMBINACIONES ENTRE ELEMENTOS
            SEQ
              NP:= J+(LDFT*(I-1))          -80
              NQ:=NP+(LDFT/2)
              SAVE1:= Y1[NP]+((W1[K][NP]*Y1[NQ])-(W2[K][NP]*Y2[NQ]))
              SAVE2:= Y2[NP]+((W1[K][NP]*Y2[NQ])+(W2[K][NP]*Y1[NQ]))
              SAVE3:= Y1[NP]+((W2[K][NP]*Y2[NQ])-(W1[K][NP]*Y1[NQ]))
              Y1[NP]:=SAVE1
              Y2[NP]:=SAVE2
              Y1[NQ]:=SAVE3
      SEQ I=1 FOR (N/2)
        XY[I] := ((Y1[I]*Y1[I])+(Y2[I]*Y2[I]))
      clock ? I2
      dif := (I2-I1)
      W11 := (REAL32 ROUND dif)
      PS1 ! W11


      PS1 ! (N/(2 (INT)))
      PS1 ! XY
    SKIP
:
```

HP2FFT1.OCC

```
#USE "occanlib.lib"
#USE "maglmath.lib"
PROC H22(CHAN OF ANY P1,P3,PS1,P4)
  INT index1,C,index2, LL,IP,N,H:
  REAL32 W11,SUM,TT1,TT2,SIG2,TTT,TT3,TT4:
  REAL32 EP, PI, ARG,SAVE1,SAVE2,SAVE3:          -60
  [517]REAL32 A:
  [517]REAL32 XY,Y1,Y2: --10
  BYTE rec:
  [10][517]REAL32  W1,W2:
  INT L,M,J,NUM,I,K,LDFT,NDFT,NP,NQ:
  TIMER clock:
  INT IFLAG,KK,K:
  INT  dif,tiempo,tiempo1:
  INT ten,TT,PP:
  PRI PAR
    SEQ
      P1 ? A
      P1 ? M
      clock ? tiempo
      P4 ! A
      P4 ! M
      N:=1 (INT)
      SEQ K=1 FOR M
        N := (2 * N)                 --70

      --N:= C
      PI := 3.14159265 (REAL32)
      LDFT:=1
      NDFT:=N                        -80
      SEQ K=1 FOR M
        SEQ
          LDFT := (2* LDFT)
          NDFT := (NDFT/2)
          SEQ I=1 FOR NDFT
            SEQ
              SEQ J=1 FOR (LDFT/2)
                SEQ
                  NP := J+(LDFT*(I-1))--40
                  ARG:= 0.0 (REAL32)-((2.0(REAL32)*PI)*((REAL32ROUND J)- 1.0
(REAL32)))/(REAL32 ROUND  (LDFT))
                  W1[K][NP]:= COS(ARG)
                  W2[K][NP]:= COS((ARG-(PI/(2.0(REAL32)))))
        SEQ I=1 FOR N
          SEQ
            XY[I]:= A[I-1]
        Y1[1]:=XY[1]
        SEQ I=2 FOR N-1
          SEQ
            NUM:= I-1                  -50
            J:=0
            L:=N
            SEQ K=1 FOR M
              SEQ
                L:= (L/2)
                IF
                  NUM > = L
                    SEQ
                      TT2:=POWER(2.0 (REAL32),REAL32 ROUND (K-1))
                      J:= J + (INT TRUNC (TT2))  --60
                      NUM:= NUM-L
                  NUM < L
                    SKIP
            Y1[I]:=XY[J+1] -- SIGUE EL CALCULO DE FFT LINEA 188
        SEQ I=1 FOR N
          Y2[I]:=0.0 (REAL32)
        LDFT := 1
        NDFT := N
        SEQ K=1 FOR M -- SE CALCULA LOS ESTADOS DE MARIPOSA
          SEQ
            LDFT := 2*LDFT
            NDFT := (NDFT/2 )
            SEQ I=1 FOR NDFT  --SE CALCULA LAS COMBINACIONES EN CADA
ESTADO
              SEQ
                SEQ J=1 FOR (LDFT/2) -- COMBINACIONES ENTRE ELEMENTOS
                  SEQ
                    NP:= J+(LDFT*(I-1))          -80
                    NQ:=NP+(LDFT/2)
                    SAVE1:= Y1[NP]+((W1[K][NP]*Y1[NQ])-(W2[K][NP]*Y2[NQ]))
                    SAVE2:= Y2[NP]+((W1[K][NP]*Y2[NQ])+(W2[K][NP]*Y1[NQ]))
                    SAVE3:= Y1[NP]+((W2[K][NP]*Y2[NQ])-(W1[K][NP]*Y1[NQ]))
                    Y3[NQ]:= Y2[NP]-((W1[K][NP]*Y2[NQ])+(W2[K][NP]*Y1[NQ]))
                    Y1[NP]:=SAVE1
                    Y2[NP]:=SAVE2
                    Y1[NQ]:=SAVE3
        SEQ I=1 FOR (N/2)
          SEQ
            XY[I] := ((Y1[I]*Y1[I]) + (Y2[I]*Y2[I]))
        P3 ? IP
        P3 ? K
        P3 ? A
        clock ? tiempo1
        dif := (tiempo1 - tiempo)
        W11 := (REAL32 ROUND dif)
        PS1 ! W11
        PS1 ! (N/(2 (INT)))
        PS1 ! XY
        PS1 ! EP
        PS1 ! K
        PS1 ! A
      SKIP
:
```

HP4FFT2.OCC

```
-- DECLARACION DE LIBRERIAS
--{{{  INCLUDEs
#INCLUDE "hostio.inc" --
#INCLUDE "streamio.inc" --
--}}}
--{{{  USEs
#USE "hostio.lib"
#USE "streamio.lib"
#USE "string.lib"
--}}}
-- DECLARACION DEL NOMBRE DEL PROCESO
PROC H4(CHAN OF SP fs, ts, CHAN OF ANY PS1)
  --{{{  PROC DE ESCRITURA DE  ARCHIVO
  PROC esc.arch(VAL INT len, VAL []BYTE filename,
                INT jdata, []REAL32 data, BYTE brca)
    --{{{  PRINCIPAL
    INT32 id:
    SEQ
      so.open(fs, ts,[filename FROM 0 FOR len), spt.text, spm.output, id, brca)
      IF  --140
        brca = spr.ok
          SKIP
        TRUE
          so.write.string.nl(fs, ts, "EL ARCHIVO NO FUE ABIERTO")
      SEQ i = 1 FOR jdata
        SEQ
          so.fwrite.real32(fs, ts, id, data[i], 0, 8, brca)
          so.fwrite.nl(fs, ts, id, brca)
      so.close(fs, ts, id, brca)
      IF
        brca = spr.ok
          SKIP
        TRUE
          so.write.string.nl(fs, ts, "EL ARCHIVO NO FUE CERRADO")
          ------------------------------------------
    --}}}
  :
  --}}}
-- DECLARACION DE VARIABLES
  INT indx1,indx2, LL,IP,N:
  REAL32 C,SUM,TT1,TT2,SIG2,H,TTT,TT3,TT4:
  BYTE t64 IP:
  [64][64]REAL32 CC2:
  [517]REAL32 CC,X,CC1,A,A1:
  INT MODE,largo,largo1,N,L :
  BOOL erro,error :
  INT IFLAG,KK,K,tiempo:
  INT32 streamid,str2:
  VAL []BYTE sig IS "signal" :
  BYTE result,res2:
  BYTE tes :
  [60]BYTE filename1,filename2:
  INT len,TT,PP:
  BYTE res:
  SEQ
  -- INICIO DEL PROCESO DE CAPATACION DE DATOS POR MEDIO DE
CANALES

    so.write.string(fs,ts,"    vvvvvvvvvv          ")
    PS1 ? H
    so.write.string(fs,ts,"    ffffffffff          ")
    PS1 ? IP
    so.write.string(fs,ts," hhhhhhhhhhhhhhhh       ")
    PS1 ? A
    PS1 ? C
    PS1 ? L
    PS1 ? A1
    so.write.string(fs,ts,"*n*n*c      el tiempo es :  ")
    so.write.string(fs,ts,"*n*n*c    PROCESO  1   ")
    so.write.real32(fs,ts,H,0,0)
    ----so.write.real32(fs,ts,H,0,0)
    so.write.string(fs,ts,"*c*n*c    PROCESO  2   ")
    so.write.real32(fs,ts,C,0,0)
    --{{{  write data file
    --{{{  enter output filename
    so.write.string.nl(fs,ts,"*c*n*c    DAR ARCHIVO 1 ? ")
    so.read.echo.line(fs, ts, len, filename1, res)
    so.write.string.nl(fs,ts,"*c*n*c    DAR ARCHIVO 2 ? ")
    so.read.echo.line(fs,ts,len, filename2,res)
    so.write.nl(fs, ts)
    --}}}
    esc.arch(len,filename1,IP,A,res)
    esc.arch(len,filename2,L,A1,res )
    --{{{  display data if ok
    SEQ
      so.write.string(fs, ts, "utput file ")
      so.write.string(fs, ts, [filename1 FROM 0 FOR len])
      so.write.string(fs, ts, " has ")
      so.write.ini(fs, ts, IP, 0)
      so.write.string(fs, ts, " real values")
      so.write.nl(fs, ts)
      SEQ i = 1 FOR IP
        SEQ
          so.write.real32(fs, ts, A[i], 0, 0)
          so.write.string(fs,ts, "      ")
          so.write.real32(fs,ts,A1[i],0,0)
          so.write.nl(fs, ts)
    --}}}
    --}}}
    so.exit (fs,ts, sps.success)
:
```

HP5FFT1.OCC

```
-- PROCESO LIGADOR ENTRE LOS PROCESADORES TRABAJADORES Y EL
PROCESADOR MAESTRO

PROC HP5(CHAN OF ANY P1,P4,P5)
 [517]REAL32 X:
 INT MODE,IP,N:
 REAL32 EP:
 SEQ
   P1 ? X
   P1 ? N
   P5 ! X
   P5 ! N
   P4 ! X
   P4 ! N
:
```

C.3 PROGRAMA PARA EL CALCULO DE LOS PARAMETROS EN BASE AL METODO DE ESTIMACION ESPECTRAL DE COVARIANCIA MODIFICADA

El siguiente programa consiste de 5 procesos corriendo en tres modulos ; monitor , transmisor-receptor , trabajador; teniendo las siguientes actividades:

1.- Configuración del sistema (CMPS.PGM).

2.- Proceso captador de datos y transmisor de estos al procesador transmisor-receptor (procesador monitor EDCMPS.OCC)

3.- Proceso transmisor-receptor de datos provenientes del procesador monitor y con destino al procesador trabajador 2 y en espera de los resultados del mismo procesador trabajador 2 para transmitirlos al procesador monitor (procesador trabajador 1 TDCMPS.OCC)

4.- Proceso que cálcula los parámetros de la covarianza modificada en base a los datos recibidos del procesador trabajador 1. Sus resultados serán enviados de igual forma hacia el procesador trabajador 1 (procesador trabajador 2 PDCMPS.OCC)

5.- Proceso que recibe datos provenientes del procesador trabajador 1 con los que genera un archivo de salida dado por el usuario (procesador monitor SDCMPS.OCC). El proceso SDCMPS.OCC es igual al proceso SDFPS.OCC por lo que no será expuesto aquí y si se desea consultar se puede hacer uso del SDFPS.OCC

Analizando la configuración del sistema , observamos que existe una diferencia con el programa anterior , dado que se utilizaron dos procesadores trabajadores buscando probar tanto la obtención de los parámetros calculados como la revisión de las comunicaciones del sistema diseñado. Obteniendose resultados satisfactorios en ambos. Además , y al igual que en el programa anterior, se utilizaron canales de tipo anárquico por facilidad de programación sin olvidar, claro esta, las restricciones que se tienen que guardar con respecto a la secuencia de comunicación entre ellos.

Por otro lado el proceso transmisor y el proceso receptor que corren en paralelo en el procesador monitor cumplen la misma función que sus correspondientes en el programa anterior, por lo que, estos no ameritan un mayor análisis apartir de lso comentarios integrados a ellos.

El proceso transmisor-receptor tendrá como única función el transmitir y recibir datos de los procesadores adyacentes (procesador monitor y procesador de cálculo) en base a una secuencia obligada por el tipo de canales usados.

Por último señalaremos al proceso de cálculo del cual podemos decir, que tendrá la función de calcular los parámetros del modelo que representará a la señal estudiada en base a la técnica de covarianza modificada . La teoría de covarianza modificada es expuesta en el capítulo 5.

Tratando de abarcar todo lo anterior se decidio construir el siguiente diagráma a bloques que ilustra la forma en como está configurada la red para este programa.

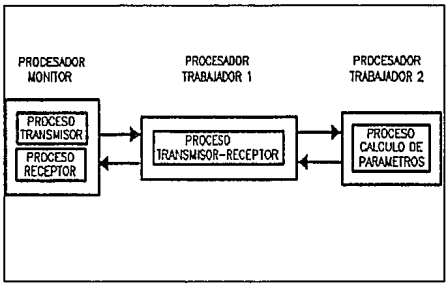


FIGURA C.2 DIAGRAMA A BLOQUES DEL PROCESO

CMPS.PGM

```
#INCLUDE "hostio.inc"
#INCLUDE "streamio.inc"
#INCLUDE "mathvsb.inc"
#INCLUDE "linkaddr.inc"
#USE "EDCMPS.c4h"
#USE "PICMPS.c4h"
#USE "TDCMPS.C8H"
#USE "SDCMPS.c8h"
CHAN OF SP fs,ts:
CHAN OF ANY P1,P2,P3,P4:
PLACED PAR
  PROCESSOR 0 T800
    PLACE ts AT link0.out:
    PLACE fs AT link0.in:
    PLACE P1 AT link1.out:
    PLACE P2 AT link1.in:
    SEQ
      H1(fs,ts,P1)
      H4(fs,ts,P2)
  PROCESSOR 1 T8(0
    PLACE P1 AT link0.in:
    PLACE P2 AT link0.out:
    PLACE P3 AT link1.in:
    PLACE P4 AT link1.out:
    MPS(P1,P2,P3,P4)
  PROCESSOR 2 T800
    PLACE P3 AT link0.out:
    PLACE P4 AT link0.in:
    H2(P4,P3)
```

EDCMPS.OCC

```
--{{{  INCLUDEs
#INCLUDE "hostio.inc" --
#INCLUDE "mathvsb.inc"
#INCLUDE "streamio.inc" --
--}}}
--{{{  USEs
#USE "hostio.lib"
#USE "streamio.lib"
#USE "string.lib"
```

```
#USE "string.lib"
#USE "convert.lib"
--}}}
PROC H(CHAN OF SP fs, ts, CHAN OF ANY PI)
  --{{{  PROC LECTURA DE DATOS DE UN ARCHIVO
  PROC loc.dat.arch(VAL INT len, VAL []BYTE filename,
                    INT j, []REAL32 data, BYTE brea)
    --{{{  principal
    -- toma los datos desde  un archivo de entrada
    SEQ
      CHAN OF KS filekeys:
      CHAN OF KS keyboard IS filekeys:
      PAR
        ------------------------------------------------------
        so.keystream.from.file (fs, ts,
                                keyboard,[filename FROM 0 FOR len], brea)
        ------------------------------------------------------
        REAL32 x:
        INT kchar:
        SEQ
          j := 0 -90
          --{{{  lectura de  una secuencia de  numeros reales
          kchar := 0
          x := 1.0(REAL32)
          WHILE (kchar < > ft.terminated)
            SEQ
              ks.read.char (keyboard, kchar)
              IF
                kchar < 0
                  SKIP
                TRUE   --100
                  ks.read.real32 (keyboard, x, kchar)
              IF
                kchar = ft.terminated
                  SKIP
                TRUE
                  SEQ
                    IF
                      kchar = ft.number.error
                        data[j] := INFINITY
                      TRUE   --110
                        SKIP
                    data[j] := x
                    j := j + 1
          --}}}
          --{{{  consume  el resto del archivo
          IF
            (kchar > = 0) OR (kchar = ft.number.error)
              ks.keystream.sink (keyboard) -- consume  el resto  del archivo
            TRUE
              SKIP  -- el archivo ha terminado o fallo
          --}}}
    IF
      brea < > spr.ok
        SKIP
      TRUE
        SEQ
          so.write.string(fs, ts, " fallo la lectura  ")
          so.write.string.nl(fs, ts,[filename FROM 0 FOR len])
    ------------------------------------------------------
  --}}}
  : --131
  --}}}
  --{{{  DECLARACION DE VARIABLES
  INT index1,index2, LL,,TP,N,H:
  REAL32 C,SUM,TT1,TT2,SIG2,TTT,TT3,TT4:
  REAL32 EP:
  [64][64]REAL32 CC2:
  [640]REAL32 CC,X,CC1,A,,A1:
  INT MODE,largo,largo1,N,L,M :
  BOOL erro,error :
  INT IPLAO,KK,K:
  INT32 streamid,str2:
  VAL []BYTE rig IS "signal" :
  BYTE result,rest2:
  BYTE trea :
  [60]BYTE filename: --170
  INT len,TT.PP,nconn,nconno:
  BYTE rea:
  --}}}
  SEQ
    --{{{  lectura de datos de un archivo
    --{{{  captura  del nombre del archivo
    so.write.string.nl(fs, ts,
     " nombre del archivo de entrada ? ")
    so.read.echo.line(fs, ts, len, filename, rea)
    so.write.nl(fs, ts)  --180
    --}}}
    loc.dat.arch(len,filename,N,X,rea)
    --{{{  despliegue de datos
    IF
      rea = spr.ok
        SEQ
          so.write.string(fs, ts, " archivo de entrada  ")
          so.write.string(fs, ts, [filename FROM 0 FOR len])
          so.write.string(fs, ts, " tiene  ")
          so.write.int(fs, ts, N, 0)
          so.write.string(fs, ts, " valores reales  ") --190
          so.write.nl(fs, ts)
          SEQ i = 0 FOR N
            SEQ
              so.write.string(fs, ts, "[")
              so.write.int(fs, ts, i, 0)
              so.write.string(fs, ts, "]= ")
              so.write.real32(fs, ts, X[i], 0, 0)
              so.write.nl(fs, ts)
      TRUE
        SKIP
    --}}}
    --}}}
    --{{{ CAPTURA DE DATOS REQUERIDOS PARA EL CALCULO
    so.write.string(fs,ts,"\n DAR EL ORDEN DEL MODELO AR: ")
    so.read.echo.int(fs,ts,IP,erro)
    so.write.string(fs,ts,"\n DAR EL METODO A USAR :  ")
    so.read.echo.int(fs,ts,MODE,erro)
    so.write.string(fs,ts,"\n DAR EL NUMERO DE CHOLESKY :  ")
    so.read.echo.real32(fs,ts,EP,error)
    --}}}
    --{{{ TRANSMISION DE INFORMACION

    PI ! X
    PI ! N
    PI ! IP
    PI ! MODE
    PI ! EP
    --}}}
```

TDCMPS.OCC

```
PROC HP3(CHAN OF ANY P1,P2,P3.PN)
  {{{  DECLARACION DE  VARIABLES
```

```
[640]REAL32 X,A:
INT MODE,IP,N:
REAL32 EP:
}}}
SEQ
  {{{  RECEPCION DE  DATOS DE PROCESO MONITOR
  P1 ? X
  P1 ? N
  P1 ? IP
  P1 ? MODE
  P1 ? EP
  }}}
  {{{  TRANSMISION DE DATOS  HACIA EL PROCESADOR ESCLAVO
  P4 ! X
  P4 ! N
  P4 ! IP
  P4 ! MODE
  P4 ! EP
  }}}
  {{{  RECEPCION DE DATOS  DEL PROCESADOR ESCLAVO
  P3 ? IP
  P3 ? A
  }}}
  {{{  TRANSMISION DE DATOS HACIA EL PROCESADOR MONITOR
  P2 ! IP
  P2 ! A
  }}}
:
```

PDCMPS.OCC

```
#INCLUDE "mathvals.inc"--
PROC H2(CHAN OF ANY P1,PS1)
  --{{{  CHOLESKY
  PROC CHOLESKY(VAL[][]REAL32 CC2,[]REAL32 A, VAL INT IP,REAL32 EP,
INT IFLAG )
    --{{{  variables
    [64][64]REAL32 L:
    [640]REAL32 A1,d,y,x:
    INT LL,i,col,col1,k,pk:
    --}}}
    SEQ
      LL:=0

      --{{{  initialization
      SEQ i=0 FOR IP
        y[i]:=A[i]
      --}}}
      IF
        LL = 0
          SEQ
            --{{{  triangularization
            d[0]:=CC2[0][0]
            SEQ i=1 FOR (IP-1)
              L[i][0]:=CC2[i][0]/d[0]
            d[1]:=CC2[1][1]-(d[0]*(L[1][0]*L[1][0]))
            SEQ col=1 FOR IP-2
              SEQ
                --{{{  computation of L
                SEQ i=(col+1) FOR ((IP-col)-1)
                  SEQ
                    L[i][col]:=CC2[i][col]
                    SEQ k=0 FOR col
                      L[i][col]:=L[i][col]-(L[i][k]*(d[k]*L[col][k]))
                    L[i][col]:=L[i][col]/d[col]
                SEQ i=0 FOR IP
                  A1[i]:= L[i][col]
                --}}}
                --{{{ CALCULO DE  d
                col1:=col+1
                d[col1]:=CC2[col1][col1]
                SEQ k=0 FOR col1
                  d[col1]:=d[col1]-(d[k]*(L[col1][k]*L[col1][k]))
                --}}}
            --}}}

            --{{{ RESUELVE Ly=b
            SEQ col=1 FOR (IP-1)
              SEQ i=col FOR (IP-col)
                y[i]:=y[i]-(L[i][col-1]*y[col-1])
            --}}}

            --{{{RESUELVE DIAGONAL
            SEQ i=0 FOR IP
              x[i]:=y[i]/d[i]
            --}}}
            --{{{ SOLUCION FINAL          SEQ k=0 FOR (IP-1)
              SEQ
                pk:=(IP-k)-1
                SEQ i=0 FOR pk
                  x[i]:=x[i]-(L[pk][i]*x[pk])
            SEQ i=0 FOR IP
              A[i+1]:= -x[i]
            --}}}
        LL <> 0
          SKIP
:
--}}}
INT index1,index2, LL,IP,N,H:
REAL32 C,SUM,TT1,TT2,SIG2,TTT,TT3,TT4:
REAL32 EP,w1:
[64][64]REAL32 CC2:
[640]REAL32 CC,X,CC1,A,A1:
BYTE res:
INT MODE,largo,largo1,N,L,ti1,ti2,ti3 :
BOOL erro,error :
TIMER j6ff:
INT IFLAG,KK,K:
INT  dif,tiempo,tiempo1:
INT nonon,nonono:  ----70
INT32 streamid,str2:
BYTE result,res2:
BYTE tres :
INT len,TT,PP,IP2:
PRI PAR
  SEQ
    P1 ? X
    P1 ? N
    P1 ? IP
    P1 ? MODE
    P1 ? EP

    j6ff ? ti1
    L:=1
    largo1:=largo
    X[N]:=X[0]
    H:=N-IP
    IFLAG:=0  --90
    TTT:=REAL32 ROUND (2 (INT) *(N-IP))
    SEQ index1 = 1 FOR IP
      SEQ
        SEQ index2 = 1 FOR IP
          SEQ
```

```
    CC[L]:=0.0 (REAL32)
    CC[L]:=0.0 (REAL32)
    SEQ I=IP FOR N+1
      SEQ
        CC[L]:=CC[L]+X[I+index2]*X[I+index1]  -100
    IP
      MODE = 1
        SEQ
          SEQ I=1 FOR N
            SEQ
              CC[L]:=CC[L]+X[I+index2]*X[I+index1]
    CC[L]:=(CC[L]+CC1[L])/TTT
      L:= L + 1
L:=1
SEQ L2=0 FOR IP                                  -110
 SEQ L3=0 FOR IP
  SEQ
   IP
     L<=36
      SEQ
        CC2[L2][L3]:=CC[L]
        L:=L+1
     L>36
       SKIP
SEQ J=1 FOR IP                                   -120
 SEQ
  A[J]:= 0.0 (REAL32)
  A1[J]:=0.0 (REAL32)
  SEQ I=IP FOR I+1
    A[J]:=A[J]-X[I-1]*X[I]
  IP
    MODE = 1
     SEQ I=1 FOR N
       SEQ
         A1[J]:=A1[J]-X[I+J]*X[I]
  A[J]:=0.0 (REAL32)-((A[J]+A1[J])/TTT)
SEQ J=0 FOR IP
 SEQ
  A[J]:=A[J+1]
IP:=6
CHOLESKY(CC2,A,IP,EP,IFLAG)
IF
 IFLAG <> (-1)
  SEQ
   SUM  := 0.0 (REAL32)
   SEQ KK=1 FOR (IP+1)
     SEQ
      SEQ
       C:=0.0 (REAL32)
       K := KK-1
       TT:=1 (INT)
       LL:= (IP + TT)
       SEQ I=LL FOR N-IP
         C:=(C  +  (X[I]*X[I-K]))
       IP
         MODE = 1
          SEQ
            SEQ I=1 FOR N-IP
              SEQ
                C:=(C + (X[I]*X[I+K]))
       IP
         K=0
          SUM:=( SUM + C )
         K<>0
           SUM:=( SUM + (C*A[K]) )
   IP
    MODE = 1
      SEQ
       TT1:=REAL32 ROUND N
       TT2:=REAL32 ROUND IP
       SIG2:=SUM/(2.0 (REAL32)*(TT1-TT2))
      MODE <> 1
       SEQ
        TT3:=REAL32 ROUND N
        TT4:=REAL32 ROUND IP
        SIG2 :=SUM/(TT3-TT4)
  IFLAG = (-1)
   SKIP
j6if ? t2
t3 := t2 MINUS t1
w1 := REAL32 ROUND t3
IP2:=(( IP*IP)+1)
P81 ! IP2
P81 ! w1
P81 ! A
SKIP
:
```

C.4 PROGRAMA PARA EL CALCULO DE LA DENSIDAD DE POTENCIA ESPECTRAL EN BASE A LA COVARIANCIA MODIFICADA PARA UN SOLO MODULO TRABAJADOR

Este programa se encuentra dividido en tres procesos, uno de transmisión , uno de recepción y un último de procesamiento. El programa se cargará en dos procesadores , uno monitor y un trabajador.

1.- Configuración de la red (CMFPS.PGM)

2.- Proceso transmisor de datos (procesador monitor PCCMFPS.OCC)

3.- Proceso diseñado para calcular la densidad de potencia espectral (procesador trabajador 1 PCMFPS.OCC)

4.- Proceso receptor de datos provenientes del procesador trabajador 1 (procesador trabajador 2 PSCMFPS.OCC)

En el siguiente programa se configuro la red de forma similar al primer progama de este apéndice , por lo que solo se utilizo dos modulos de procesamiento , buscando probar el algorítmo que calcula la densidad de potencia espectral, en forma secuencial . Como se ha señalado en ocasiones anteriores el tipo de canales que se utilizo fué nuevamente anárquico por su facilidad de manejo y la experiencia adquirida en ocasiones anteriores.

Los procesos de entrada y salida de datos que son ejecuados en el procesador monitor siguen la misma línea de diseño que sus predecesores en los programas anteriores , por lo que se hará mayor referencia de ellos.

El proceso de cálculo de la densidad de potencia espectral está conformado de los siguientes procesos:

1.- Captura de datos

2.- Cálculo de los "TWIDLE FACTORS"

3.- Cálculo de los parámetros

3.1 .- Cálculo del renglón principal de la matriz de covarianza

3.2 .- Ordenamiento de cada uno de los elementos de la matriz de covarianza

3.3 .- Cálculo de la matriz del "lado derecho "

3.4 .- Resolución del sistema de matrices utilizando la técnica de

CHOLESKY (obtención de los parámetros)

3.5 .- Cálculo de la varianza de ruido blanco

4.- Cálculo de la FFT

4.1 .- Generación del archivo para el cálculo de la FFT en base a los parámetros obtenidos en el punto anterior

4.2 .- Reordenamiento del archivo anterior de forma "BIT REVERSED"

4.3 .- Calculo de las mariposas de la FFT.

5.- Cálculo de la Densidad de Potencia Espectral

5.1 .- Cálculo del Módulo Cuadrado de cada uno de los elementos de la FFT

5.2 .- Cálculo del cociente de la varianza de ruido blanco y el Módulo Cuadrado de cada elemento de la FFT.

6.- Transmisión de datos

Estos procesos corren de manera secuencial , puesto que el desarrollo del algoritmo así lo indica.

Por último basta hacer mención del diagrama a bloques que muestra la forma en que esta diseñada la red.

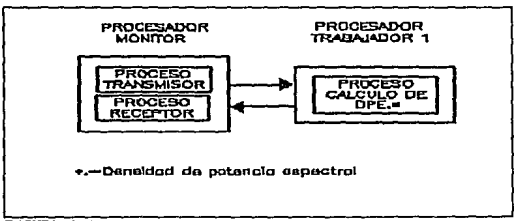


FIGURA C.3

CMFPS.PGM

```
#INCLUDE "hostio.inc"
#INCLUDE "streamio.inc"
#INCLUDE "mathvals.inc"
#INCLUDE "linkaddr.inc"
#USE "pcmfps.clb"
#USE "pcmfps.clb"
#USE "pcmfps.clb"
CHAN OF SP fs,ts:
CHAN OF ANY P1,P2:
PLACED PAR
  PROCESSOR 0 T800
    PLACE ts AT link0.out:
    PLACE fs AT link0.in:
    PLACE P1 AT link1.out:
    PLACE P2 AT link1.in:
    SEQ
      H1(fs,ts,P1)
      H4(fs,ts,P2)
  PROCESSOR 1 T800
    PLACE P1 AT link0.in:
    PLACE P2 AT link0.out:
    H2(P1,P2)
```

PCCMFPS.OCC

```
--{{{  INCLUDEs
#INCLUDE "hostio.inc" --
#INCLUDE "mathvals.inc"
#INCLUDE "streamio.inc" --
--}}}
--{{{  USEs
#USE "hostio.lib"
#USE "streamio.lib"
#USE "string.lib"
#USE "convert.lib"
--}}}
PROC H1(CHAN OF SP fs, ts, CHAN OF ANY P1)
  --{{{  PROC LECTURA. ARCHIVO
  PROC lec.arch(VAL INT len, VAL []BYTE filename,
                INT j, []REAL32 data, BYTE bres)
    --{{{  principal
    -- TOMA DATOS DE ENTRADA DESDE UN ARCHIVO
    SEQ
      CHAN OF K5 fielkeys:
      CHAN OF K5 keyboard IS fielkeys:
      PAR
        ------------------------------------------------
        so.keystream.from.file (fs, ts,
                                keyboard,[filename FROM 0 FOR len], bres)
        ------------------------------------------------
        REAL32 x:
        INT kchar:
        SEQ
          j := 0  --90
          --{{{  lee la secuencia de numeros reales
          kchar := 0
          x := 1.0(REAL32)
          WHILE (kchar <> ft.terminated)
            SEQ
              ka.read.char (keyboard, kchar)
              IF
                kchar < 0
                  SKIP
                TRUE  --100
                  ka.read.real32 (keyboard, x, kchar)
              IF
                kchar = ft.terminated
                  SKIP
                TRUE
                  SEQ
                    IF
                      kchar = ft.number.error
                        data[j] := INFINITO
                      TRUE  --110
                        SKIP
                    data[j] := x
                    j := j + 1
          --}}}
          --{{{  CONSUME EL RESTO DEL ARCHIVO
          IF
            (kchar >= 0) OR (kchar = ft.number.error)
              ka.keystream.sink (keyboard)
            TRUE
              SKIP -- EL ARCHIVO A TERMINADO O FALLO
          --}}}
        IF
          bres <> spr.ok
            SKIP
          TRUE
            SEQ
              so.write.string(fs, ts, "Fallo la lectura")
              so.write.string.nl(fs, ts,[filename FROM 0 FOR len])
        ------------------------------------------------
    --}}}
  : --13)
  --}}}
  INT index1,index2,  LL,.IP,N,H:
  REAL32 C,SUM,TT1,TT2,SIG2,TTT,TT3,TT4:
  REAL32 EP:
  [640]REAL32 X,CC),A,A1:
  INT MODE,largo.largo1,L,M :
  BOOL ero,error :
  INT IPLAG,KK,K:
  INT32 streamid,str2:
  VAL []BYTE sig IS "signal" :
  BYTE result,res2:
  BYTE tres :
  [60]BYTE filename: --170
  INT len,TT,PP,nonon,nonono:
  BYTE res:
  SEQ
    --{{{  LECTURA DE DATOS
    --{{{  LECTURA DE DATOS DEL ARCHIVO
    so.write.string.nl(fs, ts,
     "Archivo de entrada ? ")
    so.read.echo.line(fs, ts, len, filename, res)
    so.write.nl(fs, ts)  --180
    --}}}
    lec.arch(len,filename,N,X,res)
    --{{{  DESPLIEGUE DE DATOS
    IF
      res = spr.ok
        SEQ
          so.write.string(fs, ts, "archivo de entrada ")
          so.write.string(fs, ts, [filename FROM 0 FOR len])
          so.write.string(fs, ts, " tiene ")
          so.write.int(fs, ts, N, 0)
          so.write.string(fs, ts, "  valores reales  ") --190
          so.write.nl(fs, ts)
          SEQ i = 0 FOR N
```

```
      SEQ
        so.write.string(fs, ts, "(")
        so.write.int(fs, ts, i, 0)
        so.write.string(fs, ts, ")= ")
        so.write.real32(fs, ts, X[i], 0, 0)
        so.write.nl(fs, ts)
    TRUE
      SKIP
  -}}}
  -}}}
  so.write.string(fs,ts,"\n DAR EL ORDEN DEL MODELO AR: ")
  so.read.echo.int(fs,ts,IP,erro)
  so.write.string(fs,ts,"\n DAR EL METODO A USAR : ")
  so.read.echo.int(fs,ts,MODE,erro)
  so.write.string(fs,ts,"\n\n DAR EL NUMERO DE CHOLESKY :  ")
  so.read.echo.real32(fs,ts,EP,erro)
  so.write.string(fs,ts,"\n\n DAR EL NUMERO MARIPOSA  ")
  so.read.echo.int(fs,ts,K,erro)
  so.write.string(fs,ts,"              ")
  P1 ? X
  P1 ? N
  P1 ? IP
  P1 ? MODE
  P1 ? K
  P1 ? EP
:
```

PCMFPS.OCC

```
#INCLUDE "mathvals.inc"
#USE "maglmath.lib"
PROC H2(CHAN OF ANY P1,PS1)
  -{{{  CHOLESKY
  PROC CHOLESKY(VAL [][]REAL32 CC2,[]REAL32 A, VAL INT IP,REAL32 EP,
INT IFLAG )
    -{{{  variables
    [64][64]REAL32 L:
    [640]REAL32 d,y,x:
    INT i,col,col1,k,pk:
    -}}}
    SEQ
    --EL SOUIENTE PROCESO DESARROLLA UNA TECNICA PARA LA
    --SOLUCION DE ECUACIONES CON VARIAS INCOGNITAS CON AYUDA
    -- DE MATRICES EL PROCESO TRABAJARA DE FORMA EFICIENTE EN LA
    --SOLUCION DE LA ECUACION Ly =b TENIENDO COMO DATOS
    --CONOCIDOS A * L^ Y *b*
      -{{{  INICIALIZACION
      SEQ i=0 FOR IP
        y[i]:=A[i]
      -}}}
      -{{{  TRIANGULARIZACION
      d[0]:=CC2[0][0]
      SEQ i=1 FOR (IP-1)
        L[i][0]:=CC2[i][0]/d[0]
      d[1]:=CC2[1][1]-(d[0]*L[1][0]*L[1][0])
      SEQ col=1 FOR IP-2
        SEQ
          -{{{  CALCULO DE  L
          SEQ i=(col+1) FOR ((IP-col)-1)
            SEQ
              L[i][col]:=CC2[i][col]
              SEQ k=0 FOR col
                L[i][col]:=L[i][col]-(L[i][k]*(d[k]*L[col][k]))
              L[i][col]:=L[i][col]/d[col]
          -}}}
          -{{{  CALCULO DE  d
          col1:=col+1
          d[col1]:=CC2[col1][col1]
          SEQ k=0 FOR col
            d[col1]:=d[col1]-(d[k]*(L[col1][k]*L[col1][k]))
          -}}}
    -}}}
    -{{{  SOLUCION Ly=b
    SEQ col=1 FOR (IP-1)
      SEQ i=col FOR (IP-col)
        y[i]:=y[i]-(L[i][col-1]*y[col-1])
    -}}}
    --{{{  RESUELVE DIAGONAL
    SEQ i=0 FOR IP
      x[i]:=y[i]/d[i]
    -}}}
    --{{{  SOLUCION FINAL
    SEQ k=0 FOR (IP-1)
      SEQ
        pk:=(IP-k)-1
        SEQ i=0 FOR pk
          x[i]:=x[i]-(L[pk][i]*x[pk])
    SEQ i=0 FOR IP
      A[i+1]:=-x[i]
    -}}}
:
-}}}
INT IP2,index1,index2, il,iil,LL,IP,N,H:
REAL32 C,SUM,TT1,TT2,SIG2,TTT,TT3,TT4:
REAL32 EP,W1,W2,W3, PI, ARG,SAVE1I,SAVE3,SAVE2:          --60
[10][640]REAL32 CC2,W11:
[640]REAL32 XY,CC,X,CC1,A,A1:
INT MODE,largo,largo1,M,L,M.J,NUM,I,K,LDFT,NDFT,NP,NQ:
INT IFLAG,KK,k:
INT32 streamid,st2:
INT km,TT,PP,il,il2,il3,il4,il5,il6,il7,il8:
TIMER jdpf:
PRI PAR
  SEQ
    -{{{  cov
    -{{{  RECEPCION DE DATOS
    P1 ? X
    P1 ? N
    P1 ? IP
    P1 ? MODE
    P1 ? M
    P1 ? EP
    -}}}
    jdpf ? il
    L:=1
    largo1:=largo
    X[N]:=X[0]
    H:=N-IP
    IFLAG:=0  --45
    TTT:=REAL32 ROUND (2 (INT) *(N-IP))
    --SEQ index1 = 1 FOR IP
    --  SEQ
    --{{{  CALCULO DEL RENGLON INICIAL DE LA MATRIZ DE
    --COVARIANCIA MODIFICADA
    SEQ index2 = 1 FOR IP
      SEQ
        CC1[L]:=0.0 (REAL32)
        CC[L]:=0.0 (REAL32)
        SEQ i=(IP+1) FOR (N-IP)
          SEQ
            CC1[L]:=CC1[L]+(X[i-index2]*X[i-1])  --95
          IF
            MODE = 1
```

```
      SEQ
        SEQ I=1 FOR H
          SEQ
            CC[L]:=CC[L]+(X[I+index2]*X[I+J])
    CC[L]:=0.0 (REAL32) +((CC[L]+CC1[L])/(TT))
    L:= L + 1
  -}}}
  --{{{ DESIGNACION DE LOS VALORES CALCULADOS A TODA LA
MATRIZ
    L:= 0
    SEQ index2 =1 FOR IP
      SEQ
        IP2:=IP-L
        CC[(index2+(L*(IP-1)))]:=CC[index2]
        SEQ index1=1 FOR (IP2-1)
          SEQ
            CC[(index2 + (index1*(IP+1)))]:= CC[index2]
            IF
              index2 <> 1
                SEQ
                  CC[(((L*(IP-1))+index2)+(index1*(IP+1)))]:=     CC[(index2
+(index1*(IP+1)))]
              index2 = 1
                SEQ
                  K:=0
        L:=L+1
  -}}}
  --{{{ CONVERSION DE LA MATRIZ DE UNA FORMA LINEAL A UNA
FORMA CUADRADA
    L:=1
    IP2:= IP*IP
    SEQ L2=0 FOR IP                                    -105
      SEQ L3=0 FOR IP
        SEQ
          IF
            L <= IP2
              SEQ
                CC2[L2][L3]:=CC[L]
                L:=L+1
            L > IP2
              SKIP
  -}}}
  --{{{ CALCULO DE LA MATRIZ DE TERMINOS INDEPENDIENTES
  SEQ J=1 FOR IP                                       -115
    SEQ
      A[J]:= 0.0 (REAL32)
      A1[J]:=0.0 (REAL32)
      SEQ I=IP FOR N
        A[I]:=A[J]-(X[I-J]*X[I])
      IF
        MODE = 1
          SEQ I=1 FOR H
            SEQ
              A1[I]:=A1[J]-(X[I+J]*X[I])
      A[J]:=0.0 (REAL32)-((A[J]+A1[J])/TT)
  SEQ J=0 FOR IP
    SEQ
      A[J]:=A[J+1]
  -}}}
  CHOLESKY(CC2,A,IP,EP,IFLAG)
  --{{{ CALCULO DE SIGMA CUADRADA
  IFLAG := 0 (INT)
  IF
    IFLAG <> (-1)
      SEQ
        SUM := 0.0 (REAL32)
        SEQ KK=1 FOR (IP+1)
          SEQ
            C:=0.0 (REAL32)
            K:= KK-1
            TT:=1 (INT)
            LL:= (IP + TT)
            SEQ I=LL FOR N-IP
              C:=(C + (X[I]*X[I-K]))
          IF
            MODE = 1
              SEQ
                SEQ I=1 FOR N-IP
                  SEQ
                    C:=(C + (X[I]*X[I+K]))
            IF
              K=0
                SUM:=( SUM + C )
              K<>0
                SUM:=( SUM + (C*A[K]) )
        IF
          MODE = 1
            SEQ
              TT1:=REAL32 ROUND N
              TT2:=REAL32 ROUND IP
              SIG2:=SUM/(2.0 (REAL32)*(TT1-TT2))
          MODE <> 1
            SEQ
              TT3:=REAL32 ROUND N
              TT4:=REAL32 ROUND IP
              SIG2 :=SUM/(TT3-TT4)
    IFLAG = (-1)
  -}}}
    SKIP
  j&p[ ? u2

  -}}}
  --{{{ FFT
  TT4 := REAL32 ROUND M
  N := 1
  --{{{ CALCULO DE ELEMENTOS NECESARIOS PARA LA MARIPOSA
  SEQ I=1 FOR M
    SEQ
      N := (N *( 2 (INT)))
  -}}}
  --SEQ I=1 FOR 9
  --  SEQ
  --    SEQ J=1  FOR N
  --      W1[I][J]:= 0.0 (REAL32)
  --{{{ DESIGNACION DE ELEMENTOS PARA EL CALCULO DE LA  FFT
  A1[1]:=1.0 (REAL32)
  SEQ I=1  FOR IP
    SEQ
      A1[I+1]:= A[I]
  IP := (IP + 1 (INT))
  SEQ I= IP  FOR N-IP
    SEQ
      A1[I+1]:= 0.0 (REAL32)
  CC1[1]:=A1[1]
  -}}}
  --{{{ DESIGNACION DEL ARCHIVO DE ELEMENTOS EN FORMA DE BIT
REVERSE
    SEQ I=2 FOR N-1
      SEQ
        NUM:= I-1
        J:=0
        L:=N
        SEQ K=1 FOR M
```

```
    SEQ
      L:= (L/2)
      IF
        NUM > = L
          SEQ
            III:=1
            SEQ II=1 FOR K-1
              SEQ
                ID := (III * (2 (INT)))
              J:= J + (III)
              NUM:= NUM-L
        NUM < L
          SKIP
    CC1[I]:= A1[J + 1] -- SIGUE EL CALCULO DE FFT LINEA 188
  -}}}
  SEQ I = 1 FOR N
    CC[I]:= 0.0 (REAL32)
  PI:= 3.141592654 (REAL32)
  LDFT := 1
  NDFT := N
  --{{{  CALCULO DE LOS TWIDLE FACTORS
  SEQ K = 1 FOR M
    SEQ
      LDFT := (2*LDFT)
      NDFT := (NDFT/2)
      SEQ I = 1 FOR NDFT
        SEQ
          LL:=(LDFT/2)
          SEQ J = 1 FOR LL
            SEQ
              ARG    := (0.0(REAL32)-((2.0(REAL32)*PI)*((REAL32 ROUND
J)-1.0(REAL32)))/(REAL32 ROUND LDFT)))
              NP := J + (LDFT*(I-1))
              W1[K][NP] := COS(ARG)
              CC2[K][NP] := SIN(ARG)
  -}}}
  jkpf ? t4
  LDFT := 1
  NDFT := N
  --{{{ CALCULO DE LAS MARIPOSAS DE LA FFT
  SEQ K = 1 FOR M
    SEQ
      LDFT := (2*LDFT)
      NDFT := (NDFT/2)
      SEQ I = 1 FOR NDFT
        SEQ
          LL:= LDFT/2
          SEQ J = 1 FOR LL
            SEQ
              --ARG: =    (0.0(REAL32)-((2.0(REAL32)*PI)*((REAL32 ROUND
J)-1.0(REAL32)))/(REAL32 ROUND LDFT)))
              NP:= J + (LDFT*(I-1))
              NQ:= (NP + (LDFT/2))
              --W1 := COS(ARG)
              --W2 := SIN(ARG)
                                                    S A V E 1 : =
CC1[NP] + ((W1[K][NP]*CC1[NQ])-(CC2[K][NP]*CC[NQ]))
                                                    S A V E 2 : =     C C [ N P ]
+ ((CC2[K][NP]*CC1[NQ] + (W1[K][NP]*CC[NQ])
                                                    S A V E 3 : =
CC1[NP] + ((CC2[K][NP]*CC[NQ])-(W1[K][NP]*CC1[NQ]))
              CC[NQ]: =CC[NP]-((CC2[K][NP]*CC1[NQ]) + (W1[K][NP]*CC[NQ]))
              CC1[NP]:= SAVE1
              CC[NP] = SAVE2
              CC1[NQ]: = SAVE3
  -}}}
  jkpf ? t5
```

```
  --{{{ CALCULO DE LA DENSIDAD DE POTENCIA ESPECTRAL
  SEQ I = 1 FOR N
    XY[I] := (0.0(REAL32)-(SIG2/(((CC1[I]*CC1[I]) + (CC[I]*CC[I]))))
  -}}}
  jkpf ? t7
  --{{{ CALCULO DE TIEMPOS
  t3 := (t2 MINUS t1)
  t6 := (t5 MINUS t4)
  t8 := (t7 MINUS t5)

  W1 := REAL32 ROUND (t3)
  W2 := REAL32 ROUND (t6)
  W3 := REAL32 ROUND (t8)
  -}}}
  --{{{ TRANSMISION DE DATOS
  PS1 ! N
  PS1 ! W1
  PS1 ! W2
  PS1 ! W3
  PS1 ! XY
  -}}}
  -}}}
  SKIP
:
```

PSCMFPS.OCC

```
--{{{  INCLUDEs
#INCLUDE "hostio.inc" --
#INCLUDE "streamio.inc" --
--}}}
--{{{  USEs
#USE "hostio.lib"
#USE "streamio.lib"
#USE "string.lib"
--}}}
PROC P4(CHAN OF SP fs, ts, CHAN OF ANY PS1)
  --{{{PROC ESCRITURA DE DATOS EN UN ARCHIVO
  PROC esc.dat.arch(VAL INT len, VAL []BYTE filename,
                    INT idata, []REAL32 data, BYTE brea)
    --{{{ PRINCIPAL
    INT32 id:
    SEQ
      so.open(fs, ts,1[filename FROM 0 FOR len], spt.text, spm.output, id, brea)
      IF  --140
        brea = spr.ok
          SKIP
        TRUE
          so.write.string.nl(fs, ts, "archivo no fue abierto  ")
      SEQ i = 1 FOR idata
        SEQ
          so.fwrite.real32(fs, ts, id, data[i], 0, 8, brea)
          so.fwrite.nl(fs, ts, id, brea)
      so.close(fs, ts, id, brea)
      IF
        brea = spr.ok
          SKIP
        TRUE
          so.write.string.nl(fs, ts, "archivo no fue cerrado  ")
          ------------------------------------
    -}}}
  :
  -}}}
  INT index1,index2, LL,IP,N:
```

```
REAL32 C,SUM,TT1,TT2,SIG2,H,TTT,TT3,TT4:
REAL64 PP:
[640]REAL32 CC,A,A1:
INT MODE,largo,largo1,N,L :
BOOL ctrl,error :
INT IFLAG,KK,K,tiempo:
INT32 streamid,str2:
VAL []BYTE sig IS "signal" :
BYTE result,res2:
BYTE tres :
[60]BYTE filename:
INT len,TT,PP:
BYTE res:
SEQ
  PS1 ? IP
  PS1 ? H
  PS1 ? TT3
  PS1 ? TT4
  PS1 ? A
  so.write.string(fs,ts,"*c*n*c       el tiempo es :  ")
  so.write.string(fs,ts,"*c*n*c*n cov  modif      ")
  so.write.real32(fs,ts,H,0,0)
  so.write.string(fs,ts,"*c*n*c*n  fourier       ")
  so.write.real32(fs,ts,TT3,0,0)
  so.write.string(fs,ts,"*c*n*c*n densidad de potencia espectral    ")
  so.write.real32(fs,ts,TT4,0,0)
  so.write.string(fs,ts,"     *n*c*n*c     ")
  --{{{ ESCRITURA DE DATOS EN UN ARCHIVO
  --{{{ ENTRADA DE ARCHIVO DE SALIDA
  so.write.string.nl(fs, ts, "archivo de salida ? ")
  so.read.echo.line(fs, ts, len, filename, res)
  so.write.nl(fs, ts)
  --}}}
  cac.dat.arch(len,filename,IP,A,res)
  --{{{ DESPLIEGUE DE DATOS
  SEQ
    so.write.string([fs, ts, "  archivo de salida   ")
    so.write.string([fs, ts, [filename FROM 0 FOR len])
    so.write.string([fs, ts, " tiene   ")
    so.write.int([fs, ts, IP, 0)
    so.write.string([fs, ts, " valores reales   ")
    so.write.nl(fs, ts)
    SEQ i = 1 FOR IP
      SEQ
        so.write.real32(fs, ts, A[i], 0, 0)
        so.write.nl(fs, ts)
  --}}}
  --}}}
  so.exit (fs,ts, sps.success)
:
```

C.5 PROGRAMA PARA EL CALCULO DE LA DENSISDAD DE POTENCIA ESPECTRAL DE FORMA PARALELA PARA EL MISMO NUMERO DE DATOS EN CADA MODULO

El siguiente programa propone el mismo algoritmo dado en el programa anterior pero con la ventaja de que este será copiado en dos módulos de procesamiento , por lo que podrá procesar el doble de datos en casi el mismo tiempo que para un proceso secuencial (Capítulo 6)

Al igual que los anteriores programas ,este consta con dos procesos en el módulo monitor y una serie de procesos distribuidos en los procesadores trabajadores.

1.- Configuración del sistema (CMPF12.PGM)

2.- Proceso de captúra de datos (procesador monitor eds4.occ). Este proceso es igual al proceso PCCMFPS.OCC por lo que no se mostrará en el presente apéndice

3.- Proceso que realiza el cálculo de la densidad de potencia espectral para el primer tramo de los datos. Además , este proceso se encarga de transmitir y recibir la información que requiere el procesador trabajador 2 (procesador trabajador 1 PCMFPDO.OCC)

4.- Proceso que realiza también el cálculo de la densidad de potencia espectral pero desde el procesador trabajador 2 (PCMFPUN.OCC)

5.- Proceso que recibe la información generada por los módulos y la almacena en una serie de archivos dados por el usuario (procesador monitor SDS4.OCC)

Como podemos observar los procesos de lectura y escritura de archivos que se encuentran en el procesador monitor son los mismos con los que hemos trabajado en los programas anteriores.

Los procesos de cálculo que se alojan en los módulos trabajadores son casi iguales a el proceso utilizado en el programa anterior . Teniendo como principal diferencia el cálculo de los "TWIDLE FACTORS" , pues , este cálculo solo se llevará acabo desde el procesador trabajador 1 y será transmitido al procesador trabajador 2 . Con esto se logrará ganar tiempo en el procesamiento paralelo de la señal. La caracterisitca más importante de este programa , es que, se puede procesar un segmento del doble del tamaño del que se procesa en el programa anterior que tiene un procesamiento completamente secuencial.

Cabe señalar que, los procesos paralelos mostrados aquí guardan el paralelismo sobre el segmento de datos que se está procesando.

Sin embargo, se pudiera sugerir que se dividiera el algoritmo en partes , con lo que los módulos lo procesarían en forma paralela . Esta opción no es posible , puesto que , el algoritmo es construido de forma secuencial y resulta poco eficáz para el procesamiento de señales.

Esta aplicación presenta la siguiente configuración en diagráma a bloques con sus respecetivos procesos

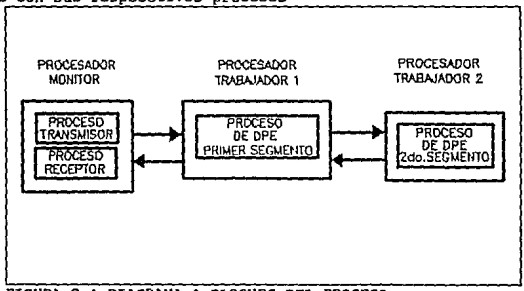


FIGURA C.4 DIAGRAMA A BLOQUES DEL PROCESO

CMFP12.PGM

```
#INCLUDE "hostio.inc"
#INCLUDE "streamio.inc"
#INCLUDE "mathvals.inc"
#INCLUDE "linkaddr.inc"
#USE "cda4.clib"
#USE "pcmfpm.clib"
#USE "pcmfpdo.clib"
#USE "cda4.clib"
CHAN OF SP fs,ts
CHAN OF ANY P1,P2,P3,P4:
PLACED PAR
  PROCESSOR 0 T800
    PLACE ts AT link0.out:
    PLACE fs AT link0.in:
    PLACE P1 AT link1.out:
    PLACE P2 AT link1.in:
    PAR
      Hi(fs,ts,P1)
      Hi(fs,ts,P2)
  PROCESSOR 1 T800
    PLACE P1 AT link0.in:
    PLACE P2 AT link0.out:
    PLACE P4 AT link1.in:
    PLACE P3 AT link1.out:
    CHAN OF ANY P5:
    H2(P1,P3,P4,P5)
  PROCESSOR 2 T800
    PLACE P3 AT link0.in:
    PLACE P4 AT link0.out:
    H2(P3,P4)
```

PCMFPDO.OCC

```
#INCLUDE "mathvals.inc"
#USE "arg-lmath.lib"
--AQUI SE PRESENTARA EL CALCULO DE LA DENSIDAD DE POTENCIA
ESPECTRAL
--EN BASE AL METODO DE COVARIANCIA MODIFICADA Y LA FFT
PROC H2(CHAN OF ANY P1,P5!)
  --{{{  CHOLESKY
--ESTE ES UN METODO DE SOLUCION DE ECUACIONES POR MEDIO DE
--MATRICES UTILIZANDO LAS CARACTERISTICAS PRINCIPALES DE LA
MATRIZ DE
--COFACTORES,ADEMAS,DE LA UTILIZACION DE SOLO PARTE DE DICHA
MATRIZ
  PROC CHOLESKY(VAL[][]REAL32 CC2,[]REAL32 A, VAL INT IP,REAL32 EP,
INT IFLAG )
    --{{{  variables
    [64][64]REAL32 L:
```

```
[513]REAL32 d,y,x;
INT i,col,col1,k,pk;
--}}}
SEQ
  --{{{ INICIALIZACION
  SEQ i=0 FOR IP
    y[i]:=A[i]
  --}}}
  --{{{  triangularizacion
  d[0]:=CC2[0][0]
  SEQ i=1 FOR (IP-1)
    L[i][0]:=CC2[i][0]/d[0]
  d[1]:=CC2[1][1]-(d[0]*L[1][0]*L[1][0])
  SEQ col=1 FOR (P-2
    SEQ
      --{{{  computation of L
      SEQ i=(col+1) FOR ((IP-col)-1)
        SEQ
          L[i][col]:=CC2[i][col]
          SEQ k=0 FOR col
            L[i][col]:=L[i][col]-(L[i][k]*d[k]*L[col][k])
          L[i][col]:=L[i][col]/d[col]
      --}}}
      --{{{  computation of d
      col1:=col+1
      d[col1]:=CC2[col1][col1]
      SEQ k=0 FOR col1
        d[col1]:=d[col1]-(d[k]*(L[col1][k]*L[col1][k]))
      --}}}
  --}}}
  --{{{  resuelve Ly=b
  SEQ col=1 FOR (IP-1)
    SEQ i=col FOR ((P-col)
      y[i]:=y[i]-(L[i][col-1]*y[col-1])
  --}}}
  --{{{  resuelve la diagonal
  SEQ i=0 FOR IP
    z[i]:=y[i]/d[i]
  --}}}
  --{{{  solucion final
  SEQ k=0 FOR (IP-1)
    SEQ
      pk:=(IP-k)-1
      SEQ i=0 FOR pk
        x[i]:=z[i]-(L[pk][i]*x[pk])
  SEQ i=0 FOR IP
    A[i+1]:=-x[i]
  --}}}
:
--}}}
INT IP2,index1,index2, I1,I2,L1,,IP,N,H:
REAL32 C,SUM,TT1,TT2,NG2,TTT,TT3,TT4:
REAL32 EP, PI, ARG,W1,W2,SAVE1,SAVE3,SAVE2:          --60
[64][64]REAL32 CC2:
[513]REAL32 XY,CC,X,CC1,A,A1:
INT MODE,largo,largo1,N,L,M,J,NUM,I,K,LDPT,NDPT,NP,NQ:
INT IFLAG,KK,K:
INT32 screamId, my2:
INT i1,i2,i3:
INT len,TT,PP:
TIMER jtif:
PRI PAR
  SEQ
    --{{{  cov
    PI ? X
    PI ? M
    PI ? IP
    PI ? MODE
    PI ? M
    PI ? EP
    -- recibe datos  del mundo exterior "

    --empieza el calculo  de la matriz de cofactores

    jtif ? i1
    L:=1
    largo1:=largo
    X[N]:=X[0]
    H:=N-IP
    IFLAG:=0  --85
    TTT:=REAL32 ROUND (2 (INT) *(N-IP))
    --SEQ index1 = 1 FOR IP
    --  SEQ
    SEQ index2 = 1 FOR IP
      SEQ
        CC1[L]:=0.0 (REAL32)
        CC[L]:=0.0 (REAL32)
        SEQ I=(IP+1) FOR N
          SEQ
            CC1[L]:=CC1[L]+(X[I-index2]*X[I-1])  --95
          IF
            MODE = 1
              SEQ
                SEQ I=1 FOR H
                  SEQ
                    CC[L]:=CC[L]+(X[I+index2]*X[I+1])
        CC[L]:=(CC[L]+CC1[L])/TTT
        L:= (L  + 1)
    -- aqui  solo se calculo un renglon  de  la matriz de cofactores,  dado que,
    -- este se repetira de alguno u otra forma en toda la matriz
    L:=0
    SEQ index2  = 1 FOR IP
      SEQ
        IP2 :=  IP-L
        CC[(index2 + (L*(IP-1)))]:= CC[index2]
        SEQ index1 = 1 FOR (IP2-1)
          SEQ
            CC[(index2 + (index1*(IP+1)))]:= CC[index2]
            IF
              index2  < > 1
                SEQ

CC[(((L*(IP-1)) + index2) + (index1*(IP+1)))]:= CC[(index2 + (index1*(IP+1)))]
                index2  = 1
                  SEQ
                    K:=0
        L:=L+1
    I:=1
    -- aqui  se designo toda  la matriz a sus respectivos valores dentro de la
    -- mencionada  matriz


    IP2:= IP*IP                                      --105
    SEQ L2=0 FOR IP
      SEQ L3=0 FOR IP
        SEQ
          IF
            L  < = IP2
              SEQ
                CC2[I1,I2]:=CC[I1]
                L:=L+1
            L  > IP2
              SKIP
    -- aqui se designo la matriz de forma cuadrada para  efectos
```

```
~ de la solucion de cholesky

SEQ I = 1 FOR IP                                    ~115
  SEQ
    A[I]: = 0.0 (REAL32)
    A1[I]: = 0.0 (REAL32)
    SEQ I = IP FOR K + 1
      A[I]: = A[I]-(X[I-I]*X[I])
    IF
      MODE = 1
        SEQ I = 1 FOR M
          SEQ
            A1[I]: = A1[I]-(X[I+I]*X[I])
    A[I]: = 0.0 (REAL32)-(A[I]+A1[I])/TTT)
~ aqui es el calculo de la matriz de terminos independientes  * de lado
~ derecho"
SEQ I = 0 FOR IP
  SEQ
    A[I]: = A[I + 1]

CHOLESKY(CC2,A,IP,EP,IFLAG)
~ se calcula la solucion de la ecuacion  CC * II = A
~ esta solucion son los parametros obtenidos del modelo de
~ covarianza  modificada
IF
  IFLAG < > (-1)
    SEQ
      SUM  := 0.0 (REAL32)
      SEQ KK = 1 FOR (IP + 1)
        SEQ
          SEQ
            C: = 0.0 (REAL32)
            K := KK-1
            TT: = 1 (INT)
            LL: = (IP  +  TT)
            SEQ I = LL FOR N-IP
              C: = (C  +  X[I]*X[I-K])
          IF
            MODE = 1
              SEQ
                SEQ I = 1 FOR N-IP
                  SEQ
                    C: = (C + (X[I]*X[I+K]))
          IF
            K = 0
              SUM: = ( SUM + C )
            K < > 0
              SUM: = ( SUM + (C*A[K]) )
      IF
        MODE = 1
          SEQ
            TT1: = REAL32 ROUND N
            TT2: = REAL32 ROUND IP
            SIG2: = SUM/(1.0 (REAL32)*(TT1-TT2))
        MODE < > 1
          SEQ
            TT3: = REAL32 ROUND N
            TT4: = REAL32 ROUND IP
            SIG1 : = SUM/(TT3-TT4)
  IFLAG = (-1)
    SKIP
~ todo el proceso anterior es el calculo de la desviacion estandar
~ o mejor conocida como sigma cuadrada
~}}}
~{{{  m
TT4 : = REAL32 ROUND M
N : = 1

SEQ I = 1 FOR M
  SEQ
    N : = (N * 2 (INT))
~ se calcula el  numero de datos a trabajar en base a la mariposa deseada
A1[1]: = 1.0 (REAL32)
SEQ I = 1  FOR IP
  SEQ
    A1[I + 1]: = A[I]
IP : = (IP  + 1 (INT))

SEQ I = IP  FOR N-IP
  SEQ

    A1[I + 1]: = 0.0 (REAL32)
~ se designaron los parametros en un solo arreglo de tal forma que,
~ el  arreglo sera como: 1,a1,a2,a3,a4,....aIP, 0,0,0,0,.....,0s
CC1[1]: = A1[1]
SEQ I = 2 FOR N-1
  SEQ
    NUM: = I-1
    I: = 0
    L: = N
    SEQ K = 1 FOR M
      SEQ
        L: = (L/2)
        IF
          NUM  > = L
            SEQ
              II: = 1
              SEQ II = 1 FOR K-1
                SEQ
                  II: = (II  + (2 (INT)))
                I: = I  + (II)
              NUM: = NUM-L
          NUM  < L
            SKIP
    CC1[I]: = A1[I + 1] - SIG(II EL CALCULO DE FFT LINEA 188
~ se hace el arreglo obtenido en A  se modifico este , para obtener un
~ nuevo arreglo pero direccionado de la forma bit reverse
SEQ I = 1 FOR N

  CC1[1]: = 0.0 (REAL32)
  PI: = 3.141592654 (REAL32)
  LDPT : = 1
  NDPT : = N
  SEQ K = 1 FOR M
    SEQ
      LDPT : = (2*LDPT)
      NDPT : = (NDPT/2)
      SEQ I = 1 FOR NDPT
        SEQ
          LL: = LDPT/2
          SEQ J = 1 FOR LL
            SEQ
              ARG: = (0.0(REAL32) - (((2.0(REAL32) * PI)*((REAL32 ROUND
J)-1.0(REAL32)))/(REAL32 ROUND LDPT)))
              W1 : = COS(ARG)
              W2 : = SIN(ARG)
              NP: = J + (LDPT*(I-1))
              NQ: = (NP + (LDPT/2))
              SAVE1: = CC1[NP]+((W1*CC1[NQ])-(W2*CC[NQ]))
              SAVE2: = CC[NP]+((W2*CC)[NQ]) +(W1*CC[NQ])
              SAVE3: = CC1[NP]+((W2*CC[NQ])-(W1*CC1[NQ])
              CC[NQ]: = CC[NP]-((W2*CC1(NQ)) +(W1*CC[NQ])
              CC1[NP]: = SAVE1
              CC[NP]: = SAVE2
```

```
        CC1[NQ]:= SAVE3
  -- todo lo anterior  es el calculo de los estados de la FFT con el
  -- calculo de los TWIDLE factors integrado
  SEQ i=1 FOR N
    SEQ
      XY[i] := SIG2/((CC1[i]*CC1[i]) + (CC1[j]*CC1[j])
  -- esta ultima parte es el calculo de la densidad de potencia espectral de
  -- los elementos obtenidos en la FFT
  j4d ? ti2
  i3 := i2 MINUS i1
  W1 := REAL32 ROUND i3
  -- calculos de  tiempos
  PS1 ! N
  PS1 ! W1
  PS1 ! XY
  -- calculos de  datos
  --}}}
SKIP
:
```

PCMFPUN.OCC

```
#INCLUDE "mathvals.inc"
#USE "maglmath.lib"
#USE "occmath.lib"
-- aqui se presenta el programa del calculo de covarianza modificada
-- y su FFT para la obtencion de la densidad de potencia espectral
-- como tambien el proceso de comunicacion de dos transputers tomando
-- al que acumula este programa como intermediario  entre sus
-- comunicaciones
PROC HIZCHAN OF ANY P1,PS1,P3.P2)
  --{{{  CHOLESKY
  --  calculo del metodo de CHOLESKY  para la solucion  del sistema de
  -- ecuaciones  dado por la matriz de cofactores y por la matriz de
  -- estados  independientes .  Aparte  de esta solucion se obtendra
  -- los parametros del modelo deseado

  PROC CHOLESKY(VAL [][]REAL32 CC2,[]REAL32A , VAL INT IP,REAL32 EP,
INT IFLAG )
    --{{{  variables
    [64][64]REAL32 L:
    [513]REAL32 d,y,x:
    INT i,col,col1,k,pk:
    --}}}
    SEQ
      --{{{  inicializacion
      SEQ i=0 FOR IP
        y[i]:= A[i]
      --}}}
      --{{{  triangularizacion
      d[0]:= CC2[0][0]
      SEQ i=1 FOR (IP-1)
        L[i][0]:= CC2[i][0]/d[0]
      d[1]:= CC2[1][1]-(d[0]*(L[1][0]*L[1][0]))
      SEQ col=1 FOR IP-2
        SEQ
          --{{{  calculo de L
          SEQ i=(col+1) FOR (IP-col)-1
            SEQ
              L[i][col]:= CC2[i][col]
              SEQ k=0 FOR col
                L[i][col]:=L[i][col]-(L[i][k]*(d[k]*L[col][k]))
              L[i][col]:= L[i][col]/d[col]
          --}}}
          --{{{  calculo de d
          col1:= col+1
          d[col1]:= CC2[col1][col1]
          SEQ k=0 FOR col1
            d[col1]:= d[col1]-(d[k]*(L[col1][k]*L[col1][k]))
          --}}}
      --}}}
      --{{{  resolucion de  Ly=b
      SEQ col=1 FOR (IP-1)
        SEQ i=col FOR (IP-col)
          y[i]:= y[i]-(L[i][col-1]*y[col-1])
      --}}}
      --{{{  resolucion de la diagonal
      SEQ i=0 FOR IP
        x[i]:= y[i]/d[i]
      --}}}
      --{{{  solucion final
      SEQ k=0 FOR (IP-1)
        SEQ
          pk:= (IP-k)-1
          SEQ i=0 FOR pk
            x[i]:= x[i]-(L[pk][i]*x[pk])
      SEQ i=0 FOR IP
        A[i+1]:= -x[i]
      --}}}
:
--}}}
INT N2,I2,index1,index2, I1,I3,L1, IP,N,H:
REAL32 C,SUM,TT1,TT2,SIG2,TTT,TT3,TT4:
REAL32 EP, PI,W11, ARG,W1,W2,SAVE1,SAVE3,SAVE2:          -60
[64][64]REAL32 CC2:
[513]REAL32 XY,CC,X,CC1,A,A1:
INT MODE,M1,largo,largo1,N,L,M,J,NUM,I,K,LDFT,NDFT,NP,NQ:
INT IFLAG,KK,K:
INT32 starmid,stz2:
INT i1,i2,i3,i1:
INT tes,TT,PP:
TIMER j4d:
PRI PAR
  SEQ
    --{{{  cov
    P1 ? X
    P1 ? N
    P1 ? IP
    P1 ? MODE
    P1 ? M
    P1 ? EP
    -- aqui se recibieron todos  los datos necesarios  para  el proceso

    N2  := (INT 2 (INT ))
    i1 := 0
    SEQ i= N2 FOR 256
      SEQ
        XY[i1] := X[i]

        i1 := i1 + 1
    -- se dividio el arreglo original dividido en dos para ser procesada por
    -- los dos procesadores respectivamente
    W1 := 0.0 (REAL32)

    N := (N/2 (INT))
    P2 ! XY
    P2 ! N
    P2 ! IP
    P2 ! MODE
    P2 ! M
    P2 ! EP
    -- se transmitio la information  necesaria para que el procesador dos
    -- trabaja con una parte del arreglo segmentado anteriormente
    j4d ? t1
```

```
L:=1
largo1:=largo
X[N]:=X[0]
R:=N-IP
IFLAG:=0 --85
TTT:=REAL32 ROUND (2 (INT) *(N-IP))
--SEQ index1 = 1 FOR IP
--  SEQ
SEQ index2 = 1 FOR IP
  SEQ
    CC1[L]:=0.0 (REAL32)
    CC[L]:=0.0 (REAL32)
    SEQ I=(IP+1) FOR N
      SEQ
        CC1[L]:=CC1[L]+(X[I-index2]*X[I-1]) --95
    IF
      MODE = 1
        SEQ
          SEQ I=1 FOR H
            SEQ
              CC[L]:=CC[L]+(X[I+index2]*X[I+1])
    CC[L]:=(CC[L]+CC1[L])/TTT
    L:= L + 1
-- se calculo un solo renglon de la matriz de coeficientes, puesto que,
-- este renglon se repetira a lo largo y a lo ancho de toda la matriz
L:=0
SEQ index2 = 1 FOR IP
  SEQ
    IP2 := IP-L
    CC[(index2 +(L*(IP -1)))]:=CC[index2]
    SEQ index1=1 FOR (IP2-1)
      SEQ
        CC[(index2+(index1*(IP+1)))]:= CC[index2]
        IF
          index2  < >1
            SEQ

CC[(((L*(IP-1))+index2)+(index1*(IP+1))))]:=CC[(index2+(index1*(IP+1)))]
          index2 =1
            SEQ
              K:=0
    L:=L+1
-- aqui se designo de manera adecuada a todos los elementos de la matriz su
-- valor correspondiente en base al unico renglon calculado. Esto es
-- necesario para ganar tiempo al momento de procesar la senal
--K := K+1
L:=1
IP2:= IP*IP
SEQ L2=0 FOR IP                                  --105
  SEQ L3=0 FOR IP
    SEQ
      IF
        L <= IP2
          SEQ
            CC2[L2][L3]:=CC[L]
            L:=L+1
        L > IP2
          SKIP
-- se construyo la matriz en forma cuadrada , dado que el proceso
-- CHOLESKY , asi lo requiere
SEQ J=1 FOR IP                                   --115
  SEQ
    A[I]:= 0.0 (REAL32)
    A1[J]:=0.0 (REAL32)
    SEQ I=(IP+1) FOR N
      A[I]:=A[I]-(X[I-J]*X[I])
    IF
      MODE = 1
        SEQ I=1 FOR H
          SEQ
            A1[I]:=A1[I]-(X[I+J]*X[I])
  A[J]:=0.0 (REAL32)-(A[J]+A1[J]/TTT)
-- Aqui se calculo la matriz de elementos independientes
SEQ J=0 FOR IP
  SEQ
    A[J]:=A[J+1]

-- se designo la matriz de elementos independientes en base al formato
-- requerido por el proceso CHOLESKY
CHOLESKY(CC7,A,IP,EP,IFLAG)

IP
  IFLAG < > (-1)
    SEQ
      SUM  := 0.0 (REAL32)
      SEQ KK=1 FOR (IP+1)
        SEQ
          SEQ
            C: =0.0 (REAL32)
            K := KK-1
            TT:=1 (INT)
            LL:= (IP + TT)
            SEQ I=LL FOR N-IP
              C:=(C  +  (X[I]*X[I-K]))
          IF
            MODE = 1
              SEQ
                SEQ I=1 FOR N-IP
                  SEQ
                    C:=(C + (X[I]*X[I+K]))
          IF
            K=0
              SUM:=( SUM + C )
            K < >0
              SUM:=( SUM + (C*A[K]) )
      IF
        MODE = 1
          SEQ
            TT1:=REAL32 ROUND N
            TT2:=REAL32 ROUND IP
            SIG7:=SUM/(2.0 (REAL32)*(TT1-TT2))
        MODE < > 1
          SEQ
            TT3:=REAL32 ROUND N
            TT4:=REAL32 ROUND IP
            SIG2 :=SUM/(TT3-TT4)
  IFLAG = (-1)
    SKIP
-- el calculo de la estructura IP es la derivacion estandar obtenida de
-- el metodo de covarianza modificada
--}}}
--{{{  fft
-- a continuacion se presenta el calculo de la FFT en base a los
-- parametros obtenidos del metodo de covarianza modificada

TT4 := REAL32 ROUND M
N := 1
SEQ I=1 FOR M
  SEQ
    N := (N *( 2 (INT)))
-- se  calculo el numero de datos requeridos para el numeros de estados
-- de la FFT
A1[1]:= 1.0 (REAL32)
SEQ I=1  FOR IP
```

```
  SEQ
    A1[I+1]:= A[I]
  IP := (IP + 1 (INT))
  SEQ I= IP  FOR N-IP
    SEQ
      A1[I+1]:= 0.0 (REAL32)
  -- se construyo el arreglo requerido por la FFT en base a los parametros
  -- de la covarianza modificada con el siguiente formato:
  -- 1,a1,a2,a3,...aIP,0.0,0.0,0,...,0n
  CC1[I]:= A1[I]
  SEQ I= 2 FOR N-1
    SEQ
      NUM:= I-1
      J:= 0
      L:= N
      SEQ K= 1 FOR M
        SEQ
          L:= (L/2)
          IF
            NUM >= L
              SEQ
                III:= I
                SEQ II= 1 FOR K-1
                  SEQ
                    III := (III * (2 (INT)))
                  J:= J + (III)
                  NUM:= NUM-L
            NUM < L
              SKIP
      CC1[I]:= A1[J+1] -- SIGUE EL CALCULO DE FFT LINEA 188
  -- del arreglo obtenido en  base a los parametros , se reacomodo  de
  -- una nueva forma , tal que,  al final de la FFT se obtenga los resultados
  -- de forma secuencial. La forma en que  se ordenaron es conocida  como
  -- bit-reverse
  SEQ I= 1 FOR N

    CC[I]:=0.0 (REAL32)
  PI:= 3.141592654 (REAL32)
  LDPT := 1
  NDPT := N
  SEQ K= 1 FOR M
    SEQ
      LDPT := (2*LDPT)
      NDPT := (NDPT/2)
      SEQ J= 1 FOR NDPT
        SEQ
          LL:= LDPT/2
          SEQ J= 1 FOR LL
            SEQ
              ARG:= (0.0(REAL32) - ((2.0(REAL32) * PI)*((REAL32 ROUND
J)-1.0(REAL32)))/(REAL32 ROUND LDPT))
              W1 := COS(ARG)
              --W2 := COS(ARG + (PI/(4.0 REAL32)))
              W2 := SIN(ARG)
              NP:= J + (LDPT*(I-1))
              NQ:= (NP + (LDPT/2))
              SAVE1:= CC1[NP] + ((W1*CC1[NQ])-(W2*CC[NQ]))
              SAVE2:= CC[NP] + ((W2*CC1[NQ]) + (W1*CC[NQ]))
              SAVE3:= CC1[NP] + ((W2*CC[NQ])-(W1*CC1[NQ]))
              CC[NQ]:= CC[NP]-((W2*CC1[NQ]) + (W1*CC[NQ]))
              CC1[NP]:= SAVE1
              CC[NP] := SAVE2
              CC1[NQ]:= SAVE3
  -- los ciclos  anteriores son los estados de la FFT y el calculo
  -- de los Twidle factors

  SEQ I= 1 FOR N

    SEQ
      XY[I] := SIO2((CC1[I]*CC1[I]) + (CC[I]*CC[I]))
  -- el ciclo anterior es el calculo final de la densidad de potencia
  -- espectral
  id2 ? id2
  id3 :=  id2 MINUS id1
  W1 := REAL32 ROUND id3

  P3 ? N1
  P3 ? W1[
  P3 ? X
  -- aqui  se recibio la informacion proveniente del modulo dos

  PS1 ! N
  PS1 ! W1
  PS1 ! XY
  -- se transmitio la informacion por este proceso
  PS1 ! N1
  PS1 ! W1[
  PS1 ! X
  -- se transmitio la informacion del modulo dos
  --}}}
SKIP
:
```

SDS4.OCC

```
--{{{  INCLUDEs
#INCLUDE "hostio.inc" --
#INCLUDE "streamio.inc" --
--}}}
--{{{  USEs
#USE "hostio.lib"
#USE "streamio.lib"
#USE "string.lib"
--}}}
PROC lit4(CHAN OF SP fs, ts, CHAN OF ANY PS1)
  --{{{  PROC escritura de datos
  PROC escritura.archivo(VAL INT len, VAL []BYTE filename,
                         INT jdata, []REAL32 data, BYTE brea)
    --{{{  principal
    INT32 id:
    SEQ
      so.open(fs, ts,[filename FROM 0 FOR len], spt.text, spm.output, id, brea)
      IF  --140
        brea = spr.ok
          SKIP
        TRUE
          so.write.string.nl(fs, ts, "el archivo  no fue abierto ")
      SEQ i = 1 FOR jdata
        SEQ
          so.fwrite.real32(fs, ts, id, data[i], 0, 8, brea)
          so.fwrite.nl(fs, ts, id, brea)
      so.close(fs, ts, id, brea)
      IF
        brea = spr.ok
          SKIP
        TRUE
          so.write.string.nl(fs, ts, "archivo no fue  abierto")
          ------------------------------
    --}}}
:
--}}}
INT index1,index2, I.L,IP,N,H:
REAL32 C,SUM,TT1,TT2,SIO2,TT7,TT3,TT4:
REAL64 EP:
```

```
[51]REAL32 X,CC1,A,A1:
INT MODE,largo,largo1,N,L :
BOOL erro,error :
INT IFLAG,KK,K,tiempo:
INT32 streamid,str2:
VAL []BYTE sig IS "signal" :
BYTE result,res2:
BYTE tres :
[60]BYTE filename,filename2:
INT len,TT,PP:
BYTE rea:
SEQ
  PSI ? IP
  PSI ? TT1
  PSI ? A
  PSI ? N
  PSI ? TT2
  PSI ? X
  -- se recibieron todos los parametros provenientes del
  -- de los procesadores externos
  so.write.string(fs,ts,"*c*n*c    el tiempo es :  ")
  so.write.string(fs,ts," proceso 1     ")
  so.write.real32(fs,ts,TT1,0,0)
  so.write.string(fs,ts,"  *c*n*c")
  so.write.string(fs,ts,"            proceso 2    ")
  so.write.real32(fs,ts,TT2,0,0)
  so.write.string(fs,ts,"      *c*n*c*n ")
  -- se da un despliegue de la informacion recibida
  --{{{  escritura de datos  a un archivo
  --{{{  apertura de archivo
  so.write.string.nl(fs, ts, "archivo de salida ? ")
  so.read.echo.line(fs, ts, len, filename, res)
  so.write.string(fs, ts,"*c*n*c*n  ")
  so.write.string(fs,ts,"   da el archivo 2    ")
  so.read.echo.line(fs,ts,TT,filename2,res)
  so.write.string(fs,ts,"*c*n*c*n*c*n     ")
  --}}}
  escritura.archivo(len,filename,IP,A,res)
  escritura.archivo(TT ,filename2,N,X,res)
  -- en los proceso anteriores se almacena
  -- la informacion generada por los procesos externos
  --{{{  despliega datos de archivo
  SEQ
    so.write.string(fs, ts, "archivo de salida ")
    so.write.string(fs, ts, [filename FROM 0 FOR len])
    so.write.string(fs, ts, " tiene  ")
    so.write.int(fs, ts, IP, 0)
    so.write.string(fs, ts, " valores reales  ")
    so.write.nl(fs, ts)
    SEQ i = 0 FOR IP
      SEQ
        so.write.real32(fs, ts, A[i], 0, 0)
        so.write.string(fs,ts," ")
        so.write.real32(fs, ts, X[i], 0, 0)
        so.write.string(fs,ts,"*c*n")
        so.write.nl(fs, ts)
  --}}}
  --}}}
  so.exit (fs,ts, sps.success)
:
```