



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

90
ZED

FACULTAD DE INGENIERIA

DISEÑO E IMPLEMENTACION DE UN NODO HETEROGENEO
DE PROCESAMIENTO DIGITAL DE SEÑALES UTILIZANDO
UN TRANSPUTER Y UN DSP.

T E S I S
QUE PARA OBTENER EL TITULO DE
INGENIERA EN COMPUTACION
P R E S E N T A
DANIELA NORMA RAMOS HERNANDEZ

DIRECTOR DE TESIS:
M. EN C. JORGE MARTINEZ FLORES



MEXICO, D. F.

1995

FALLA DE ORIGEN

TESIS CON
FALLA DE ORIGEN



Universidad Nacional
Autónoma de México



UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

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

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

Dedicó este trabajo:

A la memoria de mi papá Antonio.

A mis dos mamás.

A mis hermanos Judith y Manuel.

Agradecimientos:

Agradezco profundamente al **M. en C. Jorge Martínez Flores** por la asesoría brindada en este trabajo, así como por las enseñanzas y apoyo que siempre me brindo.

Al **Dr. Fabian García Nocetti** y al **Dr. Julio Solano González** por su asesoría y apoyo técnico.

Al Departamento de Electrónica y Automatización del IIMAS de la Universidad Nacional Autónoma de México.

A la Universidad Nacional Autónoma de México y la Dirección General de Intercambio Académico (Proyecto PAPIID-DO303593) por su soporte financiero y al Consejo Nacional de Ciencia y Tecnología por el financiamiento del Sistema Paralelo basado en Transputers (Proyecto PACIME F284-A9209).

Este trabajo de tesis ha generado hasta el momento las siguientes publicaciones:

- 1) *García Nocetti, D. F., Solano J., Martínez J.* "Heterogeneous Architecture for Parallel Real-Time Spectral Estimation in Doppler Blood Flow Instrumentation". Proceedings of the International Conference on Control '94. 21-24 March 1994, University of Warwick, Coventry, UK.

- 2) *Martínez, J., Ramos, D.* "Diseño e Implementación de un Nodo Heterogéneo de Procesamiento Digital de Señales utilizando un Transputer y un DSP". XVI Congreso Nacional Académico de Ingeniería Electrónica. 24-28 Octubre, 1994 Chihuahua, México.

CONTENIDO

1 INTRODUCCIÓN

1.1 MOTIVACIÓN DEL PRESENTE TRABAJO	1-1
1.2 OBJETIVOS	1-3
1.3 CONTENIDO DE LA TESIS	1-3

2 ANTECEDENTES TEÓRICOS

2.1 PROCESAMIENTO EN PARALELO	2-1
2.1.1 AVANCE TECNOLÓGICO DE LAS COMPUTADORAS DIGITALES	2-2
2.1.2 TENDENCIAS	2-3
2.1.3 CLASIFICACIÓN DE LAS ARQUITECTURAS DE PROCESAMIENTO EN PARALELO	2-5
2.1.4 MÉTODOS DE INTERCONEXIÓN	2-9
2.1.5 FORMAS DE INTERCAMBIO DE INFORMACIÓN	2-10
2.1.6 PROGRAMACIÓN DEL SISTEMA	2-11
2.1.7 MODELO DE PARALELISMO CSP	2-13
2.1.8 PROGRAMACIÓN CONCURRENTES	2-14
2.1.9 LENGUAJES DE PROGRAMACIÓN CONCURRENTES	2-19
2.1.10 DISEÑO DE ALGORITMOS EN PARALELO Y APLICACIONES EN SOFTWARE	2-21
2.2 SISTEMA EN TIEMPO REAL	2-23
2.3 PROCESAMIENTO DE SEÑALES DIGITALES	2-26
2.3.1 DEFINICIÓN DE SEÑALES Y SISTEMAS	2-26
2.3.2 ELEMENTOS BÁSICOS DE UN SISTEMA DE PROCESAMIENTO DIGITAL DE SEÑALES	2-27
2.3.3 VENTAJAS DEL PROCESAMIENTO DIGITAL SOBRE EL PROCESAMIENTO ANALÓGICO DE SEÑALES	2-29
2.3.4 CLASIFICACIÓN DE SEÑALES	2-30
2.3.5 CONVERSIÓN ANALÓGICA-DIGITAL Y DIGITAL-ANALÓGICA	2-33
2.3.6 ANÁLISIS DE FRECUENCIA DE SEÑALES Y SISTEMAS	2-36

3 EL TRANSPUTER Y EL DSP

3.1 EL TRANSPUTER	3-2
3.1.1 LA FAMILIA DE PROCESADORES DEL TIPO TRANSPUTER	3-2
3.1.2 CANALES DE COMUNICACIÓN PUNTO A PUNTO	3-4

3.1.3 COMUNICACIÓN	3-5
3.1.4 PROCESAMIENTO EN PARALELO Y EL TRANSPUTER	3-6
3.1.5 REDES DE TRANSPUTERS [HARP, G., 1989]	3-8
3.1.6 EL TRANSPUTER Y SUS LENGUAJES DE PROGRAMACIÓN	3-11
3.1.7 LENGUAJE OCCAM	3-12
3.1.8 INCLUYENDO CÓDIGO ESCRITO EN OTROS LENGUAJES	3-13
3.2 EL PROCESADOR DIGITAL DE SEÑALES (DSP)	3-14
3.2.1 INTRODUCCIÓN A MICROPROCESADORES	3-14
3.2.2 MICROPROCESADORES PARA PROCESAMIENTO DE SEÑALES	3-15
3.2.4 FAMILIA TMS320 DE PROCESADORES DIGITALES DE SEÑALES	3-18

4 ESTUDIO DE ALTERNATIVAS PARA EL DISEÑO DEL NODO DE PROCESAMIENTO HETEROGÉNEO

4.1 EVALUACIÓN DE PROCESADORES	4-1
4.2 MÉTODOS DE INTERCONEXIÓN	4-3
4.3 INTERFAZ POR MEMORIA COMPARTIDA	4-4
4.4 INTERFAZ POR CANAL DE COMUNICACIÓN	4-6
4.5 DISEÑO DEL CANAL DE COMUNICACIÓN EN BASE A UN MICROCONTROLADOR	4-13
4.6 DISEÑO DEL CANAL DE COMUNICACIÓN EN BASE A REGISTROS DE CORRIMIENTO	4-16

5 IMPLEMENTACIÓN DEL NODO DE PROCESAMIENTO HETEROGÉNEO EN BASE A UN TRANSPUTER T805 Y UN DSP TMS320C30

5.1 EL TRANSPUTER IMS T805 Y EL PROCESADOR DE SEÑALES DIGITALES TMS320C30	5-1
5.1.1 EL TRANSPUTER IMS T805	5-1
5.1.2 EL TMS320C30	5-3
5.2 DESCRIPCIÓN DEL NODO DE PROCESAMIENTO HETEROGÉNEO	5-3
5.2.1 SISTEMA DE DESARROLLO PARA EL TRANSPUTER T805	5-5
5.2.2 MÓDULO DE EVALUACIÓN PARA EL TMS320C30	5-7
5.3 TEORÍA DE OPERACIÓN DE LA INTERFAZ DEL NODO DE PROCESAMIENTO HETEROGÉNEO	5-10

6 DESARROLLO DE UNA APLICACIÓN CON FINES EVALUATIVOS

6.1 DESCRIPCIÓN DE LA APLICACIÓN	6-2
6.1.1 EFECTO DOPPLER	6-2
6.1.2 ESPECTRO DE LA SEÑAL DOPPLER	6-7
6.1.2 MÉTODOS CONVENCIONALES	6-11
6.1.3 MÉTODOS NO CONVENCIONALES O PARÁMETRICOS	6-15
6.2 SOFTWARE DE LA APLICACIÓN	6-18

6.2.1 PROGRAMA PARA EL SISTEMA DE PROCESAMIENTO HOMOGÉNEO	6-18
6.2.2 PROGRAMA PARA EL SISTEMA CON EL NODO DE PROCESAMIENTO HETEROGÉNEO	6-22
6.2.3 RESULTADOS	6-25
<u>7 CONCLUSIONES Y COMENTARIOS PARA UN TRABAJO FUTURO</u>	7-1
APÉNDICE A EL TRANSPUTER IMS T805	A-1
APÉNDICE B EL DSP TMS320C30	B-1
APÉNDICE C LENGUAJE C CONCURRENTES	C-1
APÉNDICE D LISTADOS DE PROGRAMAS	D-1

1 Introducción

1.1 Motivación del presente trabajo

El Departamento de Electrónica y Automatización (DEA) del Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS) de la UNAM, cuenta actualmente con un grupo de investigación enfocado al desarrollo de proyectos en el área de procesamiento en paralelo con aplicaciones en tiempo real.

Una de las líneas de investigación se ha orientado al estudio y desarrollo de técnicas de procesamiento de señales Doppler, cuyo objetivo es el de obtener, cuantitativa y cualitativamente, la información que permita realizar la detección temprana de oclusiones en las arterias, causantes de padecimientos cardiovasculares. Esta línea de investigación ha dado pauta a varios trabajos, como los de [Ruano, 1992] y [García Nocetti, 1994]. Los trabajos antes mencionados, presentan la innovación de utilizar técnicas de procesamiento en paralelo en la implementación en tiempo real de algunos métodos de estimación espectral, como la Transformada Rápida de Fourier (FFT) y el algoritmo de Covariancia Modificada.

La experiencia derivada del desarrollo de los trabajos de [Ruano, 1992] y [García Nocetti, 1994], ha mostrado que los transputers pueden calcular y coordinar eficientemente operaciones irregulares en paralelo, particularmente a nivel de procesos. Sin embargo, a pesar de las cualidades ya mencionadas, la arquitectura del transputer ha mostrado deficiencias en aquellas tareas asociadas al procesamiento de señales.

Por todo lo anterior, surge entonces el interés por integrar dentro de una misma plataforma de procesamiento en paralelo un nuevo tipo de procesador con características complementarias a las del transputer. Dichas características se orientan primordialmente al concepto de granularidad. Se puede definir la granularidad como la relación entre el nivel de procesamiento y el de comunicación a la cual esta última se presenta, esto es, a nivel instrucción, a nivel proceso y a nivel programa.

El transputer es considerado como un procesador de granularidad media o gruesa, esto es, porque su capacidad de comunicación se presenta a nivel de proceso y a nivel de programa. Por esta razón, un procesador con características complementarias debe ser de granularidad fina, esto es, que su capacidad de comunicación se presente a nivel de instrucción. De esta forma, el integrar un DSP, considerado un procesador de granularidad fina, en una plataforma de procesamiento en paralelo da origen al concepto de **nodo de procesamiento heterogéneo**.

El DSP seleccionado para el desarrollo del nodo de procesamiento heterogéneo pertenece a la familia TMS320C3x de Texas Instruments, en tanto que los transputers que integran la plataforma de procesamiento en paralelo son de la serie T800 de Inmos.

Se eligió el transputer T805 porque es un procesador robusto y el más completo de Inmos. Dicho transputer cuenta con una unidad de punto flotante que le permite efectuar operaciones en punto flotante a una velocidad de 2.2 MFLOPS a 20 MHz.

Del mismo modo se eligió al DSP TMS320C30 por ser un dispositivo con una elevada capacidad de procesamiento (16.7 millones de instrucciones por segundo con un ciclo de instrucción de 60 nanosegundos) y por disponer asimismo de una unidad de punto flotante de 64 bits, como características más relevantes. Cabe señalar que esta familia de DSPs ha sido utilizada desde su primera generación en el Departamento de Electrónica y Automatización, por lo cual se cuenta con la infraestructura y experiencia necesarias para el desarrollo de proyectos basados en DSPs.

1.2 Objetivos

Los objetivos del trabajo son:

1. Diseño, puesta en marcha y evaluación de un nodo de procesamiento heterogéneo.
2. Reducción de los tiempos de procesamiento logrados con una plataforma de procesadores del tipo transputer, al integrar un nodo de procesamiento heterogéneo.
3. Obtención de una respuesta en tiempo real, utilizando la arquitectura heterogénea propuesta, en una aplicación en flujometría Doppler.

1.3 Contenido de la tesis

Para alcanzar los objetivos de este trabajo de tesis, se presenta a continuación una breve descripción de cada uno de los capítulos que la integran.

Capítulo 1. En este capítulo se da la motivación y los objetivos generales de este trabajo, así como una descripción del contenido de cada capítulo.

Capítulo 2. Proporciona los antecedentes necesarios para la realización de este trabajo, y que son: procesamiento en paralelo, sistemas en tiempo real y procesamiento de señales digitales. En la parte correspondiente al tema de procesamiento en paralelo, se presenta el avance tecnológico de las computadoras digitales, las clasificaciones que existen para arquitecturas en paralelo, las técnicas de interconexión e intercambio de información entre sistemas multiprocesadores y finalmente los lenguajes concurrentes, que permiten explotar el paralelismo así como resolver problemas de comunicación y sincronización entre procesos.

En la sección de sistemas en tiempo real, se define lo que es un sistema en tiempo real, los tipos de sistemas existentes y se menciona la importancia de manejar lenguajes de alto nivel concurrentes para lograr las características básicas de un sistema de este tipo. Por último, se describen los elementos que forman un procesador digital de señales, los diferentes

procesadores, así como también los lenguajes de programación concurrente que existen y el diseño de algoritmos paralelos.

2.1.1 Avance tecnológico de las computadoras digitales

Con el surgimiento de las computadoras digitales y la búsqueda constante por incrementar las velocidades de procesamiento, permitió el superar las limitaciones del llamado cuello de botella de la máquina von Neumann [Hwang, K., y Degroot, D., 1989]. En las computadoras digitales secuenciales, el buffer de memoria sirve como el único medio de acceso entre la memoria de alta velocidad y la unidad de procesamiento central, ocasionando que todas las tareas de cómputo se lleven a cabo en forma secuencial, incrementando el tiempo de procesamiento.

El avance de la tecnología con respecto al hardware, ha permitido incrementar la velocidad de las operaciones aritméticas individuales. Dos innovaciones en el diseño del hardware de computadoras digitales han evitado el efecto de cuello de botella de la máquina von Neumann ya citado: *el pipeline* (traslape en la ejecución de instrucciones) y *el paralelismo*. La implementación de estas técnicas ha permitido el surgimiento de distintas familias de computadoras digitales de alta velocidad como supercomputadoras, arreglos de procesadores y multiprocesadores. La primera supercomputadora fue la ILLIAC IV, la cual estaba formada por un arreglo de 64 elementos de procesamiento, cada uno con una unidad aritmética y lógica, y una memoria local propia [Hwang, K., y Degroot, D., 1989].

En una estructura pipeline, las operaciones aritméticas son fraccionadas en estados sucesivos y las unidades de hardware realizan los cálculos de cada estado. En los años 70's se desarrollaron dos supercomputadoras con el diseño pipeline. Estas fueron la STAR-100 de Control Corporation Data y la ASC de Texas Instruments. Dichas computadoras son conocidas con el nombre de *procesadores vectoriales*, pues funcionan eficientemente sólo si las operaciones aritméticas a ser ejecutadas son vectorizadas, es decir, arregladas como flujos continuos de datos. La supercomputadora Cray-1 desarrollada por Cray Research fue la más aceptada con este diseño [Hwang, K. y Degroot, D., 1989].

Al final de los años 70's aparecieron nuevos dispositivos en el mercado denominados *arreglos de procesadores*. Actualmente se conocen como

procesadores vectoriales de pequeña escala y se pretende que funcionen como dispositivos periféricos para computadoras digitales secuenciales. Por esta razón, se les conoce también como *arreglos de procesadores periféricos* o *arreglos de procesadores de unión*. El funcionamiento de estos arreglos se basa en la comunicación con una computadora digital (*host*) a través de un acceso directo a memoria, o bien, canales de comunicación de entrada/salida, mejorando la eficiencia del host por medio de un paralelismo interno extensivo y pipeline.

A mediados de los ochentas aparece el multiprocesamiento, que es el resultado de unir procesadores vectoriales semejantes a las supercomputadoras pero a menor costo, por lo cual, se les denomina *superminicomputadoras*. Una superminicomputadora logra el 25 por ciento del rendimiento de una Cray-1S a un costo del 10 por ciento [Hwang, K., y Degroot, D., 1989].

Para los años 90's aparecen sistemas multiprocesadores con 16 procesadores como el proyecto S-1 del Lawrence Livermore National Laboratory y el sistema multiprocesador HEP de Denelcor. Se espera que los sistemas de esta década alcancen más de 1000 megaflops (millones de operaciones en punto flotante por segundo).

2.1.2 Tendencias

El uso principal de las computadoras se da en 4 niveles ascendentes [Hwang, K. y Briggs, F., 1984]:

- Procesamiento de datos
- Procesamiento de información
- Procesamiento de conocimiento
- Procesamiento de inteligencia

Podemos concebir el espacio de datos como un universo que incluye números en varios formatos, símbolos y medidas multidimensionales, donde los datos carecen de relación alguna entre ellos. La información por su parte, es una colección de datos que están relacionados por alguna estructura sintáctica o relacional. La información forma subespacios del espacio de datos. El conocimiento consiste a su vez de información más un significado semántico. El conocimiento forma un

subespacio del espacio de información. Finalmente, la inteligencia se deriva de una colección de partes del conocimiento.

El uso de las computadoras se inició con el procesamiento de datos. Posteriormente, con el desarrollo de más estructuras de datos, muchos usuarios han cambiado al procesamiento de información. En estos dos niveles de procesamiento se encuentra un alto grado de paralelismo. Como las bases de conocimiento se han expandido rápidamente, existe gran demanda para el procesamiento del conocimiento y su uso ha sido proyectado para los 90's. Sin embargo, el procesamiento de inteligencia está más lejos de realizarse en esta década, ya que las computadoras son incapaces de comunicarse con el hombre de forma natural y más aún, en demostrar teoremas o realizar inferencia lógica o pensar creativamente. Al llegar a estos niveles de procesamiento más altos (conocimiento e inteligencia) el grado de paralelismo explotable podrá ser más grande que con los niveles de procesamiento de datos o de información.

Por su parte, desde el punto de vista de un sistema operativo, los sistemas de cómputo se han perfeccionado cronológicamente en cuatro fases:

- Procesamiento por lotes
- Multiprogramación
- Tiempo compartido
- Multiprocesamiento (Procesamiento en paralelo)

En estos 4 modos operativos, el grado de paralelismo incrementa rápidamente de fase a fase. La tendencia general es enfatizar el procesamiento en paralelo de información, usando el término de información para incluir datos, información, conocimiento e inteligencia.

Procesamiento en paralelo

El procesamiento en paralelo es una eficiente forma de procesar información, el cual enfatiza la explotación de eventos concurrentes en el cálculo de procesos. Concurrencia implica paralelismo, simultaneidad y pipeline. Eventos paralelos pueden ocurrir en múltiples recursos durante el mismo intervalo de tiempo, eventos simultáneos pueden ocurrir en el mismo instante de tiempo, y eventos en pipeline ocurren en intervalos de tiempo traslapados [Hwang, K. y Briggs, F., 1984].

Definición. Es la actividad de varias entidades (idénticas o heterogéneas), trabajando conjuntamente hacia un objetivo común [Carlini, 1991]. Para el caso del cómputo paralelo, dichas entidades corresponden a computadoras o procesadores.

2.1.3 Clasificación de las arquitecturas de procesamiento en paralelo

Un sistema de procesamiento en paralelo está compuesto por varios elementos de procesamiento (PEs), los cuales pueden operar comunicándose unos con otros cuando es necesario. Las arquitecturas en paralelo se caracterizan por el poder de procesamiento de sus PEs y por el grado de interconectividad entre ellos. Esto último define el concepto de granularidad, que es la forma de medir las tareas que pueden ser ejecutadas eficientemente por los PEs. Los PEs de arquitecturas con granularidad fina se caracterizan por ser elementos de funcionalidad limitada o específica y por tener un amplio ancho de banda para la comunicación local de datos. Por otro lado, los PEs de arquitecturas con granularidad gruesa o mediana son de propósito general y tienen un ancho de banda reducido para la comunicación entre los elementos de procesamiento.

Las arquitecturas en paralelo son clasificadas de acuerdo a diferentes criterios. Una de las clasificaciones de sistemas en paralelo más usadas es la sugerida por Flynn [Irwin, G. y Fleming, P., 1992], que considera al modelo secuencial de la máquina von Neumann como un flujo simple de instrucciones operando en un flujo simple de datos SISD (*Single Instruction-stream Single Data-stream*).

Las arquitecturas en paralelo son clasificadas de acuerdo a la taxonomía de Flynn, (figura 2.1) en las siguientes categorías:

1. Flujo de instrucción simple-flujo de datos múltiple (SIMD, *Single Instruction-stream Multiple Data-stream*).

La misma instrucción es transmitida a todos los PEs los cuales ejecutan esta instrucción simultáneamente con diferentes datos. Los arreglos de procesadores como ICL DAP y TMC Connection Machine son un ejemplo de esta arquitectura y son apropiados para la implementación de algoritmos que involucran operaciones con matrices.

2. Flujo de múltiples instrucciones-flujo único de datos (MISD, *Multiple Instruction-stream, Single Data-stream*).

Varios procesadores ejecutan simultáneamente diferentes instrucciones con un mismo flujo de datos. Existen muy pocos ejemplos para esta clasificación, considerándola impráctica. Un sistema pipeline es un ejemplo típico de una arquitectura MISD.

3. Flujo de múltiples instrucciones-flujo de múltiples datos (MIMD, *Multiple Instruction-stream Multiple Data-stream*).

Esta arquitectura está formada por varios procesadores independientes, cada uno capaz de ejecutar diferentes instrucciones con diferentes datos. Un arreglo de *transputers* es un ejemplo de una arquitectura MIMD.

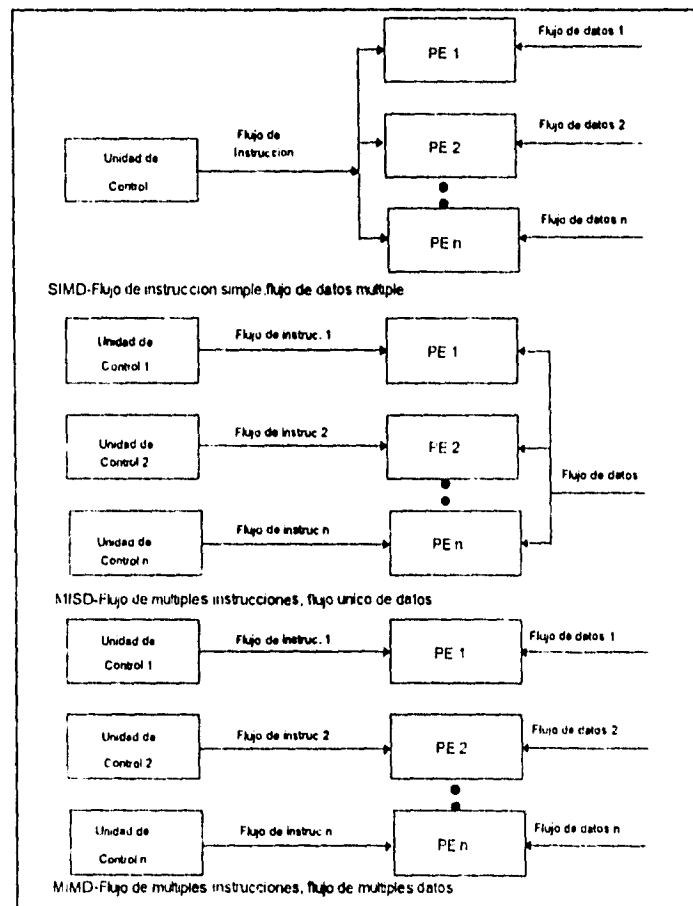


Figura 2.1 Clasificación de Flynn.

En general, las arquitecturas SIMD disponibles son consideradas arquitecturas de granularidad fina, con tareas individuales de una sola operación, mientras que las arquitecturas MIMD son de granularidad media o gruesa.

Cabe señalar dos esquemas más de clasificación de arquitecturas: el esquema de Feng basado en el procesamiento serial vs paralelo y la clasificación de Händler que determina el grado de paralelismo y pipeline en varios niveles de subsistemas [Hwang, K. y Briggs, F., 1984].

Procesamiento serial vs paralelo. Tse-Yun Feng sugiere el uso del grado de paralelismo para clasificar varias arquitecturas de computadoras. El número máximo de dígitos binarios (*bits*) que pueden ser procesados en una unidad de tiempo por un sistema es llamado *máximo grado de paralelismo (P)*.

El máximo grado de paralelismo $P(c)$ de un sistema de cómputo c es representado por el producto de la longitud de palabra n y la longitud bit-slice (cadena de bits) m , esto es

$$P(c) = n \cdot m$$

Dentro de esta clasificación existen cuatro métodos de procesamiento:

1. Procesamiento WSBS (*Word-serial and bit-serial*). Este procesamiento ha sido llamado también procesamiento bit-serial debido a que se procesa un bit ($n = m = 1$) en una unidad de tiempo. Este tipo de procesamiento se dió en la primera generación de computadoras, por ejemplo la máquina llamada MINIMA.
2. Procesamiento WPBS (*Word-parallel and bit-serial*). A este tipo de procesamiento donde ($n = 1, m > 1$) se le conoce como procesamiento bis o bit-slice ya que se procesan m bit-slice en una unidad de tiempo. Procesadores bit-slice son STARAN, MPP y DAP.
3. Procesamiento WSBP (*Word-serial and bit-parallel*). El procesamiento WSBP ($n > 1, m = 1$) se llama procesamiento word-slice, debido a que se procesa una palabra de n bits en una unidad de tiempo. Ejemplos de este tipo de procesamiento se encuentra en Illiac-IV y PEPE.

4. Procesamiento WPBP (*Word-parallel and bit parallel*). El procesamiento WPBP ($n > 1, m > 1$) es conocido como procesamiento en paralelo, en el cual un arreglo de $n \cdot m$ bits son procesados en una unidad de tiempo, resultando el modo de procesamiento más rápido de los cuatro.

Paralelismo vs Pipelining. Wolfgang Händler propone un esquema de clasificación para identificar el grado de paralelismo y pipeline construido en las estructuras de hardware de un sistema de cómputo. Las estructuras de hardware consideradas son:

- Unidad de control del procesador (CPU)
- Unidad aritmética-lógica (ALU)
- Bit-level circuit (BLC)

Para obtener el grado de paralelismo y pipeline se aplica la siguiente ecuación:

$$T(c) = \langle K \times K', D \times D', W \times W' \rangle$$

donde:

K=Número de procesadores (CPUs) dentro de la computadora

D=Número de ALUs (o PEs) bajo el control de un CPU

W=Longitud de palabra de un ALU o de un PE

W'=Número de etapas pipeline en todos los ALUs o en un PE

D'=Número de ALUs que pueden estar en pipeline

K'=Número de CPUs que pueden estar en pipeline

T(c)=Modelo de computadora

Por ejemplo, el grado de paralelismo y pipeline de la máquina ASC (*Advanced Scientific Computer*) de Texas Instruments, se obtiene considerando que esta máquina tiene un controlador con 4 ALUs (pipelines), cada uno con una palabra de 64 bits y 8 estados o etapas.

$$\begin{aligned} T(ASC) &= \langle 1 \times 1, 4 \times 1, 64 \times 8 \rangle = \langle 1, 4, 64 \times 8 \rangle \\ &= \langle 1, 4, 512 \rangle \end{aligned}$$

2.1.4 Métodos de Interconexión

De acuerdo a la clasificación de Flynn, las máquinas MIMD ofrecen mayor flexibilidad en la interconexión de un sistema multiprocesador que las máquinas SIMD. Además, un sistema MIMD abarca un mayor número de aplicaciones que un sistema SIMD, ya que este último es un subconjunto de un sistema MIMD.

Las técnicas de interconexión utilizadas para conectar elementos de procesamiento de un sistema MIMD a otro y a dispositivos periféricos es la mejor forma para determinar el rendimiento de un sistema multiprocesador. El método más empleado es determinar si las interconexiones son dinámicas o estáticas [Hwang, K. y Degroot, D., 1989].

En una red de **interconexión dinámica** existen interruptores que permiten a los elementos de procesamiento, módulos de memoria y dispositivos periféricos, conectarse unos con otros mediante un programa de control. Algunos medios de interconexión dinámica son : el *crossbar switch* y la *estructura de bus*. En esta última, todos los elementos comparten un sólo medio de conexión y se emplean una variedad de técnicas como lo es el paso de una muestra (*token passing*) ó la transmisión de una porción de tiempo (*time-sliced broadcasting*) para asegurar que los mensajes sean recibidos correctamente.

Por otra parte, en una **interconexión estática** los elementos de procesamiento son arreglados en patrones multidimensionales y son conectados de manera permanente unos a otros. Las topologías de redes estáticas incluyen estructuras de una o dos dimensiones como el anillo, la estrella, la malla de vecinos cercanos y el arreglo sistólico. El hipercubo es una topología que minimiza la distancia máxima entre los elementos de procesamiento en redes grandes. El número de elementos de procesamiento (N) es igual a una potencia de 2 y cada uno de ellos se conecta a $\log_2 N$ vecinos cercanos. Para $N = 4$ y $N = 8$, las configuraciones hipercúbicas son el cuadrado y el cubo, respectivamente. Para un gran número de elementos de procesamiento las redes toman configuraciones complejas denominadas configuraciones hipercúbicas de dimensiones superiores. La figura 2.2 muestra las topologías hipercúbicas antes mencionadas.


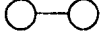
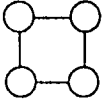
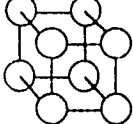
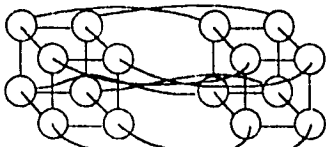
Dimensiones hipercúbicas				
Dimensión	Nodos	Canales/Nodo	Canales	Topología
0D	1	0	0	
1D	2	1	1	
2D	4	2	4	
3D	8	3	12	
4D	16	4	32	

Figura 2.2 Topologías hipercúbicas.

2.1.5 Formas de intercambio de información

Muchos sistemas multiprocesadores (MIMD) emplean memorias compartidas. En estos sistemas los elementos de procesamiento tienen acceso a recursos de memoria comunes y el intercambio de datos es mediante operaciones de registro y lectura a localidades de memoria determinadas. Este tipo de acceso a los recursos puede ocasionar problemas de contención cuando el tráfico de datos es muy grande. Para evitar este problema, los sistemas de procesamiento actuales emplean accesos rápidos o unidades caché que actúan como buffers en las memorias usadas con mayor frecuencia. Una mejor alternativa que las memorias compartidas son los sistemas llamados paso de mensajes (*message-passing*). En éstos, cada elemento de procesamiento tiene una memoria local propia y no existen unidades de memoria compartida. De esta forma, en lugar de que exista intercambio de datos, los datos son transmitidos como mensajes entre dos elementos de procesamiento.

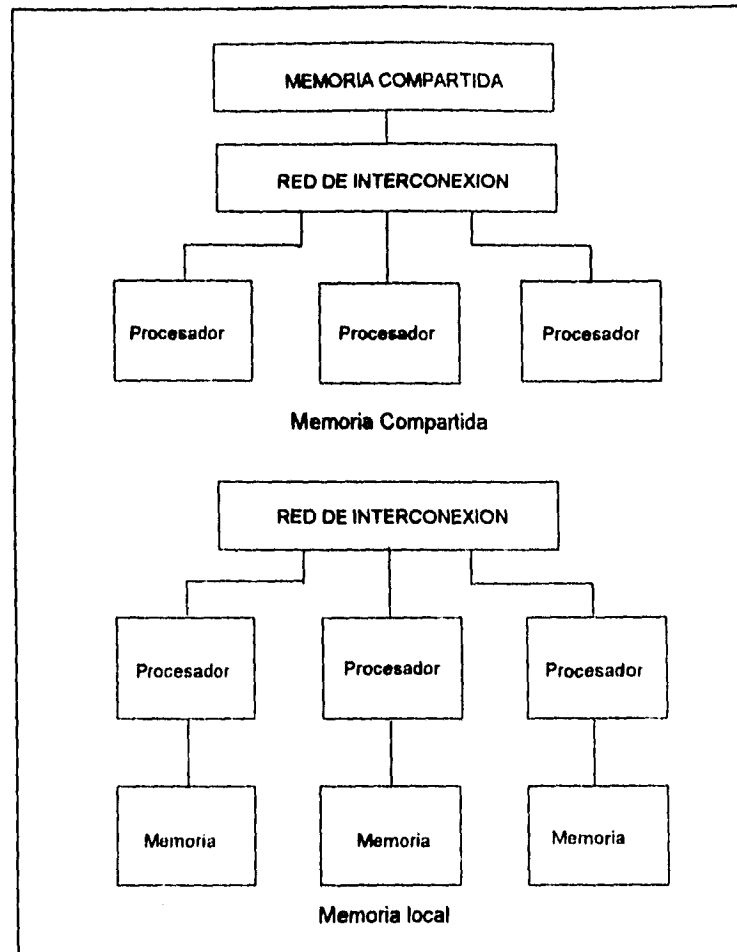


Figura 2.3 Gráfica de memoria compartida y memoria local.

2.1.6 Programación del sistema

Para hacer uso efectivo de los sistemas multiprocesadores es necesario asegurar que todos los elementos de procesamiento estén activos la mayor parte del tiempo. Adicionalmente, es necesario minimizar el tiempo dedicado a la comunicación entre procesadores.

Para la simulación de grandes problemas, el programa debe ser dividido y cada porción debe ser alojada en cada uno de los diferentes elementos de procesamiento, los cuales requieren de un algoritmo independiente. Asimismo, el programa de entrada y el flujo de datos deben ser cuidadosamente planeados y sincronizados. La combinación de los

problemas de partición, planeación y sincronización es la parte más desafiante en la implementación de las tareas en sistemas multiprocesadores.

Niveles de paralelismo

El paralelismo se refiere a la ejecución simultánea de tareas, partes de tareas, programas, rutinas, subrutinas, iteraciones o declaraciones. En la **tabla 2.1** se muestran los diferentes niveles de paralelismo en la ejecución de un programa [Hwang, K. y Degroot, D., 1989].

Tabla 2.1 Cinco niveles de paralelismo en la ejecución de un programa.

NIVEL	PARALELISMO
1	Programas y tareas independientes
2	Partes de tareas y partes relacionadas de programas
3	Rutinas, subrutinas y corutinas
4	Lazos de repetición e iteraciones
5	Declaraciones e instrucciones

El procesamiento en paralelo se refiere al paralelismo explotado por alguno o bien, la combinación de varios niveles de la tabla 2.1. El procesamiento por vector es una ejecución en paralelo por iteraciones de lazos de repetición a nivel 4. La ejecución en paralelo de declaraciones independientes a nivel 5 ha sido implementado en muchas máquinas con la técnica *look-ahead* usando múltiples unidades funcionales. Hoy en día muchas computadoras soportan multiprogramación, lo que significa que comparten los recursos del procesador, independientemente de los procesos. La multiprogramación se lleva a cabo en sistemas uniprocador, en los cuales el CPU y las actividades de E/S son intercaladas. El uniprocamiento explota el paralelismo en el nivel 1 en un modo múltiple_SISD. La multitarea es un caso especial de multiprocesamiento definiendo una tarea como una parte de un subprograma en los niveles 2 y 3. Para máquinas con un menor número de procesadores como la Cray X-MP el paralelismo es ejecutado en los niveles más altos (1, 2 y 3) a través de los procesadores. Sin embargo, dentro de cada procesador, el paralelismo es llevado a cabo en los niveles 4 y 5. Para máquinas de paralelismo masivo como lo es la MPP, el paralelismo es ejecutado en los niveles más bajos.

2.1.7 Modelo de paralelismo CSP

El modelo CSP (*Communicating Sequential Processes*), fue desarrollado por C.A.R. Hoare, y representa un modelo generalizado de concurrencia basado en la idea de ejecución de procesos de manera independiente, intercambiando datos unos con otros por conexiones llamados canales de comunicación. Un lenguaje que se adapta al modelo CSP es el lenguaje C concurrente [Inmos ANSI C, 1990].

Procesos

Los procesos son los elementos principales del modelo CSP. Un proceso puede ser visto como una caja negra con un estado interno, el cual se puede comunicar con otros procesos usando canales de comunicación de punto a punto. Los procesos son utilizados para representar el comportamiento de muchas cosas, por ejemplo, una compuerta lógica, un microprocesador, una máquina o una oficina.

Los procesos por sí mismos son finitos. Cada proceso inicia, ejecuta un número de acciones y después termina. Una acción puede ser a su vez un conjunto de procesos secuenciales ejecutándose uno después de otro, como en un lenguaje de programación convencional, o un conjunto de procesos paralelos ejecutados al mismo tiempo junto con otros.

La **figura 2.4** muestra la vida de un proceso, el cual pasa por varios estados: creación, inicialización, ejecución y terminación. Algunos procesos pueden no terminar y otros pueden fallar durante la inicialización, pasando directamente a la terminación sin ser ejecutados. Después de la terminación, un proceso pasa a un estado de no existencia cuando no es accesado.

Todos los procesos están contruídos por 3 primitivas: asignación, entrada y salida. Una asignación calcula el valor de una expresión asignando éste a una variable. La entrada y salida son usadas para la comunicación entre procesos [Inmos, 1989].

Un proceso describe el comportamiento de un componente discreto de una aplicación: Este puede consistir de otros procesos, operaciones secuenciales o de combinaciones de éstos. Las aplicaciones pueden ser

fraccionadas en varios procesos y los procesos pueden ser mapeados en un sistema multiprocesador [Inmos ANSI C, 1990].

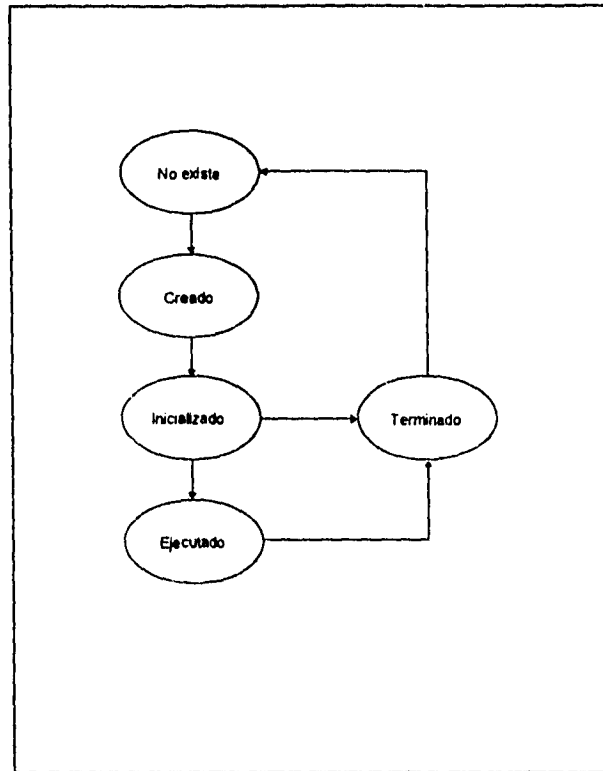


Figura 2.4 Ciclo de vida de un proceso.

2.1.8 Programación concurrente

La programación concurrente es el nombre con el que se conocen las técnicas de programación para expresar el potencial del paralelismo y resolver problemas de sincronización y comunicación. La implementación del paralelismo es independiente de la programación concurrente [Burns, A. y Wellings, A., 1989].

Dijkstra (1968) define un programa concurrente como una colección de procesos secuenciales autónomos ejecutándose lógicamente en paralelo. Los lenguajes de programación concurrente incorporan implícita o explícitamente la noción de proceso. Cada proceso tiene una sola ruta de ejecución.

La ejecución de una colección de procesos puede tener alguna de las siguientes tres formas:

1. Múltiples ejecuciones en un sólo procesador
2. Múltiples ejecuciones en un sistema multiprocesador con memoria compartida
3. Múltiples ejecuciones en muchos procesadores los cuales no comparten memoria (sistemas distribuidos)

En un programa concurrente los procesos deben ser creados, enviados a los procesadores disponibles y terminados.

La construcción de un programa concurrente varía de un lenguaje a otro, existiendo tres características fundamentales:

1. Ejecución concurrente por medio de procesos.
2. Sincronización de procesos.
3. Comunicación entre procesos.

De acuerdo a la noción de un proceso, es común que los lenguajes de programación concurrente tengan variaciones en los modelos de concurrencia adoptados. Esas variaciones son:

- Estructura
- Nivel
- Granularidad
- Inicialización
- Terminación
- Representación

La estructura de un proceso se clasifica como:

1. Estática: si el número de procesos es fijo al momento de compilar.
2. Dinámica: si los procesos son creados en cualquier momento.

El nivel de paralelismo soportado puede ser:

1. Anidado (*Nested*): los procesos son definidos en cualquier nivel del texto del programa, en particular se permite definir procesos dentro de otros procesos.

2. Básico (*Flat*): los procesos son definidos fuera del texto del programa.

Los lenguajes que soportan anidamiento también pueden distinguirse por su tipo de granularidad: paralelismo de granularidad gruesa y paralelismo de granularidad fina. Un programa concurrente de granularidad gruesa contiene pocos procesos, pero cada uno con una vida trascendente. Un programa paralelo con granularidad fina tiene un gran número de procesos simples, algunos de los cuales pueden existir por una acción simple. Un ejemplo de un programa de paralelismo de granularidad gruesa es aquel escrito en el lenguaje Ada y de granularidad fina en lenguaje Occam 2 [Burns, A. y Wellings, A., 1987].

Tabla 2.2 Características de estructura y nivel de paralelismo para varios lenguajes de programación concurrente.

LENGUAJE	ESTRUCTURA	NIVEL DE PARALELISMO
Pascal concurrente	Estática	Básico
DP	Estática	Básico
Occam 2	Estática	Anidado
Pascal-Plus	Estática	Anidado
Modula-1	Dinámica	Básico
Modula-2	Dinámica	Básico
Ada	Dinámica	Anidado
Mesa	Dinámica	Anidado
C	Dinámica	Anidado

Cuando un proceso es creado, puede necesitar información para su ejecución. Hay dos formas de efectuar la inicialización: la primera es pasar información en la forma de parámetros al proceso; la segunda es comunicarse explícitamente con el proceso después de que ha comenzado su ejecución.

La terminación de un proceso puede realizarse de diferentes maneras:

- Finalización de la ejecución del proceso
- Autoterminación, por la ejecución de una instrucción
- Terminación por la acción explícita de otro proceso
- Terminación sin condición de error
- Cuando los procesos ejecutan lazos de repetición interminables
- Cuando no se requiere que el proceso sea tan largo

Existen cuatro mecanismos básicos para representar la ejecución concurrente de un proceso: fork y join, cobegin, declaración explícita de un proceso y las corutinas [Burns, A. y Wellings, A., 1987].

Corutinas: Parecidas a las subrutinas, pero éstas permiten pasar el control explícitamente entre ellas, en un camino más simétrico que jerárquico. Las corutinas no son adecuadas para procesamiento en paralelo, ya que su semántica sólo permite la ejecución de una corutina a la vez.

Fork y join: La instrucción fork especifica que una rutina puede iniciar su ejecución concurrente con otra rutina al invocarla con una instrucción fork y la instrucción join permite sincronizar la terminación de una rutina con otra. Por ejemplo:

```
function F return...,
procedure P;
...
C:=fork F;
.
.
J:=join C;
...
end P;
```

A partir de la instrucción fork y hasta la instrucción join, el procedimiento P y la función F son ejecutados en paralelo. Estas dos instrucciones permiten la creación dinámica de procesos y proporcionan el paso de mensajes al proceso hijo por medio de parámetros.

Cobegin: Cobegin (o parbegin o par) es una forma estructurada para denotar la ejecución concurrente de una colección de instrucciones:

```
cobegin
S1;
S2;
S3;
.
.
Sn
coend
```

Este código ocasiona que las instrucciones S1, S2 etc. sean ejecutadas concurrentemente. La instrucción cobegin termina cuando todas las instrucciones concurrentes han terminado. Este mecanismo se encuentra en el lenguaje Occam 2 (como PAR).

Declaración de procesos explícitamente: Se refiere a la ejecución concurrente de las rutinas sin usar instrucciones como fork o cobegin. Ejemplos de lenguajes de programación con ejecución concurrente son Modula-2, Occam 2 y Ada.

Comunicación y sincronización de procesos

La interacción de procesos requiere de lenguajes de programación concurrente que soporten sincronización y comunicación de datos. La comunicación puede estar basada en variables compartidas o en el paso de mensajes.

En los sistemas basados en el paso de mensajes la sincronización es implícita, debido a que un proceso que recibe no puede obtener un mensaje antes de que el mensaje sea enviado. Con un sistema basado en variables compartidas el proceso que recibe puede leer una variable y no conocer si ésta ha sido escrita por el proceso que envía. Variaciones en los modelos de sincronización de procesos surgen de la semántica de la operación de envío.

La sincronización de procesos se puede clasificar como:

1. Asíncrono (o *no-wait*): el que envía procede de inmediato, sin preocuparse si el mensaje es recibido o no. Este tipo de sincronización se encuentra en los sistemas PLITS, CONIC, y algunos sistemas operativos.
2. Síncrono: el que envía continúa sólo cuando el mensaje ha sido recibido. Ejemplos de esta sincronización son CSP y Occam 2.
3. Invocación remota: el que envía continúa sólo cuando el receptor le envía una contestación. En este caso, el receptor contesta de inmediato. Este tipo de sincronización es encontrada en Ada, SR, CONIC y varios sistemas operativos.

2.1.9 Lenguajes de programación concurrentes

Para implementar algoritmos rápidos se requiere de un lenguaje de programación de alto nivel que posea las siguientes características:

1. Flexibilidad. El lenguaje debe dar facilidad al programador de especificar varias formas de paralelismo en los programas de aplicación.
2. Eficiencia. El lenguaje debe ser fácilmente implementado en varios sistemas de computadora (paralelo/vector).

Debido a los problemas que se presentan cuando un lenguaje no posee estas características, se han desarrollado las siguientes aproximaciones:

Aproximación de un compilador

La mayoría de los paquetes de software generan su código a través de lenguajes secuenciales, como por ejemplo Fortran. Existen compiladores inteligentes que detectan paralelismo en programas secuenciales convirtiéndolo en código paralelo (ejemplos de estos compiladores son el compilador CFT de la Cray X-MP y el compilador KAP/205 diseñado para la Cyber 205), figura 2.5. La ventaja de esta aproximación es que el software que está en código secuencial puede ser usado en computadoras de procesamiento en paralelo con muy pocas modificaciones. Sin embargo, el lenguaje secuencial fuerza al programador a codificar sus algoritmos en paralelo en forma secuencial, y los tipos de paralelismo detectado se reducen a estructuras de datos como vectores, matrices o listas.

Aproximación de extensión de lenguaje

Los lenguajes secuenciales pueden ser extendidos construyendo arquitecturas orientadas para soportar programación concurrente. Regularmente sólo un tipo de paralelismo es soportado en cada lenguaje con extensión. La figura 2.6 muestra la extensión de lenguaje para Fortran y C.

Para que Fortran tuviera multitasking en la Cray X-MP, se le aumentaron las instrucciones EVWAIT, LOCKON, y TASKSTART. El lenguaje C concurrente se extendió del C estándar para el multicomputer Flex/32. Otras

extensiones se dieron en la Vectran, Actus-2 y Vector C para soportar procesamiento vectorial en procesadores pipeline o arreglos de procesadores. Si las extensiones se realizan en máquinas orientadas la implementación resulta eficiente, pero si la máquina es dependiente implica poca portabilidad.

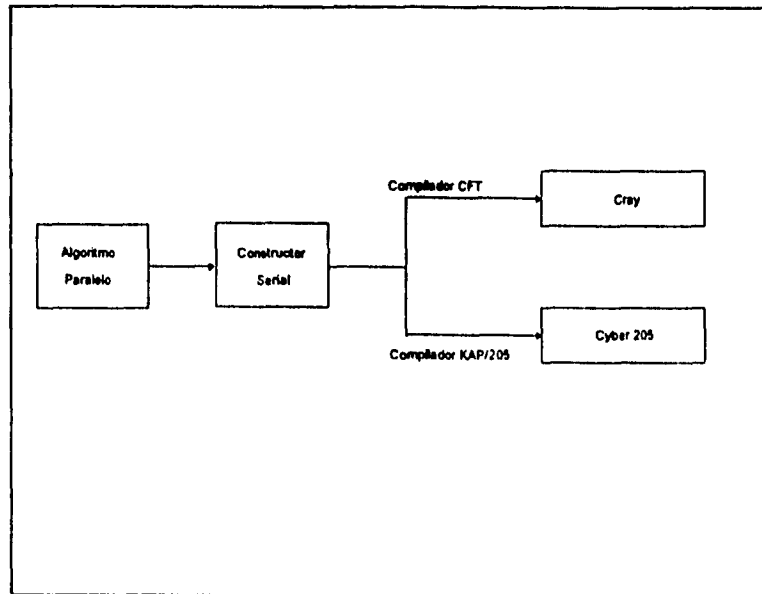


Figura 2.5 Aproximación de un pequeño compilador.

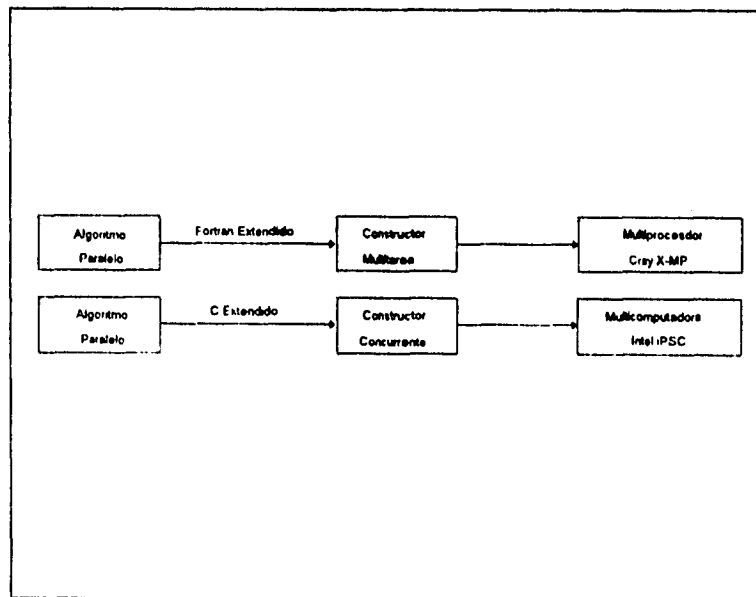


Figura 2.6 Aproximación de lenguajes de extensión.

tipos de señales que existen, las ventajas de manejar señales digitales en lugar de señales analógicas, el análisis de las señales en el dominio de la frecuencia, su clasificación y la herramientas disponibles para efectuar este análisis.

Capítulo 3. Describe las características generales de los transputers de Inmos y los procesadores digitales de señales (DSPs) de la familia TMS320C3x de Texas Instruments.

Capítulo 4. Se analizan dos alternativas para el diseño de la interfaz de comunicación entre el transputer T805 y el DSP TMS320C30: memoria compartida y canales de comunicación. Se describen dos implementaciones realizadas con canales de comunicación. La primera de ellas con un microcontrolador y la segunda con registros de corrimiento.

Capítulo 5. Detalla la interfaz de comunicación realizada con registros de corrimiento. Se describen los módulos que la integran y los sistemas de desarrollo del transputer (TEK805) y del TMS320C30 (EVMTMS30), utilizados para la implementación de la interfaz.

Capítulo 6. Describe la aplicación utilizada para evaluar el desempeño del nodo de procesamiento heterogéneo en el área de flujometría Doppler. Para entender el comportamiento y la información que ofrece la señal Doppler al ser procesada, se abarcan tres temas: ultrasonido Doppler, métodos de estimación espectral convencionales y métodos no convencionales. Asimismo, se describe el funcionamiento de los programas del sistema heterogéneo, así como los programas de un sistema homogéneo formado sólo por transputers. Estos programas sirven de base para establecer el tipo de desempeño logrado con un sistema de procesamiento heterogéneo. Por último se analizan los resultados obtenidos de los dos sistemas.

Capítulo 7. Presenta las conclusiones generales y comentarios sobre este trabajo de tesis.

Referencias

García Nocetti, F., Ruano, M. G., Fish, P. J., Fleming, P. J. Parallel Implementation of a Model-based Spectral Estimator for Doppler Blood Flow Instrumentation. Proceedings of the 8th International Parallel Symposium April 26-29 1994 Cancún, México. Págs. 810-814.

Ruano, M.G., García Nocetti, D.F., Fish, P., Flemeing, P., 1992, "Parallel Implementation of an AR Estimator for Real-Time Ultrasonic Blood Flow Instrumentation". International Conference on Parallel Computing and Transputers Applications PACTA 92, Barcelona, Spain, (Eds. Valero, M., Oñate, E. et al), IOS Press/CIMNE, 337-345.

2 Antecedentes teóricos

Introducción

El propósito de este capítulo es proporcionar los antecedentes y características generales sobre tres temas de suma importancia para el desarrollo de este trabajo de tesis. El primero de estos tres temas se refiere al procesamiento en paralelo, del cual se da una reseña de su avance tecnológico actual, se menciona la importancia de dichos sistemas, las distintas formas de clasificarlos, así como los lenguajes de programación existentes para desarrollar sistemas paralelos.

El siguiente tema se refiere a los sistemas en tiempo real: se habla sobre las características generales de un sistema en tiempo real, así como del uso de lenguajes de programación que permitan optimizar en lo referente a una respuesta más rápida del sistema.

El último tema a tratar es el procesamiento digital de señales. En esta parte se describen los elementos básicos que conforman un procesador digital, los tipos de señales que existen, la conversión digital-analógica (D/A) y analógica-digital (A/D) y finalmente, el análisis en frecuencia.

2.1 Procesamiento en paralelo

En el procesamiento en paralelo un sólo proceso (o aplicación) es separado en múltiples partes para poder ser ejecutadas simultáneamente en diversos CPUs. Hace apenas unas décadas, el procesamiento paralelo requería una programación especial y por ello, fue solamente utilizado en supercomputadoras y sistemas similares. Debido al avance tecnológico en cuanto a hardware y software de los sistemas digitales, los sistemas multiprocesadores actuales pueden efectuar procesamiento en paralelo de una manera más sencilla. En esta sección se describe este avance tecnológico, tres clasificaciones de arquitecturas en paralelo considerando a la de Flynn como la más importante, los métodos de interconexión y el intercambio de información entre múltiples

Aproximación de nuevos lenguajes

Los nuevos lenguajes concurrentes han sido desarrollados para soportar procesamiento en paralelo. Ejemplos de lenguajes son Pascal concurrente, Modula-2, CSP, Occam2 y Ada. A pesar de que estos lenguajes tienen aplicaciones orientadas a construcciones paralelas, sólo soportan una forma de paralelismo.

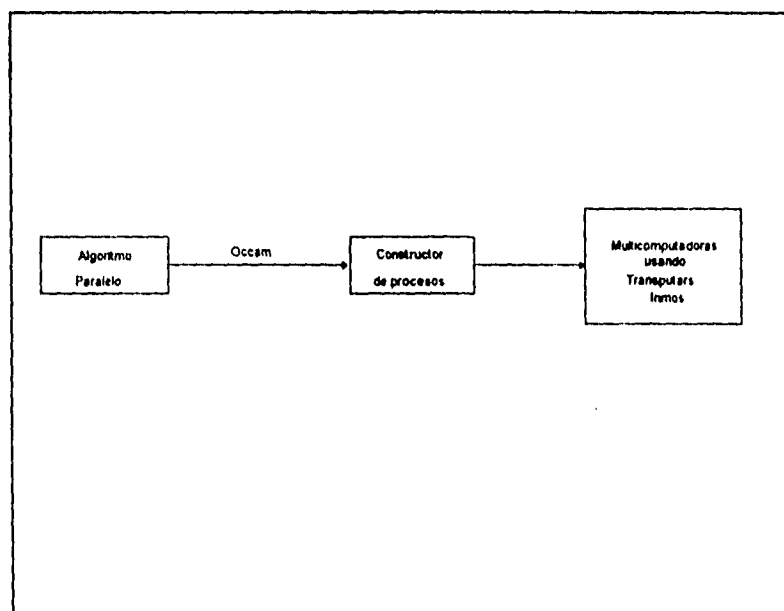


Figura 2.7 Aproximación de nuevos lenguajes.

2.1.10 Diseño de Algoritmos en Paralelo y Aplicaciones en Software

El diseño de algoritmos involucra varias fases de desarrollo. El problema físico debe ser modelado por una formulación matemática, tal como una ecuación diferencial o sistema algebraico. En muchos casos esas ecuaciones o sistemas son definidos en el dominio real como funciones en tiempo continuo. Para que la computadora resuelva esos sistemas, se emplean algunas formas de discretización y aproximaciones numéricas. Finalmente, se necesita dividir el algoritmo para mapearlas en arquitecturas paralelas o vectoriales.

Se han identificado varias aproximaciones en el diseño de algoritmos en paralelo. Dos de ellas son:

1. Convertir un algoritmo secuencial a una versión en paralelo. Este proceso no es sencillo, por lo cual debe realizarse un cuidadoso análisis del flujo de datos para encontrar todas las dependencias de datos/programa.
2. Codificar un nuevo algoritmo en paralelo.
3. La complejidad de la implementación del algoritmo depende del grado de paralelismo, de la granularidad, del gasto de comunicación y sincronización, y de las demandas de E/S entre otras.

Para arquitecturas estructuradas tales como arreglos, prismas, pirámides, hipercubos y árboles, el algoritmo es más sensible a la topología y a la capacidad de procesamiento.

2.2 Sistema en tiempo real

Hay muchas interpretaciones de la definición de un sistema en tiempo real, sin embargo, todas ellas coinciden en el concepto de tiempo de respuesta, o sea, el tiempo tomado por el sistema para generar una salida desde una entrada asociada. El diccionario de computación de Oxford da la siguiente definición de un sistema en tiempo real:

"Un sistema en tiempo real es aquel donde el tiempo utilizado para producir una salida se considera importante. Esto es porque la entrada está relacionada a algún estado dinámico en el mundo físico y la salida está relacionada al mismo estado. El intervalo de tiempo entre la entrada y la salida debe ser lo suficientemente pequeño para ser aceptado por el sistema".

Young (1982) define un sistema de tiempo real como:

"Cualquier actividad de procesamiento de información o sistema el cual tiene que responder al estímulo de una entrada generada externamente en un período finito y específico de tiempo".

Ambas definiciones cubren un amplio rango de actividades de cómputo. Por ejemplo, un sistema UNIX debe considerar el tiempo real en el cual un usuario teclea un comando esperando una respuesta en pocos segundos.

Un sistema en tiempo real no sólo depende del resultado lógico del cálculo efectuado sino también del tiempo en el cual se produce el resultado. Se distinguen dos tipos de sistemas en tiempo real:

1. Sistemas de tiempo real rígido (*Hard real-time systems*): son aquellos donde es importante que su respuesta ocurra dentro del tiempo especificado.
2. Sistemas de tiempo real suave (*Soft real-time systems*): son aquellos en los cuales el tiempo de respuesta es importante, pero el sistema podría aún funcionar correctamente si el tiempo especificado termina.

Un sistema de control de vuelo de una aeronave de combate es un sistema de tiempo real rígido ya que una falla en límite de tiempo de

respuesta puede ocasionar un desastre, en tanto que un sistema de adquisición de datos para una aplicación de control de procesos es suave si realiza un muestreo de la entrada de un sensor en intervalos regulares y se toleran retrasos ocasionales.

En un sistema de tiempo real, la computadora es usualmente la interfaz directa a algún equipo físico y está dedicada a monitorear o controlar la operación de este equipo. Dentro de estas aplicaciones, el rol de la computadora es la de ser el componente de procesamiento de información dentro de un sistema de ingeniería grande. Por esta razón, tales aplicaciones son conocidas como *embedded computer systems*.

Las características básicas de un sistema en tiempo real son las siguientes:

- Tamaño y complejidad
- Manipulación de números reales
- Extrema seguridad
- Control de concurrencia de los componentes separados del sistema.
- Control en tiempo real
- Intercomunicación con interfaces de hardware
- Eficiente implementación

El estado más importante en el desarrollo de cualquier sistema en tiempo real es la generación de un diseño que satisfaga una especificación de requerimientos. Los sistemas en tiempo real no varían mucho con respecto a otras aplicaciones, sin embargo, en términos más generales se presentan más problemas de diseño.

En general, los métodos de diseño para cualquier sistema involucran una serie de transformaciones desde los requerimientos iniciales hasta el código de ejecución. Algunos de los pasos típicos de estos métodos son:

- Especificación de requerimientos
- Diseño a alto nivel del sistema
- Diseño detallado
- Implementación
- Pruebas

La implementación requiere de un lenguaje de programación y los siguientes criterios son considerados como la base para el diseño de un lenguaje de tiempo real:

- Seguridad
- Legibilidad
- Flexibilidad
- Simplicidad
- Portabilidad
- Eficiencia

Hay tres clases de lenguajes de programación usados en sistemas de tiempo real: lenguajes de ensamblador, lenguajes secuenciales y lenguajes concurrentes de alto nivel.

El principal problema del lenguaje ensamblador es que está orientado más a las máquinas que a los problemas. Además, incrementa el costo y es muy difícil de modificar cuando hay errores o cuando se requiere mejorar el programa.

Los lenguajes secuenciales tienden a ser deficientes para el control y seguridad de los sistemas en tiempo real y como resultado de esta deficiencia requieren del soporte de un sistema operativo y código ensamblador.

Modula, Modula-2, PEARL, Mesa, CHILL y Ada son lenguajes de programación concurrente de alto nivel, que reúnen características para el desarrollo de sistemas en tiempo real. Otro lenguaje es Occam que comparándolo con Ada, CHILL y Mesa es un lenguaje más pequeño, pero proporciona un eficiente soporte para el desarrollo de sistemas largos y complejos, manejo de procedimientos no-recursivos y estructuras de control comunes, por lo que representa un papel muy importante en aplicaciones en tiempo real distribuidas.

Los lenguajes de programación concurrente más usados en los sistemas de tiempo real son: Ada, Modula-2 y Occam 2. Ada es preferido para sistemas de aplicación militar, Modula-2 lo es por su parecido a Pascal y Occam 2 porque es el más cercano a un lenguaje de propósito general que incluye comunicación secuencial de procesos (CSP). Occam 2 fue diseñado para multiprocesamiento, lo cual permite tener sistemas en el dominio del tiempo real.

2.3 Procesamiento de Señales Digitales

El procesamiento de señales digitales es una área de ciencias e ingeniería que se ha desarrollado rápidamente en los últimos 20 años. Este desarrollo es el resultado de los avances en la tecnología de computadoras digitales y en la fabricación de circuitos integrados. Hace dos décadas, las computadoras digitales y el hardware eran demasiado caros y de grandes dimensiones, por lo que su uso era limitado a cálculos científicos de propósito general fuera de línea (*off-line*) y a negocios. El rápido desarrollo de la tecnología de circuitos integrados inició con la integración a mediana escala (MSI), continuando con la integración a alta escala (LSI) y recientemente con la integración a muy alta escala (VLSI) de circuitos electrónicos, incrementando la velocidad y potencia, y reduciendo el tamaño de las computadoras digitales y el hardware de propósito especial. Estos circuitos digitales, relativamente más rápidos y baratos, han hecho posible construir sistemas digitales capaces de ejecutar funciones y tareas complejas de procesamiento digital de señales. El hardware para procesamiento digital de señales, permite tener operaciones programadas. A través del software es más fácil modificar funciones de procesamiento digital a ser ejecutados por el hardware, proporcionando un mayor grado de flexibilidad en el diseño de sistemas. Asimismo, se logra mayor precisión con hardware digital y software que con circuitos analógicos y procesamiento de señales analógicas.

2.3.1 Definición de señales y sistemas

Una señal es definida como una variable física que cambia con el tiempo, espacio o cualquier variable independiente. Matemáticamente una señal es una función de una o más variables independientes. Existen señales cuya relación funcional puede ser tan complicada como, por ejemplo, la de un segmento de voz, que puede ser definida como la suma de senoidales de diferentes frecuencias y amplitudes:

$$\sum_{i=1}^N A_i(t) \sin[2\pi F_i(t)t + \theta_i(t)]$$

Un sistema puede ser definido como un dispositivo físico que ejecuta una operación en una señal. Cuando una señal es pasada a través de un sistema, se dice que se ha procesado la señal. En general, el sistema es caracterizado por el tipo de operación que éste ejecute en la señal. Por ejemplo, si la operación es lineal, el sistema es llamado lineal, si la operación en la señal es no-lineal, el sistema es no-lineal. Tales operaciones son conocidas como *procesamiento de señales*.

En el procesamiento de señales digitales en una computadora digital, las operaciones ejecutadas en una señal son operaciones matemáticas especificadas por un programa. En este caso, el programa representa una implementación del sistema en software. El procesamiento digital en una señal puede ser realizado por hardware digital (circuitos lógicos), el cual asimismo es configurado para ejecutar las operaciones deseadas. Un sistema digital también puede ser realizado por una combinación de hardware y software digital, ejecutando cada uno un conjunto de operaciones.

2.3.2 Elementos básicos de un sistema de procesamiento digital de señales

La mayoría de las señales en ciencias e ingeniería son de naturaleza analógica, es decir, las señales son funciones de una variable continua, como el tiempo o el espacio y toman valores en un rango continuo. Tales señales pueden ser procesadas directamente por sistemas analógicos como filtros, analizadores de frecuencia o multiplicadores de frecuencia para el propósito de cambiar sus características o extraer alguna información. En este caso, se dice que la señal ha sido procesada directamente en forma analógica, donde la señal de entrada y salida son analógicas como se muestra en la *figura 2.8*.

El procesamiento de señales digitales proporciona un método alternativo para el procesamiento de señales analógicas de acuerdo a la *figura 2.9*:

Para realizar el procesamiento digital es necesario una interfaz entre la señal analógica y el procesador digital. Esta interfaz es llamada *convertidor analógico-digital (A/D)*. La salida de convertidor A/D es una señal digital que es una entrada apropiada para el procesador digital.

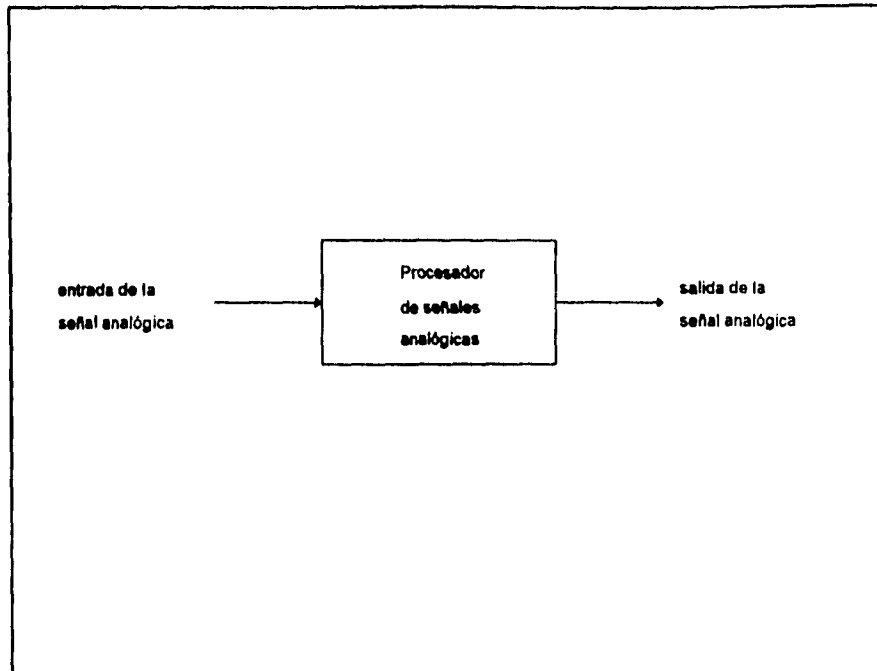


Figura 2.8 Procesamiento analógico de señales.

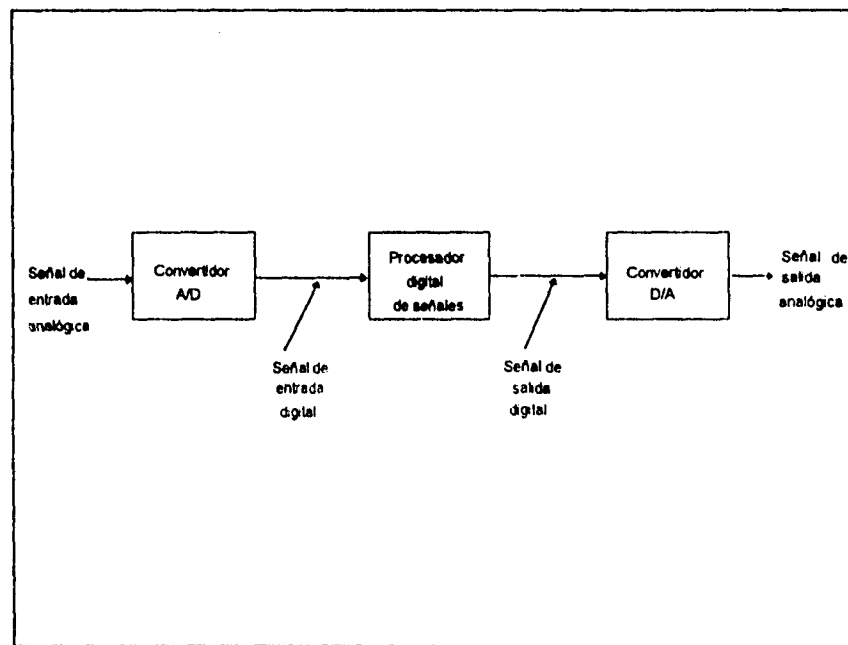


Figura 2.9 Diagrama de bloques de un sistema de procesamiento digital de señales.

El procesador de señales digitales puede ser una computadora digital programable o un microprocesador programados para ejecutar las operaciones deseadas en la señal de entrada. Este puede ser también un procesador digital alambrado (*hardwired*) que es configurado para ejecutar un conjunto de operaciones en la señal de entrada. Las máquinas programables proporcionan mayor flexibilidad a los cambios en las operaciones de procesamiento a través de cambios en software, donde las máquinas de propósito específico son difíciles de reconfigurar. En las aplicaciones donde la salida digital del procesador de señales digitales se tiene que proporcionar en forma analógica, se debe adicionar una interfaz llamada *convertidor digital-analógico (D/A)*.

2.3.3 Ventajas del procesamiento digital sobre el procesamiento analógico de señales

Existen varias ventajas. Primero, un sistema digital programable permite flexibilidad en la reconfiguración de las operaciones, simplemente modificando el programa. La reconfiguración en un sistema analógico implica el rediseño del hardware, probar y verificar que éste opere correctamente. El procesamiento de señales digitales proporciona mejor control de la precisión. La precisión se determina en el convertidor A/D y en el procesador de señales digitales, en términos de longitud de palabra, aritmética de punto fijo vs. punto flotante, primordialmente. Por su parte, la tolerancia de los componentes de circuitos analógicos hace extremadamente difícil controlar la precisión de los sistemas de procesamiento de señales analógicas.

Las señales digitales son fácilmente almacenadas en medios magnéticos (cintas o discos) sin deterioro o pérdida de la fidelidad de la señal más allá de lo que introduce el convertidor A/D. Como una consecuencia, las señales son transportables y pueden ser procesadas fuera de línea.

En algunos casos, una implementación digital de un sistema de procesamiento de señales es más barato que uno analógico. El bajo costo puede deberse al que el hardware digital es más barato, o quizá sea un resultado de la flexibilidad para las modificaciones, proporcionada por la implementación digital.

Las técnicas de procesamiento de señales digitales se aplican en procesamiento de voz, transmisión de señales en canales telefónicos, procesamiento y transmisión de imágenes, sismología y geografía, exploración de petróleo, en detección de explosiones nucleares, en el procesamiento de señales recibidas del espacio por mencionar sólo algunos.

La implementación digital también tiene sus limitaciones, una de ellas es la velocidad de operación de los convertidores A/D y de los procesadores de señales digitales, debido a que existen señales con un ancho de banda muy grande que requieren convertidores A/D con una rápida velocidad de muestreo y procesadores de señales digitales muy rápidos.

2.3.4 Clasificación de señales

Las señales pueden clasificarse de la siguiente forma [Strum, R., y Kirk, D., 1988]:

1. Tiempo continuo y amplitud continua. Estas son comúnmente llamadas *señales analógicas* y se pueden encontrar en cualquier fenómeno físico que asume cualquier rango continuo de valores tanto en tiempo como en amplitud.
2. Tiempo discreto y amplitud continua. Se conoce como *señal muestreada*. Los intervalos de tiempo son uniformemente espaciados, pero la señal puede tener cualquier nivel.
3. Tiempo discreto y amplitud discreta. Una señal de éstas es cuantizada en amplitud con intervalos de tiempo uniformes, por lo cual se dice que tanto la amplitud como el tiempo son cuantizados. Un convertidor analógico-digital (A/D) genera este tipo de señal la cual se le conoce como *señal digital*.
4. Tiempo continuo y amplitud continua con intervalos de tiempo uniformes. Esta es conocida como una señal *analógica muestreada (sampled-analog)* o *señal de datos muestreados (sampled-data)* y es característica de la salida de un S/H (*sample and hold*). La señal tiene un intervalo continuo de amplitudes que resultan de muestrear una señal analógica cada T segundos. Este valor muestreado es mantenido constante durante un período de tiempo por el convertidor A/D, que

normalmente sigue el S/H, para hacer la conversión analógica-digital. Durante los intervalos en los que la señal es cero, el S/H es la vía de la entrada analógica en preparación para la siguiente muestra.

5. Tiempo continuo y amplitud discreta con intervalos de tiempo uniformes. Es similar a la señal digital descrita en el punto 3, pero aquí, la señal mantiene su valor entre muestras. Es característica de la salida de un convertidor digital-analógico (D/A).

Las señales también pueden clasificarse de acuerdo al número de variables independientes que manejen o al número de señales que se manejan en la salida de un sistema.

Por ejemplo, la salida de un electrocardiógrafo que tiene tres electrodos (sensores) para monitorear la actividad eléctrica cardíaca (potencial) en tres diferentes posiciones del cuerpo. Si $s_k(t)$, $k = 1, 2, 3$, denota la señal eléctrica del k -ésimo sensor como una función del tiempo, el conjunto de 3 señales puede ser representado por un vector $S_3(t)$

$$S_3(t) = \begin{bmatrix} s_1(t) \\ s_2(t) \\ s_3(t) \end{bmatrix}$$

el cual es un vector de señales, denominándose a este tipo de señal como *señal multicanal*. En un electrocardiógrafo, electrocardiogramas (ECG) de 3 y 12 electrodos son usados en la práctica, los cuales resultan en señales de 3 y 12 canales.

Si la señal es una función de una sola variable independiente, la señal es llamada señal de *una dimensión*, de otra manera, una señal es llamada de *M-dimensiones* si su valor es una función de M variables independientes.

Otros tipos de señales

Una señal que puede describirse por una expresión matemática explícita, una tabla de datos o una regla bien definida es llamada *determinística*, es decir, que los valores pasados, presentes y futuros son conocidos.

En muchas aplicaciones prácticas, hay señales que no pueden ser descritas por fórmulas matemáticas o su descripción es muy complicada para ser usada prácticamente. Tales señales tienen un comportamiento impredecible en el tiempo. Este tipo de señales se denominan *aleatorias*. La salida de un generador de ruido, una señal sísmica y una señal de voz son ejemplos de señales aleatorias, como lo muestra la figura 2.10.

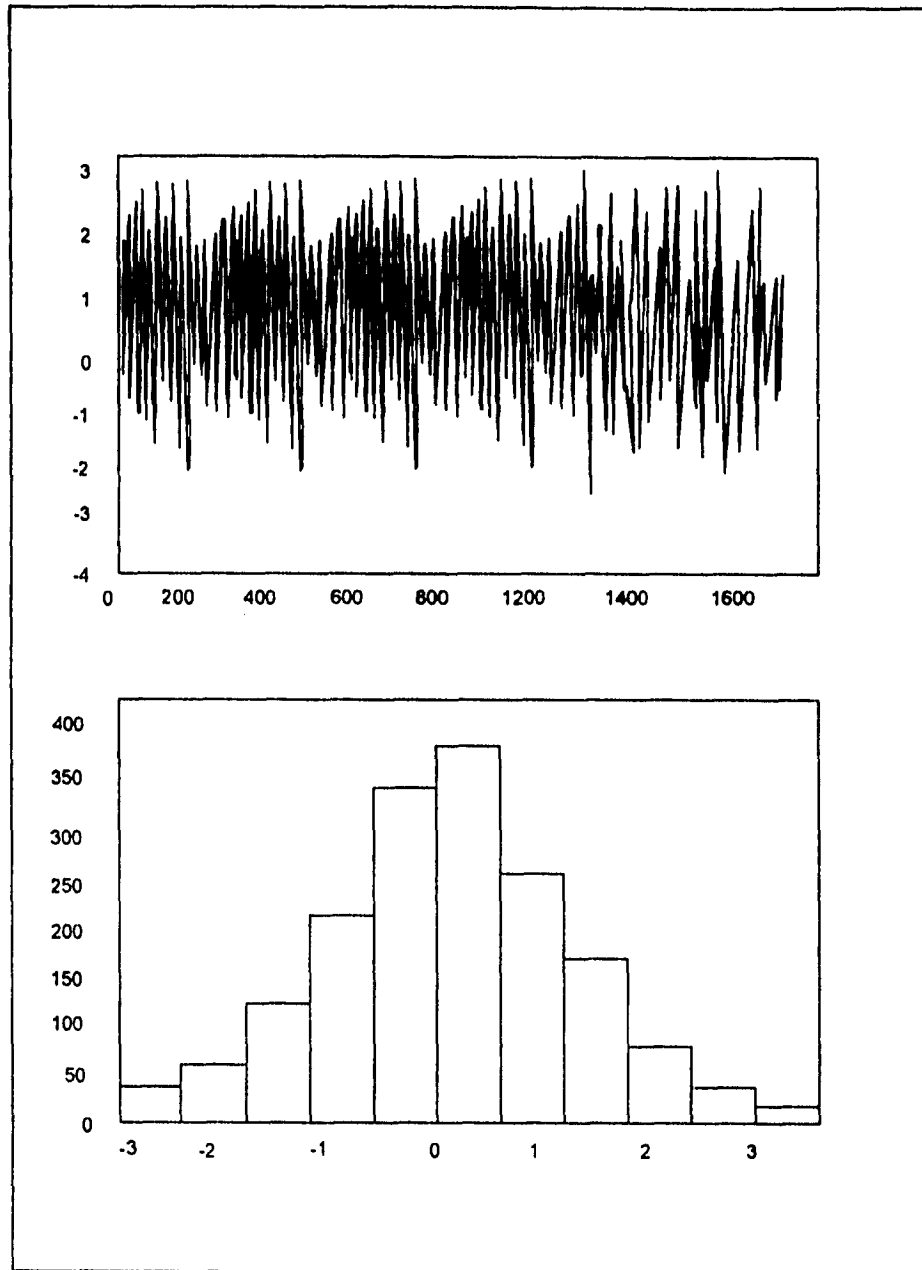


Figura 2.10 Señal aleatoria y su histograma.

2.3.5 Conversión analógica-digital y digital-analógica

Muchas señales de interés práctico como voz, señales biológicas, señales sísmicas, señales de ultrasonido y varias señales de comunicación tales como audio y señales de video, son analógicas. Para procesar señales analógicas a digitales, es necesario convertirlas a su forma digital, esto es convertirlas a una secuencia de números que tengan precisión finita. Este procedimiento se llama *conversión analógica a digital (A/D)* y los dispositivos son llamados *convertidores A/D (ADCs)*. La conversión A/D se puede definir en tres pasos:

- 1) Muestreo. Es la conversión de una señal en tiempo continuo a una señal en tiempo discreto al tomar muestras de la señal en tiempo continuo a instantes de tiempo discreto. Esto es, si $x_a(t)$ es la entrada al muestreador, la salida es $x_a(nT) = x(n)$, donde T es el *intervalo de muestreo*.
- 2) Cuantización. Es la conversión de los valores continuos de una señal a valores discretos. El valor de cada señal muestreada es representado por un valor seleccionado de un conjunto finito de valores posibles. La diferencia entre la muestra no cuantizada $x(n)$ y la salida cuantizada $x_q(n)$ es llamado error de cuantización.
- 3) Codificación. En el proceso de codificación, cada valor discreto $x_q(n)$ es representado por una secuencia binaria de b bits. Se asigna un número binario a cada nivel de cuantización. Si se tienen L niveles se necesitan por lo menos L números binarios diferentes. Con una longitud de palabra de b bits se pueden crear 2^b números binarios diferentes. Se tiene $2^b \geq L$, o su equivalente $b \geq \log_2 L$. La **figura 2.11** muestra las partes básicas de un convertidor analógico-digital (A/D).

En muchos casos de interés práctico (por ejemplo procesamiento de voz), es deseable convertir las señales procesadas digitalmente en forma analógica. El proceso de conversión de una señal digital a una señal analógica es conocido como *conversión digital analógica (D/A)*. Todos los convertidores D/A realizan alguna clase de interpolación, para unir los puntos de una señal digital, la precisión depende de la calidad de los procesos de conversión D/A. Las aproximaciones más utilizadas son la interpolación lineal (conexión de un par de muestras sucesivamente),

retén de orden cero o *aproximación staircase* y la interpolación cuadrada (acoplamiento cuadrático a través de 3 muestras sucesivas). Para señales con un contenido limitado de frecuencia (ancho de banda finito), el teorema de muestreo resulta ser la forma óptima de interpolación.

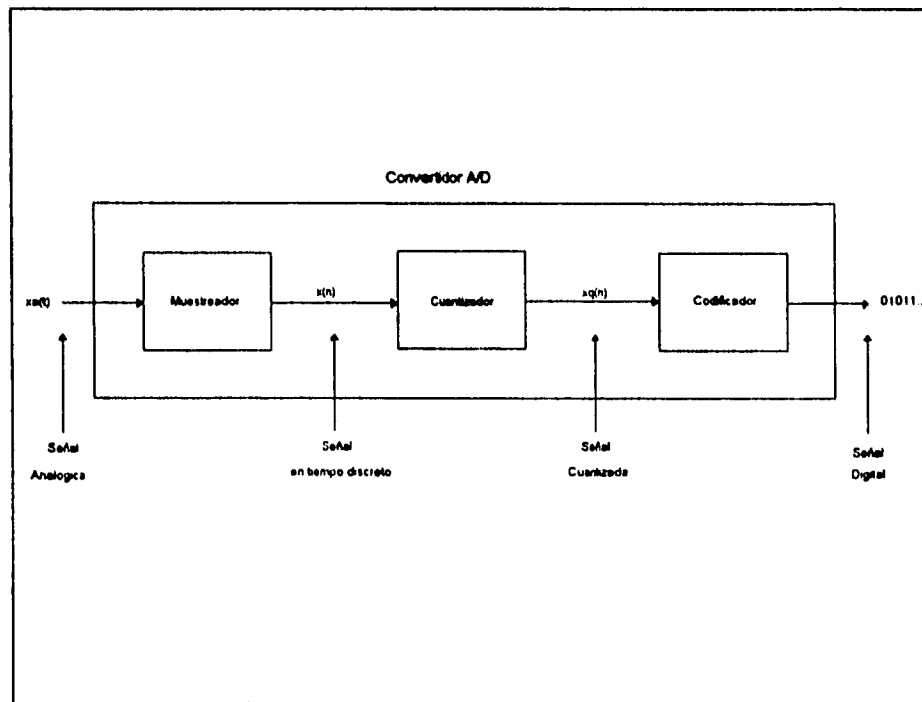


Figura 2.11 Parte básica de un convertidor analógico-digital (A/D).

Muestreo de señales analógicas

El tipo de muestreo más usado es el muestreo periódico o uniforme. La relación:

$$x(n) = x_a(nT) \quad -\infty < n < \infty$$

donde $x(n)$ es la señal en tiempo discreto obtenida al muestrear la señal analógica $x_a(t)$ cada T segundos. El intervalo de tiempo T entre muestras sucesivas es llamado *período de muestreo* o *intervalo de muestreo* y su recíproco $1/T = f_s$ es llamado *velocidad de muestreo* (muestras por segundo) o *frecuencia de muestreo* (Hertz).

Estas variables están relacionadas a través del período de muestreo T o, a través de la velocidad de muestreo $F_s = 1/T$, como

$$t = nT = \frac{n}{F_s}$$

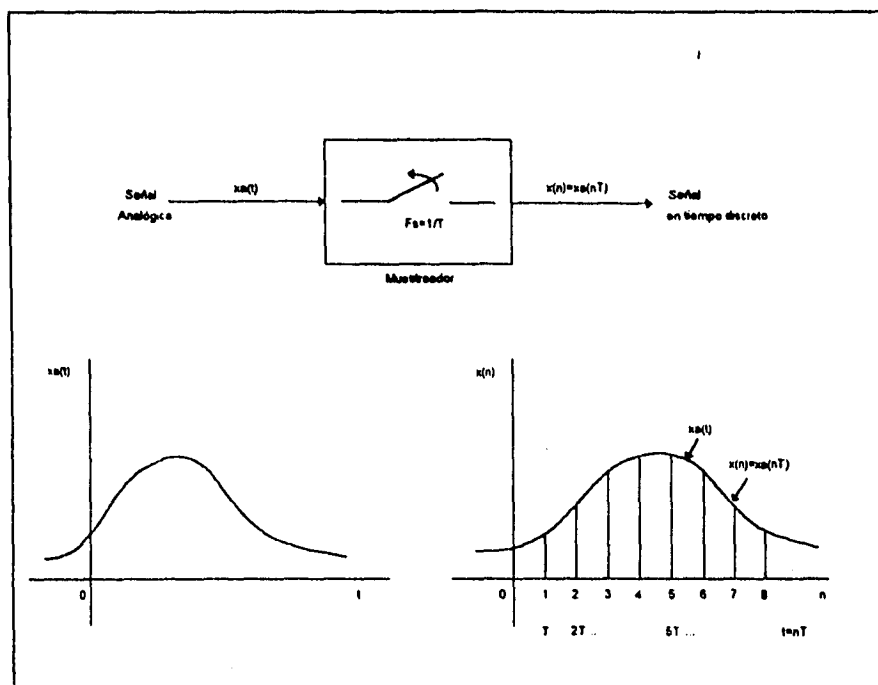


Figura 2.12 Muestreo periódico de una señal analógica.

Teorema de muestreo

El propósito del procesamiento de señales es extraer información detallada. Si se conoce la máxima frecuencia contenida en la señal se puede especificar la velocidad de muestreo necesaria para convertir la señal analógica a una señal digital.

Teorema de Muestreo. Si la frecuencia más alta contenida en una señal analógica $x_a(t)$ es $F_{max} = B$ y la señal es muestreada a una velocidad $F_s > 2F_{max} \equiv 2B$, entonces $x_a(t)$ puede ser recuperada totalmente de sus valores muestreados usando la función de interpolación:

$$g(t) = \frac{\sin 2\pi Bt}{2\pi Bt}$$

Por lo tanto $x_a(t)$ puede ser expresada como:

$$x_a(t) = \sum_{n=-\infty}^{\infty} x_a\left(\frac{n}{F_s}\right) g\left(t - \frac{n}{F_s}\right)$$

donde $x_a(n/F_s) = x_a(nT) \equiv x(t)$ son las muestras de $x_a(t)$.

La velocidad de muestreo $F_N = 2B = 2F_{max}$ es llamada velocidad de *Nyquist*.

2.3.6 Análisis de frecuencia de señales y sistemas

Las señales naturales tales como, sísmicas, biológicas y señales electromagnéticas, pueden ser caracterizadas por su contenido de frecuencia. Por ejemplo, en el caso de señales biológicas, el análisis de frecuencia es usado para diagnóstico de enfermedades. En señales sísmicas, el análisis de frecuencia se utiliza para explorar la estructura de la tierra y en búsqueda de petróleo. En el caso de señales electromagnéticas (por ejemplo el radar) el análisis de frecuencia proporciona información sobre la velocidad de un avión.

Muchas señales de interés práctico pueden ser descompuestas en una suma de componentes senoidales. Con una descomposición de este tipo, es posible representar a una señal en el dominio de la frecuencia. Para la clase de señales periódicas, una descomposición de este tipo se llama *series de Fourier*. Para las señales de energía finita, la descomposición se llama *transformada de Fourier*.

Herramientas para el análisis de frecuencia

De acuerdo a los diferentes tipos de señales que existen, las herramientas para el análisis de frecuencia se pueden resumir en las siguientes:

1. Series de Fourier para señales periódicas en tiempo continuo.
2. Transformada de Fourier para señales periódicas en tiempo continuo.
3. Series de Fourier para señales periódicas en tiempo discreto.
4. Transformada de Fourier para señales aperiódicas en tiempo discreto.

Transformada de Fourier de señales aperiódicas en tiempo discreto

El análisis de frecuencia de señales aperiódicas de energía finita en tiempo discreto involucra una Transformada de Fourier de la señal en el dominio del tiempo. La Transformada de Fourier de una señal de energía finita en tiempo discreto está definida como:

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}$$

Físicamente $X(\omega)$, representa el contenido de frecuencia de la señal $x(n)$, es decir $X(\omega)$ es una descomposición de $x(n)$ en sus componentes de frecuencia. La transformada de Fourier de una señal de energía finita en tiempo discreto se diferencia de una señal analógica de energía finita en que para señales continuas la transformada de Fourier y el espectro de la señal tiene un rango de frecuencia de $(-\infty, \infty)$, mientras que el rango de una señal discreta está sobre $(-\pi, \pi)$ o su equivalente $(0, 2\pi)$.

Existen dos características en el dominio del tiempo que determinan el tipo de espectro de la señal: si la variable tiempo es continua o discreta, y si la señal es periódica o aperiódica.

De lo anterior se tiene que:

- Las señales en tiempo continuo tienen un espectro aperiódico
- Las señales en tiempo discreto tienen un espectro periódico
- Las señales periódicas tienen un espectro discreto
- Las señales aperiódicas de energía finita tienen espectro continuo

El término de espectro de densidad de energía se usa para caracterizar señales aperiódicas de energía finita y el término de espectro de densidad de potencia para señales periódicas. Las señales periódicas son señales de potencia y las señales aperiódicas con energía finita son señales de energía.

Clasificación de señales en el dominio de frecuencia

En particular, si una señal de potencia tiene su espectro de densidad de potencia concentrado cerca de la frecuencia cero, se llama *señal de baja frecuencia*. De otra forma, si el espectro de densidad de potencia es concentrado en altas frecuencias, la señal es llamada *señal de alta frecuencia*. Una señal que tiene su espectro de densidad de potencia en un rango entre bajas y altas frecuencias es llamada *señal de frecuencia media o señal pasa banda*.

Es deseable expresar cuantitativamente el rango de frecuencias sobre las cuales el espectro de densidad de potencia está concentrado. Esta medida cuantitativa es llamada *ancho de banda* de la señal. Por ejemplo, si una señal en tiempo continuo tiene el 95% de su espectro de densidad de potencia concentrado en el rango $F_1 \leq F \leq F_2$ entonces el 95% del ancho de la señal es $F_2 - F_1$.

En el caso de una señal pasa-banda, el término banda-reducida es usada para describir a la señal si su ancho de banda $F_2 - F_1$ es más pequeña que la frecuencia media $(F_2 + F_1)/2$. De otra manera, la señal es llamada de banda ancha.

Una señal es limitada en banda si su espectro es cero fuera del rango de frecuencia $|F| \geq B$. Por ejemplo, una señal de energía finita en tiempo continuo $x(t)$ es de banda limitada si su transformada de Fourier $X(F) = 0$ para $|F| > B$. Una señal de energía finita en tiempo discreto $x(n)$ es llamada también de banda limitada si:

$$|X(w)| = 0, \text{ for } w_0 < |w| < \pi$$

Rangos de Frecuencia de algunas señales naturales

En el procesamiento de cualquier señal con el propósito de medir ciertos parámetros o extraer otro tipo de información, se requiere conocer el rango de frecuencias contenidas por la señal. Las siguientes tablas dan límites aproximados en el dominio de la frecuencia para señales biológicas y sísmicas.

Tabla 2.3a Rangos de frecuencia de algunas señales biológicas.

TIPO DE SEÑAL	RANGO DE FRECUENCIA (Hz)
Electroretinograma ^a	0-20
Electronystagmogram ^b	0-20
Neumonograma ^c	0-40
Electrocardiograma (ECG)	0-100
Electroencefalograma (EEG)	0-100
Electromiograma ^d	10-200
Sphygmomanogram ^e	0-200
Voz	100-400

NOTAS:

- ^a Registro gráfico de características de retina.
^b Registro gráfico del movimiento involuntario de los ojos.
^c Registro gráfico de actividad respiratoria.
^d Registro gráfico de acción muscular.
^e Registro de la presión de la sangre.

Tabla 2.3b Rangos de frecuencia de Señales Sísmicas.

TIPO DE SEÑAL	RANGO DE FRECUENCIA (Hz)
Ruido del viento	100-1000
Señales de exploración sísmica	10-100
Terremotos y explosiones nucleares	0.01-10
Ruido sísmico	0.1-10

Tabla 2.3a Rangos de frecuencia de algunas señales biológicas.

TIPO DE SEÑAL	RANGO DE FRECUENCIA (Hz)
Electroretinograma ^a	0-20
Electronystagmogram ^b	0-20
Neumonograma ^c	0-40
Electrocardiograma(ECG)	0-100
Electroencefalograma (EEG)	0-100
Electromiograma ^d	10-200
Sphygmomanogram ^e	0-200
Voz	100-400

NOTAS:

^a Registro gráfico de características de retina.^b Registro gráfico del movimiento involuntario de los ojos.^c Registro gráfico de actividad respiratoria.^d Registro gráfico de acción muscular.^e Registro de la presión de la sangre.

Tabla 2.3b Rangos de frecuencia de Señales Sísmicas.

TIPO DE SEÑAL	RANGO DE FRECUENCIA (Hz)
Ruido del viento	100-1000
Señales de exploración sísmica	10-100
Terremotos y explosiones nucleares	0.01-10
Ruido sísmico	0.1-10

Referencias

Burns, A., Wellings, A., 1989. Real-Time Systems and their Programming Languages. Ed. Addison-Wesley.

Carlini, U., Villano, U., 1991. Transputers and Parallel Architectures: message-passing distributed systems. Ed. Ellis Horwood Limited.

Harp, G., 1989. Transputer Applications. Ed. Pitman Publishing. Capitulo 1 y 3.

Hwang, K., Briggs, F., 1984. Computer Architecture and Parallel Processing. Ed. McGraw-Hill.

Hwang, K., Degroot, D., 1989. Parallel Processing for Supercomputers & Artificial Intelligence. Ed. McGraw-Hill.

Inmos, 1989. The Transputer Databook.

Inmos, 1990. ANSI C Toolset User Manual.

Irwin, G., Fleming, P., 1992. Transputers in Real-time Control. Ed. Research Studies Press LTD y Jonh Wiley & Sons Inc.

Proakis, J., Manolakis, D., 1992. Digital Processing. Principles, Algorithms, and Applications. Ed. John Griffin.

Strum, R., Kirk, D., 1988. First Principles of Discrete Systems and Digital Signal processing. Ed. Addison-Wesley Publishing Company.

3 El Transputer y el DSP

Introducción

El propósito de este capítulo es presentar las características generales del transputer de Inmos, así como la familia de procesadores de señales TMS320 de Texas Instruments.

La familia del transputer de Inmos combina componentes de procesamiento, memoria e interconexión en un sólo chip del tipo VLSI. Un sistema de procesamiento en paralelo puede ser construido por una colección de transputers, que operan en forma paralela y se comunican a través de canales de comunicación serial. Tales sistemas pueden ser diseñados y programados en Occam ó en otros lenguajes basados en comunicación de procesos.

Por su parte, un DSP (*Digital Signal Processor*) es un microprocesador dedicado a la tarea de procesamiento de señales digitales. Entre otras características, un DSP cuenta con un hardware específico para efectuar operaciones de multiplicación y acumulación de forma rápida, conversión A/D y D/A integrada, aritmética de punto fijo o flotante, etc. Esto permite que los DSPs ejecuten eficientemente funciones que manejan arreglos de datos, por ejemplo, en el cálculo de la transformada rápida de Fourier (FFT) o la función de Correlación.

La presentación de estos dispositivos obedece a la utilización de un procesador de señales en el diseño y construcción del nodo heterogéneo para una plataforma de procesamiento paralelo basada en transputers.

3.1 El transputer

3.1.1 La familia de procesadores del tipo transputer

El transputer es actualmente el único microprocesador que proporciona soporte a nivel de hardware y software para el procesamiento en paralelo.

La característica más importante del transputer, es la comunicación entre procesadores. Desde la perspectiva de hardware, el transputer tiene cuatro *links* seriales bidireccionales que pueden ser conectados a otros transputers, pudiendo ser explotado por el procesamiento en paralelo. Desde la perspectiva del software, la comunicación a otro transputer ocurre de la misma forma que la comunicación entre dos procesos en el mismo transputer, permitiendo que sea más fácil el diseño de programas en paralelo.

El diseño del transputer tuvo dos fuertes influencias: el procesamiento en paralelo y la tecnología RISC (*Reduced Instruction Set Computing*). Similarmente, el transputer tuvo un impacto relevante de la tecnología de los microcontroladores. La influencia del microcontrolador en el transputer se da cuando al generarse un error en un cálculo, genera su propio *shut down*. Otra característica es que el transputer es capaz de atender interrupciones externas rápidamente. Esta rapidez se debe a la pequeña cantidad de estados asociados con cada "proceso" o *thread* de ejecución.

Nota. El término *thread* se refiere a una secuencia de instrucciones que se están ejecutando. Si se sigue las instrucciones consecutivas de un programa que se está ejecutando, entonces se dice que se sigue el "*thread*" de ejecución para el programa. Algunos sistemas operativos, como MS-DOS, puede solamente soportar un sólo *thread* de ejecución de programa. Otros sistemas operativos como UNIX, pueden soportar múltiples *threads* de ejecución de programas, lo que implica que más de un programa puede ejecutarse al mismo tiempo [Roberts, J., 1992].

Por su parte, la influencia del procesamiento en paralelo dió al transputer los mecanismos de comunicación que habilitan a múltiples transputers a comportarse como una máquina de cálculo, en la cual cada componente individual del transputer pasa mensajes a otro.

La influencia de la tecnología RISC en el transputer, ha hecho que el conjunto de instrucciones sea reducido. Esto significa que, debido al gran número de características especiales en el transputer (canales de comunicación integrados, timers dentro del chip, multiprocesamiento a nivel ensamblador, etc.), el conjunto de instrucciones no es tan pequeño, pero cada instrucción está de alguna manera limitada, ya que no abarca mucho sobre las características del hardware. La ventaja que ofrece la tecnología RISC en el transputer es que mientras menos poderosas sean las instrucciones, su ejecución es más rápida.

En adición a estas características de software, el transputer tiene una interfaz de memoria externa programable, que requiere poca circuitería. La parte más costosa de un sistema basado en transputers es actualmente la memoria RAM.

El transputer y su lenguaje de programación Occam, han sido diseñados específicamente para ser usados en sistemas multiprocesadores. Cualquier número de transputers puede ser conectados en red para construir sistemas multiprocesadores. La eficiencia es escalable, dando un incremento que puede ser lineal a medida que aumenta el número de procesadores en el sistema, los cuales son conectados a través de links sin requerir de circuitos adicionales [Harp, G., 1989].

La familia de transputers de Inmos está formada por los siguientes procesadores de 16 y 32 bits: IMS T222, IMS T225, IMS T414, IMS T425, IMS T800, IMS T801 y el IMS T805 [Inmos, 1989].

El primer miembro de la familia de transputers de Inmos fue el IMS T414 introducido en 1985, y ha sido utilizado en un amplio rango de aplicaciones tales como simulación, robótica (control), síntesis en imágenes y procesamiento señales digitales.

Otro miembro de la familia de transputers Inmos es el IMS T800, el cual fue introducido en 1987. Este transputer puede incrementar la eficiencia de los sistemas, ya que cuenta con una unidad de punto flotante capaz de manejar más de 1.5 MFLOPS (millones de operaciones en punto flotante por segundo).

El T800 está disponible en versiones de 20, 25, ó 30 MHz, mientras que el T414 se encuentra disponible en 15 o 20 MHz. Las versiones de 20 MHz del

T414 y T800 proporcionan una eficiencia de 15 MIPS (millones de instrucciones por segundo), pero al compararse con otros microprocesadores convencionales su eficiencia es de 3 a 5 MIPS debido a que sus instrucciones son más primitivas.

El transputer T801 es un T800 con buses de direcciones y de datos separados, lo que permite mejorar los tiempos de acceso a memoria. Mientras que el T805 es una actualización del T800 con instrucciones adicionales para depuración de programas, por ejemplo instrucciones de punto de ruptura (*break points*) [Roberts, J., 1992].

La nueva generación de transputers es la familia T9000, que alcanzará 200 MIPS y 25 MFLOPS (a 50 MHz). El T9000 realiza una ejecución paralela superescalar de múltiples instrucciones (como la hace el i860 de Intel). El código compilado por transputers anteriores podrá ejecutarse eficientemente en el T9000 (10 veces más rápido que en un T800 a 20 MHz). El transputer T9000 es un sistema de comunicaciones "virtual" que tiene la función de rutear los mensajes pasándolos por un hardware muy rápido. Esta innovación promete retrasos en la comunicación de sólo pocos microsegundos en redes que contienen 1000 o más procesadores. En contraste, el ruteo de mensajes hecho en el T800 muestra retrasos de varios milisegundos. También muchas instrucciones aritméticas incrementan su velocidad en comparación con el T800. Por ejemplo, una multiplicación entera con el T9000 se tarda de 2 a 5 ciclos (comparada con los 38 ciclos del T800), y una multiplicación en punto flotante de 64 bits tarda 3 ciclos (comparando con los 27 del T800). [Pountain, D., 1991].

Actualmente, sólo existe una versión del transputer T9000 a 25 MHz y se espera su comercialización a gran escala durante el año de 1995, período durante el cual se espera una versión a 30 MHz.

3.1.2 Canales de comunicación punto a punto

La arquitectura del transputer simplifica el diseño de sistemas por el uso de canales de comunicación punto a punto. Cada miembro de la familia de transputers tiene uno o más canales estándar, los cuales pueden ser conectados a otro canal de otro transputer. Los canales de comunicación punto a punto tienen muchas ventajas sobre los buses de sistemas de múltiples procesadores:

1. No hay contención en los mecanismos de comunicación, a pesar del número de transputers en el sistema.
2. No hay desventaja en la capacidad de carga al aumentar el número de transputers en el sistema.
3. El ancho de banda de las comunicaciones no se satura cuando se incrementa el tamaño del sistema [Inmos, 1989].

3.1.3 Comunicación

Al proporcionar una comunicación sincronizada, cada mensaje debe ser reconocido. Consecuentemente un canal de comunicación requiere al menos una señal alambrada en cada dirección.

Un canal entre dos transputers se implementa conectando una interfaz de canal de un transputer a otra interfaz de canal de otro transputer, a lo largo del cual el dato es transmitido serialmente.

Cada línea de señal lleva datos e información de control. El protocolo del canal proporciona la comunicación sincronizada. Asimismo, permite la transmisión de una secuencia arbitraria de bytes entre transputers, aún cuando éstos tengan diferente longitud de palabra.

Cada mensaje es transmitido como una secuencia de un sólo byte, requiriendo únicamente la presencia de un buffer de un byte en el transputer receptor (esto para asegurar que la información no se pierda). Cada byte es transmitido como un bit de inicio de nivel alto, otro bit en nivel alto para indicar que es un dato (a diferencia de un reconocimiento), seguido por 8 bits de datos y un bit de paro de nivel bajo. Después de transmitir un byte de datos, el transmisor queda en espera de una palabra de reconocimiento de parte del elemento receptor. Este reconocimiento consiste de un bit de inicio alto, seguido por un bit en nivel bajo. El reconocimiento significa que un proceso fue capaz de recibir un byte de reconocimiento y que el canal de recepción es capaz de recibir otro byte. Así, el canal de transmisión reinicia el proceso de transmisión después de que ha sido recibido el reconocimiento para el último byte.

Un reconocimiento puede ser transmitido tan pronto como la recepción de un byte de datos inicie. Por lo cual la transmisión puede ser continua, sin retrasos entre los bytes de datos.

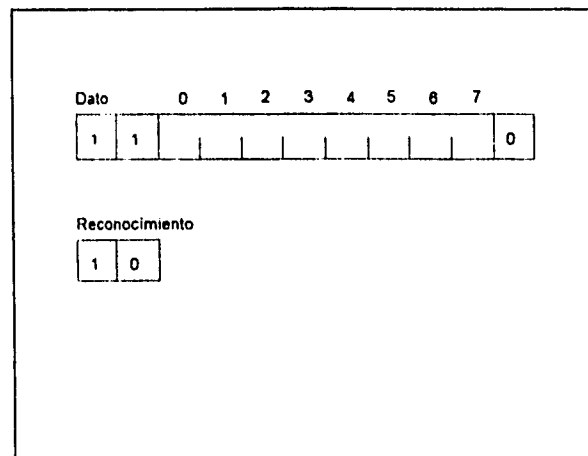


Figura 3.1 Protocolo del canal de comunicación.

Todos los transputers soportan una frecuencia de comunicación estándar de 10 Mbits/seg, sin importar la eficiencia del procesador. También pueden usarse a 5 ó 20 Mbits/seg.

La comunicación del canal no es susceptible a la fase del reloj. En consecuencia, la comunicación puede lograrse entre sistemas con un reloj independiente siempre que la frecuencia de comunicación sea la misma [Inmos, 1989].

3.1.4 Procesamiento en paralelo y el transputer

El procesamiento en paralelo es la habilidad de ejecutar múltiples cálculos simultáneamente. En sistemas de múltiples tareas (*multitasking*) esto puede aparecer como el cálculo de dos tareas al mismo tiempo, pero en realidad la computadora sólo realiza una tarea a la vez, cambiándose rápidamente entre tareas, de tal forma que parezca estar realizando varias tareas al mismo tiempo. Por su parte, el procesamiento en paralelo involucra máquinas de cálculo separadas físicamente, cada una realizando algún cálculo.

Existen dos modelos básicos de procesamiento en paralelo: **memoria compartida** y **memoria distribuída** (ver Capítulo 2).

En un modelo de memoria distribuída cada procesador tiene su propia memoria local, por lo cual no es común el intercambio de datos por memoria, así que los procesadores usan otro método de comunicación. En el transputer este método se refiere al uso de canales de comunicación seriales (*links*), que funcionan de manera similar a los puertos seriales en una computadora personal. Cada transputer tiene cuatro links bidireccionales que pueden ser conectados a otros links de otros transputers. Cada link proporciona un flujo de datos de un procesador a otro en el sistema. Esto permite conectar muchos transputers juntos en varias configuraciones, por lo cual la topología de la red de procesadores es importante en los sistemas de memoria distribuída.

Los sistemas de procesamiento se pueden clasificar en una de las siguientes categorías (ver Capítulo 2):

SIMD (*single instruction, multiple data stream*)
MISD (*multiple instruction, single data stream*)
MIMD (*multiple instruction, multiple data stream*)

El diseño del transputer es lo suficientemente flexible para configurarlo en cualquiera de las categorías antes mencionadas.

Existen arreglos de transputers que han sido usados como arquitecturas SIMD. Los transputers en este tipo de arquitecturas son conectados en una red de dos dimensiones y cada transputer ejecuta un bloque de programa más que una instrucción, con una sincronización al final de cada bloque. Se usa el término **programa simple datos múltiples** SPMD (*Single Program Multiple Data*) para definir este modo de operación, que tiene la ventaja de ser programado fácilmente [Harp, G., 1989].

Los sistemas con transputers son considerados como máquinas MIMD con memoria local y pueden ser utilizados en un amplio rango de aplicaciones.

Históricamente, las máquinas MIMD fueron limitadas a un número pequeño de procesadores, por las dificultades de programación y sincronización. Sin embargo, la combinación del transputer y de Occam

fue diseñada para salvar tales limitaciones y las facilidades de la arquitectura y de la comunicación del transputer se lograron gracias a la eficiente explotación de la tecnología VLSI.

Hay tres clases de aplicaciones para el transputer: aceleradores de tarjetas para PCs y workstations, sistemas en tiempo real (*embedded systems*) y computadoras de propósito general [Harp, G.,1989].

3.1.5 Redes de transputers [Harp, G.,1989]

Existen varias topologías o configuraciones de redes que hacen posible conectar varios transputers.

Topología de anillo

Es posible configurar cuatro transputers en forma de *anillo*. En esta configuración cada transputer se conecta a otros dos. La **figura 3.2** ilustra cuatro procesadores en una topología de anillo.

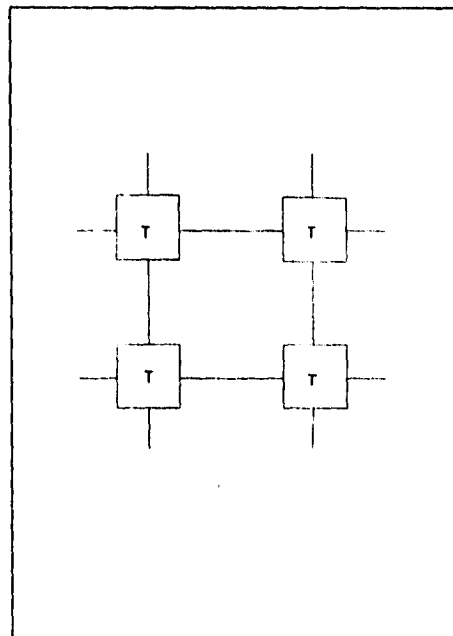


Figura 3.2 Configuración de anillo.

Cuando cada elemento de procesamiento en una red de procesadores es conectado a todos los otros procesadores en la red, se dice que la red

está completamente conectada (*fully-connected*). Una topología completamente conectada se muestra en la figura 3.3.

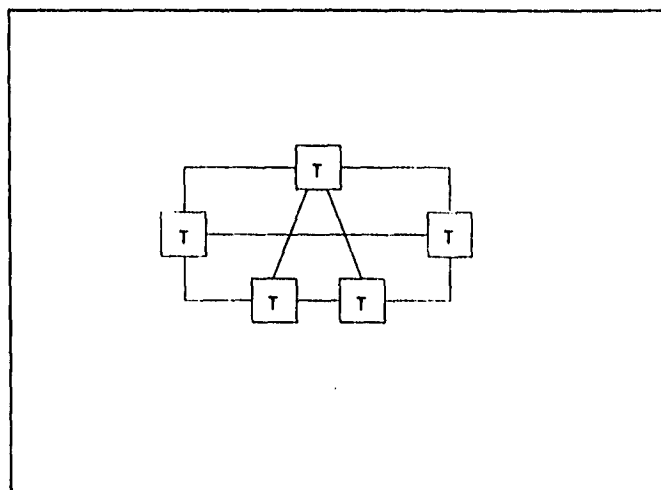


Figura 3.3 Topología completamente conectada.

Topología de árbol

El transputer no está limitado a topologías de anillo. La topología de una red es determinada por las necesidades del programa de aplicación. Para algunas aplicaciones, es deseable configurar una serie de transputers en una estructura de árbol como se muestra en la figura 3.4. La interconexión en forma de árbol es más apropiada para procesos jerárquicos tales como la reducción y el ordenamiento de datos.

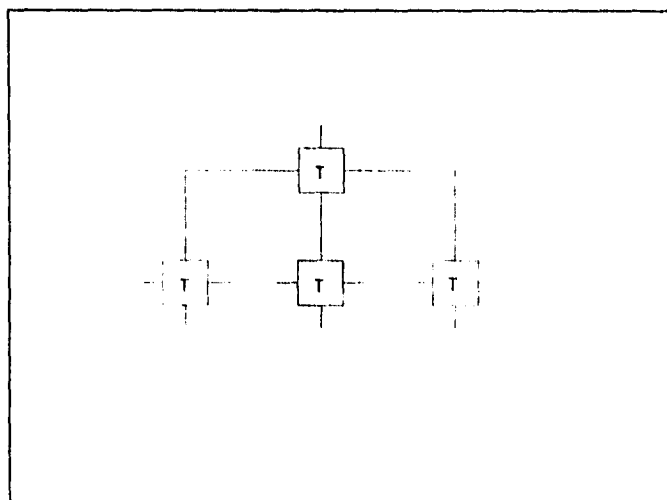


Figura 3.4 Configuración de árbol.

Topología pipeline

Existe otra topología denominada *pipeline*, donde el dato de entrada de un procesador es salida de otro. Los datos son transmitidos a varios procesadores, cada uno de los cuales ejecuta algún paso intermedio con el propósito de producir el resultado final, figura 3.5. Una topología pipeline con n estados tiene $2n$ links libres por lo que una topología de este tipo no hace uso óptimo de los elementos de comunicación de los transputers.

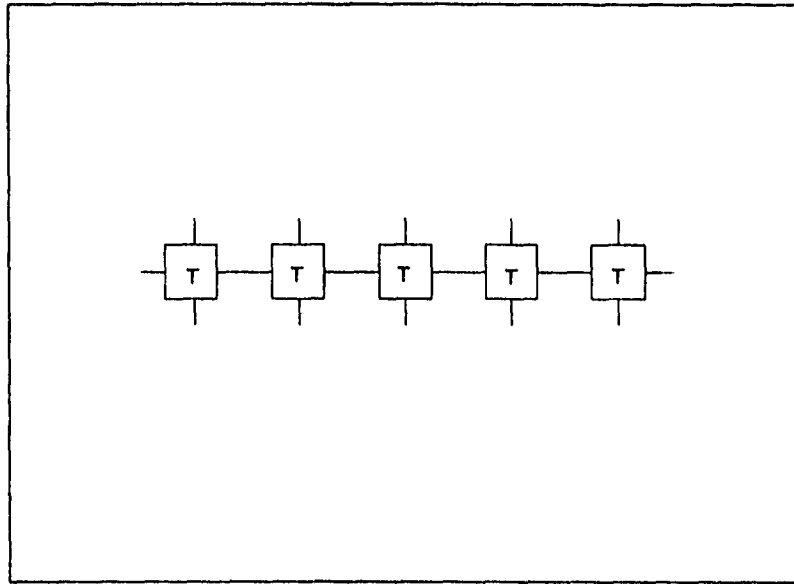


Figura 3.5 Configuración pipeline.

Topología de dos dimensiones

Un *arreglo de dos dimensiones* (2D) es otra topología apropiada para arreglos de datos estructurados, tales como imágenes o matrices y es aplicable a procesamiento de imágenes de bajo nivel. Cuando el arreglo de 2D es demasiado grande, se presentan dos problemas: el primero en transferencia de datos dentro y fuera del arreglo y el segundo por la comunicación irregular, aún y cuando se usen los cuatro links del transputer. Para un arreglo cuadrado de procesadores de lado n , el número de procesadores es n^2 con $4n$ links conectados externamente y con $2n(n-1)$ links conectados internamente.

Topologías reconfigurables

Para evitar problemas inherentes a redes fijas y proporcionar máxima flexibilidad, se puede usar una topología reconfigurable. El uso de redes reconfigurables permite reducir el tiempo de comunicación al reducir el número de procesadores intermedios, por los cuales los datos o mensajes tienen que ser transferidos. Una red reconfigurable puede ser usada en muchas aplicaciones, particularmente en sistemas multiprocesadores grandes. Una ventaja de este tipo de redes es su fácil programación, porque los sistemas desarrollados tienen la libertad de mapear algoritmos en un camino óptimo, haciendo fácil la conceptualización del software. Esta reconfiguración da adicionalmente un grado de tolerancia a fallas. En el caso de procesadores defectuosos, estos son cambiados o removidos de la red.

Por otro lado, si el tiempo de cálculo es más grande que el tiempo de comunicación entonces la comunicación puede ser insignificante y la configuración es irrelevante.

3.1.6 El transputer y sus lenguajes de programación

Inmos al introducir al transputer, proporcionó información del lenguaje ensamblador del transputer así como de otro lenguaje llamado Occam. Este último lenguaje es de un nivel más alto que el ensamblador, pero más bajo que los lenguajes de programación (tales como Pascal o C). Occam permite a un programador explotar el paralelismo y la comunicación del transputer. Para que los programadores puedan programar en los lenguajes, que les son más familiares, Inmos recientemente ha incluido los lenguajes C, C++, Fortran, Pascal, Modula-2 y Ada para transputers [Roberts, J., 1992].

Los transputers son utilizados donde se requiere explotar el procesamiento en paralelo, pudiéndose programar en lenguajes estándar y utilizar Occam para enlazar módulos escritos en otros lenguajes.

El lenguaje de programación Occam define a una aplicación como una colección de procesos que operan concurrentemente y se comunican a través de canales físicos o virtuales, es decir, un programa ejecutándose concurrentemente en un transputer es equivalente a un proceso Occam,

de esta forma, una red de transputers se describe como un programa Occam.

El componente del software es el proceso. Un sistema es diseñado en términos de un conjunto interconectado de procesos. Cada proceso puede ser visto como una unidad independiente de diseño. Su comunicación con otros procesos es a través de canales punto a punto. La comunicación entre procesos es sincronizada, lo cual evita el tener mecanismos de sincronización separados.

3.1.7 Lenguaje Occam

Los programas en Occam son contruídos por medio de tres procesos primitivos:

v	:=	e	asignación de e a la variable v
c	!	e	expresión de salida e al canal c
c	?	v	variable de entrada v desde el canal c

Los procesos primitivos son combinados para formar construcciones:

SEQ	secuencial
IF	condicional
WHILE	lazo de repetición
PAR	paralelo
ALT	alternas

Una construcción es por sí misma un proceso, y puede ser usado como una componente de otra construcción. Occam es un lenguaje estructurado con bloques jerárquicos.

Los programas secuenciales pueden ser expresados con variables y asignaciones, combinados con construcciones secuenciales, condicionales y de lazo.

Los programas concurrentes usan canales de entradas y salidas, combinados con construcciones paralelas y alternas.

3.1.8 Incluyendo código escrito en otros lenguajes

Para muchas aplicaciones es apropiado escribir el software en otro lenguaje de programación que no sea Occam. Puede ser que el software ya exista para la aplicación ó bien que un algoritmo sea más fácil de expresar en otro lenguaje. El modelo de programación Occam proporciona una simple base para incluir código escrito en otros lenguajes tales como C, Fortran y Pascal, considerados como lenguajes secuenciales. Un lenguaje secuencial puede ser considerado como un proceso secuencial Occam y compilado separadamente. La entrada y la salida de un programa puede ser implementada como en la comunicación de los canales en Occam. Cuando un programa está ejecutándose y comunicándose con canales en Occam, es indistinguible si el proceso fue escrito en Occam y puede ejecutarse en paralelo con otros procesos. Una vez que el proceso ha sido incluido, puede ser asignado, cargado y depurado en un procesador de la red, de la misma forma que un proceso en Occam.

3.2 El Procesador Digital de Señales (DSP)

3.2.1 Introducción a microprocesadores

Un microprocesador es una unidad de procesamiento central (CPU) que está típicamente dentro de un circuito integrado y que contiene:

- a) Unidad(es) lógica aritmética (ALU)
- b) Acumulador(res) (A)
- c) Apuntador(res) de stack (SP)
- d) Registro(s) índice (IR)
- e) Registro(s) de dirección de memoria (MAR)
- f) Contador(res) de programa (PC)
- g) Registro(s) de uso general (*scratchpad*)
- h) Unidad de control que regula la transferencia registro a registro de instrucciones y datos

Los microprocesadores difieren en el número de registros, y la arquitectura que los interconecta. También difieren en la longitud de su palabra (4,8,16, y 32 bits) que puede ser accesada en memoria y en los operandos en la CPU para operaciones aritméticas y lógicas. Estas diferencias determinan el conjunto de instrucciones de un microprocesador, el cual tiene una fuerte influencia en la longitud y complejidad de un programa para ejecutar una tarea particular.

Desde el punto de vista del diseño de hardware, la longitud del dato y del bus de direcciones así como el control de acceso al bus, juega un papel muy importante en el proceso de selección de un microprocesador. Algunos microprocesadores orientados al control de procesos tienen dispositivos de conversión A/D, D/A y timers que pueden ser controlados por software. Existen otros microprocesadores que contienen grandes bloques de memoria ROM (*Read Only Memory*) para almacenar programas y memoria RAM (*Random Access Memory*) para almacenamiento temporal. Algunos otros tienen también integrado una unidad de entrada/salida (E/S). A estas máquinas se les llama microcomputadoras ó microcontroladores. Esto reduce el número de circuitos integrados requeridos en una aplicación [Priemer, R.,1991].

3.2.2 Microprocesadores para procesamiento de señales

Por razones prácticas, tales como la operación en tiempo real, costo, espacio y limitaciones en potencia, puede no ser muy factible dedicar un microprocesador de propósito general a la tarea de procesamiento de señales. En cambio, un microprocesador con software y hardware necesario para procesamiento de señales puede resultar más apropiado. Actualmente existe este tipo de microprocesadores disponibles comercialmente. Estos tienen un conjunto de instrucciones que son lo suficientemente extensivas para programar algoritmos para procesamiento de señales y se les denomina DSPs (*Digital Signal Processors*). Sus circuitos integrados tienen características tales como hardware dedicado para operaciones rápidas de multiplicación y acumulación, conversión análogica-digital (A/D) y digital-análogica (D/A) integrada, lo que permite ejecutar tareas como: convolución, correlación, síntesis de señales y operaciones como la transformada rápida de Fourier (FFT).

Tabla 3.1 Comparación de microprocesadores DSP.

CARACTERÍSTICA	TI TMS32010	NEC μPD7720	AMI S2811	INTEL 2920
Tamaño de la palabra de datos (bits)	16	16	16	25
Tamaño del coeficiente (bits)	16	13	16	Variable
Acumulador	32	16	16	28
Saturación aritmética	Hardware	Software	Hardware	Hardware
Operaciones lógicas-booleanas	Si	Si	No	Si
Implementación de multiplicación	Hardware	Hardware	Hardware	Software
Precisión de la multiplicación (entrada X entrada = salida)	16x16=32	16x16=32	12x12=16	12x25=28
Tiempo de multiplicación (ns) (peor de los casos)	200	250	300	4800
E/S paralela (bits)	16	8	8	4E / 8S
Instrucción (bits)	16	23	17	24
Ciclo de instrucción (ns)	200	250	300	400
Niveles de subrutina	>50	4	1	Ninguna
Contador de iteración (loop)	Si	No	Si	No
Salto condicional	Si	Si	Si	No
Expansión de memoria externa	Si	No	No	No
Instrucción en ROM (bits)	1536x16	512x23	256x17	192x24
Coeficiente en ROM (bits)	No requerido	512x13	120x16	
Datos en RAM (bits)	144x16	128x16	128x16	40x25
función z^{-1}	Si	Si	Si	No
Tablas Look-up	Si	Si	Si	No
Encapsulado	40 pines	28 pines	28 pines	28 pines

Fuente: Priemer, R. (1991). *Introductory Signal Processing* (Advanced Series in Electrical and Computer Engineering). Ed. World Scientific Publishing.

Los primeros microprocesadores comerciales para procesamiento de señales fueron: el DSP Intel 2920, el DSP de Texas Instruments TMS32010, y el DSP AT&T DSP32, que fue el primero en realizar aritmética en punto flotante [Priemer, R.,1991].

Una característica que distingue a los DSPs de los otros microprocesadores, es su manejo rápido en la operación de multiplicación y acumulación, con una longitud de datos relativamente grande. Por ejemplo, el TMS32010 contiene hardware dedicado que ejecuta una multiplicación de 16 X 16 bits en 200 ns; mientras el microprocesador de Motorola 68000, que es un microprocesador de propósito general con una longitud de palabra de 16/32 bits que tiene una instrucción de multiplicación, ejecuta una multiplicación entera de 16 X 16 bits en 7 microsegundos con un reloj de 10 MHz. Con un reloj de 25 MHz, el DSP32 puede ejecutar una multiplicación en punto flotante en 250 ns. A través del progreso en la tecnología de circuitos integrados, el tiempo para ejecutar estas operaciones está continuamente reduciéndose.

Tabla 3.2 Comparación de dispositivos TMS320, microprocesadores y microcontroladores de propósito general.

TMS320 DSPs vs MICROPROCESADORES Y MICROCONTROLADORES DE PROPÓSITO GENERAL						
Característica	TMS320C14	TMS320C25	TMS320C30	TMS320C50	80C196	68020
Tiempo del ciclo de instrucción (MHz)	25.0	40.0	33.0	50.0	12.0	24.0
Multiplicación 16x16=32 (µs)	0.16	0.10	0.06	0.05	2.4	1.0
Filtro FIR TAP (µs)	0.32	0.10	0.06	0.05	2.8	1.1
Loop PID (µs)	2.3	1.4	0.84	0.75	27.0	5.1
Multiplicación de matriz (3x3)(3x1) (µs)	4.3	2.7	1.6	1.4	24.3	9.7
Filtro Kalman (3er orden) estacionario (µs)	15.3	7.8	4.7	3.9	N/A	N/A
Respuesta a Interrupción (µs)	0.32	0.3	0.24	0.50	2.7	1.4

Fuente: TMS320 Digital Signal Processor, Product Family Bulletin. Texas Instruments (1992).

Actualmente se siguen fabricando más DSPs con muy larga escala de integración (VLSI), realizando tareas más especializadas y complejas.

Recientemente AT&T, NEC Electronics, Motorola y Texas Instruments han desarrollado DSPs enfocados a la telefonía celular. El procesador FlashDSP1616 de AT&T es un *ROM-coded*, compatible con el DSP1616-x30. El Flash DSP1616 y el DSP1616-x30 tienen 2K de memoria RAM, dos puertos seriales de E/S, un puerto BIO y un puerto paralelo. Las versiones que existen del FlashDSP1616 son a 50 MIPS/5 Volts y 30 MIPS/ 3 Volts. NEC Electronics con el microPD7701x entra a la telefonía celular. Este procesador cumple con la división estándar de tiempo IS-54 de acceso múltiple. Motorola introduce mejoras a su familia de DSPs de propósito general 56002 de 24 bits, corriendo hasta 66 MHz y adicionando un timer de 24 bits. Texas Instruments introduce los procesadores basados en la familia de DSPs TMS320, el decodificador de video TMS320AV220 MPEG-1, el decodificador de audio TMS320AV120 MPEG y el TMS320AV420 (*National Television Standards Committee encoder*) [Hottinger, K., 1994].

También IBM tiene nuevos procesadores digitales de señales, el MDSP2020 y el MDSP2021, que operan a 3.3 volts con el doble de eficiencia en comparación con sus DSPs anteriores. Estos procesadores están basados en una arquitectura pipeline y en la tecnología de procesamiento de señales virtuales. Cuenta también con un bus interno para el manejo del tráfico de datos en memoria local [Reinhardt K., 1994].

Algunos DSPs de punto flotante disponibles actualmente son el DSP96002 de Motorola, el ADSP-21020 de Analog Devices [Reekie, J., 1994] y los procesadores TMS320C30, TMS320C31 y TMS320C40 de Texas Instruments.

La tendencia en la evolución de estos microprocesadores es incrementar la velocidad, menor consumo de potencia, mejor precisión aritmética y un conjunto de instrucciones más compatible con los requerimientos de algoritmos para procesamiento de señales digitales. Además de esto, a través del diseño asistido por computadora (CAD) y por la manufactura asistida por computadora (CAM), el tiempo requerido para fabricar un chip VLSI dando los requerimientos de los algoritmos está decrecentándose. Consecuentemente, se puede esperar la disponibilidad de incrementar hardware más diverso y especializado para ejecutar tareas de procesamiento de señales más complejas.

3.2.4 Familia TMS320 de Procesadores Digitales de Señales

3.2.4.1 Características del procesamiento digital de señales

El área del procesamiento de señales digitales DSP (*Digital Signal Processing*) abarca una amplia gama de aplicaciones, como son: filtrado digital, codificación de voz, procesamiento de imágenes, transformadas rápidas de Fourier y audio digital. Estas aplicaciones y el procesamiento digital de señales tienen las siguientes características:

- Algoritmos con cálculos matemáticos intensivos
- Operaciones en tiempo real
- Implementación de muestreo de datos
- Flexibilidad del sistema

3.2.4.2 Solución histórica del DSP

Durante las pasadas décadas, las máquinas de procesamiento de señales digitales han sufrido drásticas evoluciones. Grandes sistemas *mainframe* fueron inicialmente usadas para procesar señales en el dominio digital. Debido a las limitaciones de la tecnología, esto no fue realizado en tiempo real. Cuando el estado del arte avanzó, los arreglos de procesadores fueron adicionados a la tarea de procesamiento. Por su flexibilidad y su velocidad, los arreglos de procesadores llegan a ser la solución a la investigación a nivel de laboratorio y pudieron ser extendidos a aplicaciones finales. No obstante, la tecnología de circuitos integrados maduró, permitiendo el diseño de microprocesadores y microcomputadoras más rápidos. Como un resultado, muchas aplicaciones de procesamiento de señales digitales han migrado de los arreglos de procesadores a subsistemas con microprocesador (por ejemplo máquinas bit-slice), con soluciones en un sólo circuito integrado. Esta migración ha llevado a que el costo de las soluciones con DSP sea bajo, al punto de permitir el uso extendido de esta tecnología. El incremento de la eficiencia de estos circuitos integrados ha expandido las aplicaciones del DSP desde telecomunicaciones tradicionales hasta procesamiento de imágenes/gráficas, y procesamiento de audio.

Un avance en la tecnología DSP fue la creación de un procesador de señales digitales en un sólo chip, como los procesadores de la familia TMS320 de Texas Instruments [Papamichalis, P. 1990]. Estos procesadores dan al diseñador una solución utilizando un DSP, con una eficiencia realizable sólo por los arreglos de procesadores de hace pocos años. La figura 3.6 muestra de forma gráfica la familia TMS320, con el eje *y* que indica la eficiencia hipotética y en el eje *x*, la evolución de la tecnología del semiconductor en el procesamiento. El primer miembro de la familia, es el TMS32010, puesto en el mercado en 1982. Este dió al diseñador de sistemas el primer microcomputador capaz de ejecutar 5 millones de operaciones por segundo (5 MIPS), incluyendo funciones de multiplicación y suma. Actualmente existen otros procesadores que derivan del TMS32010 dentro de la primera generación de la familia TMS320. Algunos de estos procesadores son TMS320C10, TMS320C15, y TMS32017. La segunda generación de dispositivos incluye el TMS32020 y el TMS320C25. El TMS320C25 puede ejecutar 10 MIPS, además tiene espacio para memoria expandida, operación combinada de multiplicación/acumulación en un sólo ciclo y funciones de E/S que han dado al TMS320C25 una eficiencia de 2 a 4 veces mayor en comparación con sus predecesores. La tercera generación de procesadores de la familia TMS320, el TMS320C30, tiene una velocidad de cómputo de 33 millones de operaciones en punto flotante por segundo (33 MFLOPS). La eficiencia en velocidad, así como la cantidad de información procesada en una unidad de tiempo (*throughput*) y la precisión de uno de estos procesadores, ha excedido por mucho a los procesadores de señales digitales actuales y ha alcanzado el nivel de una supercomputadora.

La cuarta generación la forman los procesadores TMS320C4x, los cuales forman parte de la primera versión de procesadores digitales de señales para procesamiento en paralelo.

La quinta generación formada por los DSPs TMS320C5x, son la generación de punto fijo más nueva de la familia TMS320 de procesadores digitales de señales. El TMS320C50 y el TMS320C51 son los primeros dispositivos de esta generación. Su unidad central de procesamiento (CPU) está basado en el CPU del TMS320C25, con mejoras en su arquitectura. Ambos dispositivos de la generación TMS320C5x ejecutan más de 28 MIPS y su código fuente es compatible con todos los procesadores de la primera y segunda generación [TMS320C5x User's Guide, 1991].

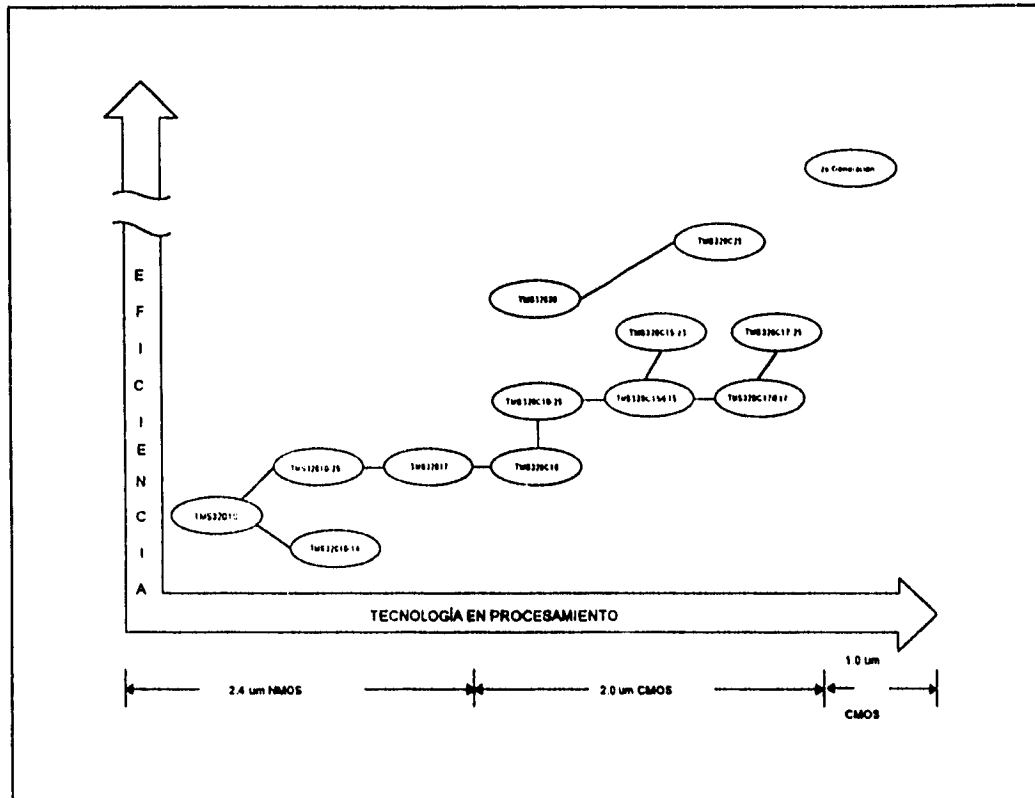


Figura 3.6 Familia de procesadores digitales de señales TMS320.

En la misma figura 3.6, se puede observar que los miembros de la misma generación como el TMS320C10, TMS320C15 y TMS320C17 son compatibles en cuanto al código objeto del ensamblador. Los dispositivos a través de las generaciones, tales como el TMS320C10 y el TMS320C25, son compatibles en cuanto al código fuente del ensamblador.

Desde la introducción del TMS32010, la tecnología del semiconductor para procesamiento ha evolucionado desde 3- μm NMOS a 2- μm CMOS y a 1- μm CMOS. Las generaciones de procesadores TMS320 han tomado la misma evolución en la tecnología del procesamiento. El consumo bajo de potencia, la alta eficiencia y la integración de circuitos de alta densidad son algunos de los beneficios directos de la evolución del semiconductor para procesamiento.

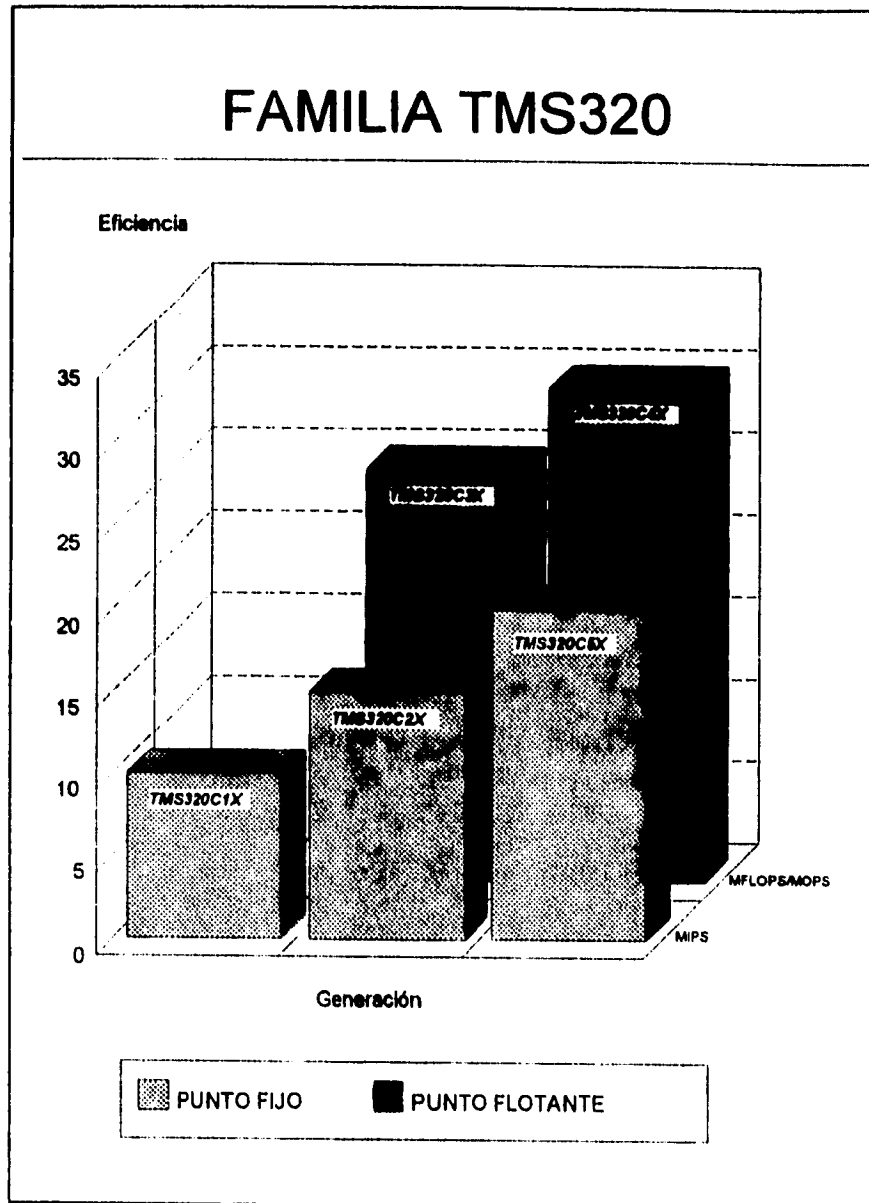


Figura 3.7 Las 5 generaciones de la familia de procesadores digitales de señales TMS320.

Los bloques que integran a un DSP tales como el CPU, memoria RAM, memoria ROM, configuraciones de E/S y velocidad del procesador, han sido diseñados como módulos individuales y pueden ser reorganizados o combinados con otros para cubrir las necesidades específicas de una

aplicación. Además de esto, como las reglas de las capas de los circuitos integrados están cambiando a reducir más la geometría (por ejemplo de 2 μm rápidamente bajó a 1 μm), no sólo los dispositivos TMS320 disminuyen su tamaño, sino que también múltiples CPUs podrán ser incorporados en el mismo dispositivo, junto con aplicaciones específicas de E/S para lograr integrar soluciones de bajo costo [Papamichalis, P., 1990].

3.2.4.3 Arquitectura básica del TMS320

Como se mencionó anteriormente, un procesador de señales digitales realiza operaciones aritméticas rápidas y maneja gran cantidad de información en los algoritmos matemáticos intensivos en tiempo real. Para llevar a cabo esto, la familia TMS320 se basa en los siguientes conceptos:

- Arquitectura Harvard
- Pipeline extensivo
- Hardware dedicado para multiplicación
- Instrucciones DSP especiales
- Ciclo rápido de instrucción

Estos conceptos fueron diseñados en los procesadores de señales digitales TMS320 para manejar una vasta cantidad de características del DSP y permitir que más operaciones puedan ser ejecutadas en un sólo ciclo de instrucción. Además, los procesadores son dispositivos programables, proporcionando flexibilidad y fácil manejo como los microprocesadores de propósito general.

Arquitectura Harvard

El TMS320 utiliza una arquitectura Harvard modificada por velocidad y flexibilidad. En una arquitectura Harvard, la memoria de programa y de datos esta separada en dos espacios, permitiendo un traslape completo en las fases de carga (*fetch*) y de ejecución de la instrucción. La modificación en la familia TMS320 de la arquitectura Harvard, permite la transferencia entre los espacios de programa y de memoria, lo cual incrementa la flexibilidad del sistema. Esta modificación elimina la necesidad de tener una memoria ROM separada y maximiza la potencia de procesamiento manteniendo las dos estructuras de bus separadas (programa y datos) para su ejecución a la máxima velocidad.

Pipeline extensivo

Conjuntamente con la arquitectura Harvard, el pipeline es usado extensivamente para reducir el tiempo del ciclo de instrucción al mínimo y para incrementar la cantidad de información procesada en una unidad de tiempo (*throughput*) del procesador. El pipeline puede ser de dos a cuatro niveles, dependiendo de cuál procesador es usado. La arquitectura de la familia TMS320 usa dos niveles de pipeline para la primera generación, tres niveles para la segunda generación y cuatro para los procesadores de la tercera generación. Esto significa que el procesamiento es de dos a cuatro instrucciones en paralelo y que cada instrucción está en una diferente etapa en su ejecución. La **figura 3.8** muestra un ejemplo de una operación pipeline de tres niveles.

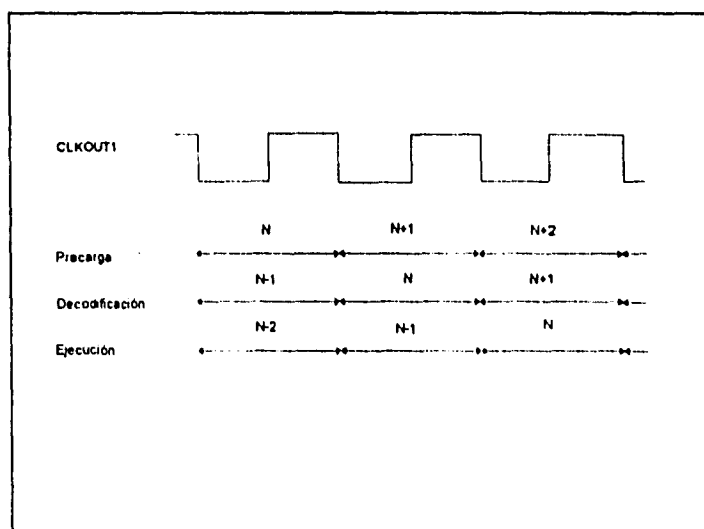


Figura 3.8 Operación pipeline de 3 niveles.

En la operación pipeline, las operaciones de precarga (*prefetch*), decodificación y ejecución puede ser manejadas independientemente. De esta manera, la ejecución de instrucciones es traslapada. Durante cualquier ciclo de instrucción, tres diferentes instrucciones son activadas, cada una con un estado diferente de terminación. Por ejemplo, cuando la instrucción N está siendo precargada, la instrucción previa (N-1) está siendo decodificada, y la instrucción previa (N-2) está siendo ejecutada. El pipeline es transparente al usuario.

Hardware dedicado para multiplicación

En microprocesadores de propósito general, la instrucción de multiplicación está construida por una serie de sumas, por consiguiente toma muchos ciclos de instrucción. En comparación con los dispositivos DSP, la multiplicación es dedicada. En la familia TMS320, una instrucción de un sólo ciclo es el resultado del hardware dedicado para la multiplicación.

Instrucciones DSP especiales

Otra característica de los dispositivos DSP es el uso de instrucciones especiales, como por ejemplo la instrucción DMOV (*data move*), cuya función es dar un retraso o mover un dato. Otra instrucción especial en el TMS320 es la instrucción LTD. Esta ejecuta las instrucciones LT, DMOV y APAC en un sólo ciclo. En la segunda generación del TMS320, el TMS320C25 tiene dos instrucciones especiales más (las instrucciones RPT y MACD) que reducen el número de ciclos.

Ciclo rápido de instrucción

El procesamiento en tiempo real es logrado por la velocidad de procesamiento en la ejecución de instrucciones. Las características antes mencionadas junto con la optimización del diseño de circuitos integrados en cuanto a velocidad, da a los dispositivos DSP un tiempo de ciclo de instrucción menor a 200 ns. Los tiempos de ciclo de instrucción para la familia TMS320 se muestran en la **Tabla 3.3**.

Tabla 3.3 Tiempos de ciclo de instrucción de la familia TMS320.

DISPOSITIVO	CICLO DE INSTRUCCION (ns)
TMS320C10*	160-200
TMS32020	160-200
TMS320C25	100-125
TMS320C3X	60-75
TMS320C4X	40-50
TMS320C5X	35-50

*El tiempo de ciclo de instrucción aplica a todos los procesadores de la primera generación.

Estos tiempos han hecho a la familia de procesadores TMS320 idónea para aplicaciones DSP en tiempo real.

En la **figura 3.9** se muestra cuantos ciclos de instrucción por muestra pueden obtenerse por varias generaciones de la familia TMS320 para distintas aplicaciones en tiempo real.

De la figura 3.9 se puede apreciar que varios ciclos de instrucción están disponibles para procesar la señal o para generar comandos para aplicaciones de control en tiempo real. Por lo tanto, para aplicaciones simples de control, los microprocesadores de propósito general resultan ser más adecuados. Sin embargo, para aplicaciones de control con mayor número de cálculos intensivos, tales como robótica y control adaptable, los procesadores de señales digitales son más apropiados.

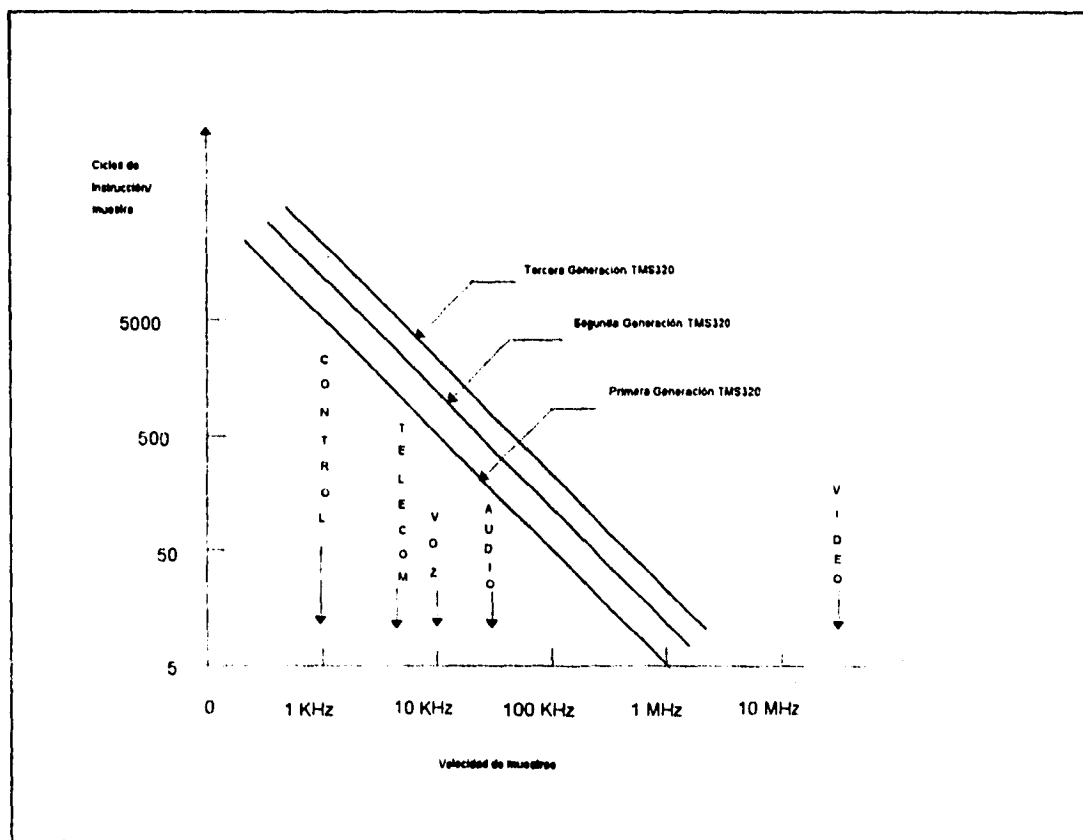


Figura 3.9 Ciclos de instrucción/muestra vs velocidad de muestreo de la familia TMS320.

El número de ciclos de instrucción se reduce cuando se incrementa la velocidad de muestreo de 8 KHz para aplicaciones de telecomunicaciones a 40-48 KHz para procesamiento de audio. Para aplicaciones de mayor velocidad de muestreo tales como el procesamiento de video/imagen, los procesadores de señales digitales disponibles actualmente no son capaces de procesar datos de video en tiempo real. Por consiguiente, para este tipo de aplicaciones se requieren múltiples procesadores de señales digitales y *buffers*. También de la figura 3.9, se observa que para aplicaciones de baja velocidad, tales como control, la primera generación TMS320 proporciona el mejor costo/eficiencia que para otros procesadores de otras marcas. Para aplicaciones de velocidad de muestreo más alta como el procesamiento de video/imagen, la segunda y tercera generación del TMS320 resultan ser las más apropiadas por su capacidad de multiprocesamiento y su alta capacidad de procesamiento (*throughput*) [Papamichalis, 1990].

En aplicaciones tradicionales como telecomunicaciones, comunicación de datos y procesamiento de audio, los DSPs ejecutan funciones que manipulan arreglos de datos, como la transformada rápida de Fourier (FFT) y el análisis espectral. El avance de la tecnología de DSPs y su bajo costo permite mirar hacia aplicaciones no tradicionales como manejadores de discos, industria automotriz y multimedia. La tabla 3.4 muestra las distintas aplicaciones para los DSPs.

La familia TMS320 posee procesadores DSP de punto fijo y de punto flotante. Los DSPs de punto fijo alcanzan hasta 30 MIPS, mientras los de punto flotante ejecutan hasta 275 MOPS/50 MFLOPS. Los DSPs TMS320 están diseñados para ejecutar una instrucción en un sólo ciclo de reloj. La familia de DSP's TMS320 proporciona una amplia variedad de procesadores que varían en costo/eficiencia. Desde el año 1982 en el que Texas Instruments introdujo el primer DSP, la familia a crecido a más de 25 procesadores compatibles de punto fijo y punto flotante. En la tabla 3.5 se muestran las características de los procesadores que integran las 5 generaciones de DSPs de la familia de Texas Instruments.

Tabla 3.4 Aplicaciones para DSP.

APLICACIONES PARA DSP	
TELECOMUNICACIONES	
Telefonos celulares e inalámbricos	Codificador y decodificador DTMF
Modems de 1200 a 19200 bps	Encriptamiento de Datos
Modems Facsimile	Teléfonos speaker
Comunicadores personales	Interpolación de voz digital
Transcoders/Digital PBXs	Cambiador de paquetes X.25
Comunicaciones en el espacio espectral	Videoconferencias
Multiplexado de canales	Líneas de repetidores
Ecuilibradores adaptativos	ISDN
CONSUMIDOR	
Máquinas de respuesta de estado sólido	Herramientas de potencia
Sintetizadores musicales	Audio/TV digital
Detectores de radar	Juguetes educativos
INDUSTRIAL	
Control numérico y robótica	Monitores de potencia
Seguridad de acceso	Control de motores
INSTRUMENTACIÓN	
Análisis espectral	Análisis transitorio
Generación de funciones	Filtrado digital
Combinación de patrones	lazos phase-locked
Procesamiento sísmico	
MILITAR	
Seguridad en comunicaciones	Procesamiento de radar/imágen/sonar
Modems RF	Teledirección de misiles y navegación
AUTOMOTRIZ	
Control del motor	Control de conducción adaptivo
Análisis de vibración	Posicionamiento global/navegación
Cancelación de ruido/amortiguadores electrónicos	Comandos por voz
Frenos antiderrapantes	Radio digital
COMPUTACIÓN	
Graficas en 3-D	Reconocimiento óptico de caracteres
Multimedia	Redes neuronales
Manejadores de disco duro	Arreglos de procesadores de alta velocidad
Impresoras Láser	Imágenes

Fuente: TMS320 Digital Signal Processor, Product Family Bulletin. Texas Instruments (1992).

Tabla 3.5 Características de procesadores de la familia de DSPs Texas Instruments.

Tipo de Dato	Procesador	MEMORIA (palabras)				E/S				Timer dentro del chip	Ciclo de Instrucción (ns)
		Dentro del chip		Fuera de Chip		Serial	Paralelo	DMA	Com		
		RAM	ROM	EPROM	Datos / programa						
Punto Fijo	TMS320C10/	144	1.5K	-	-14K	-	8x16	-	-	-	200
	TMS320C10-14	144	1.5K	-	-14K	-	8X16	-	-	-	280
	TMS320C10-25	144	1.5K	-	-14K	-	8X16	-	-	-	160
	TMS320C14	256	4K	-	-14K	1	7X16	-	-	4	160
	TMS320E14	256	-	4K	-14K	1	7X16	-	-	4	160
	TMS320P14	256	-	4K	-14K	1	7X16	-	-	4	160
	TMS320C15/	256	4K	-	-14K	-	8X16	-	-	-	200
	TMS320C15-25/	256	4K	-	-14K	-	8X16	-	-	-	160
	TMS320E15/	256	-	4K	-14K	-	8X16	-	-	-	200
	TMS320E15-25	256	-	4K	-14K	-	8X16	-	-	-	160
	TMS320LC15#	256	4K	-	-14K	-	8X16	-	-	-	200
	TMS320P15	256	-	4K	-14K	-	8X16	-	-	-	200
	TMS320C16	256	8K	-	-164K	-	8X16	-	-	-	114
	TMS320C17	256	4K	-	-14K	2	6X16	-	-	1	200
	TMS320E17	256	-	4K	-14K	2	6X16	-	-	1	200
	TMS320LC17#	256	4K	-	-14K	2	6X16	-	-	1	200
	TMS320P17	256	-	4K	-14K	2	6X16	-	-	1	200
	TMS320C25/	544	4K	-	64K/64K	1	16X16	Ext	-	1	100
	TMS320C25-33	544	4K	-	64K/64K	1	16X16	Ext	-	1	120
	TMS320C25-50/	544	4K	-	64K/64K	1	16X16	Ext	-	1	80
TMS320E25/	544	-	4K	64K/64K	1	16X16	Ext	-	1	100	
TMS320C26	1.5K	256ε	-	64K/64K	1	16X16	Ext	-	1	100	
TMS320C50/	10K	2K	-	-	2	64X16	Ext	-	1	50	
TMS320C50-57	10K	2K	-	-	2	64X16	Ext	-	1	35	
TMS320C51/	2K	8K	-	-	2	64X16	Ext	-	1	50	
TMS320C51-57	2K	8K	-	-	2	64X16	Ext	-	1	35	
Punto Flotante	TMS320C30/	2K	4K	-	16M•	2	16MX32ε	Int/Ext	-	2	60
	TMS320C30-27	2K	4K	-	16M•	2	16MX32ε	Int/Ext	-	2	74
	TMS320C30-40	2K	4K	-	16M•	2	16MX32	Int/Ext	-	2	50
	TMS320C31/	2K	ξ	-	16M•	1	16MX32	Int/Ext	-	2	60
	TMS320C31-27	2K	ξ	-	16M•	1	16MX32	Int/Ext	-	2	74
	TMS320C40/	2K	4K	-	4G•	-	4GX32	Int/Ext	6	2	40
TMS320C40-40	2K	4K	-	4G•	-	4GX32	Int/Ext	6	2	50	

- / Versión militar
- Espacio de memoria para programa, datos y E/S, menos RAM, periféricos y espacios reservados
- ξ Cargador booteable
- ε 16 puertos paralelos de E/S son mapeados a memoria
- Baja potencia (3V) de DSP

Fuente: TMS320 Digital Signal Processor, Product Family Bulletin. Texas Instruments (1992).

Texas Instruments recientemente introdujo el procesador MVP (*Multimedia Video Processor*), también se le conoce como el procesador digital de señales TMS320C80. Este procesador incluye 4 DSPs de punto fijo de 32 bits,

un procesador RISC de 32 bits con ejecución de 100 MFLOPS, dos controladores de video, 50 Kbytes de memoria SRAM y un controlador de transferencia de datos. Este procesador es programable y su ejecución de tareas paralelas está basada en un esquema de procesamiento en paralelo de múltiples datos (MIMD). Las primeras aplicaciones que han surgido para el MVP son: procesamiento de documentos por imagen, sistemas de diagnóstico por imagen, despliegue digital de video y sistemas de identificación de huellas dactilares [Reinhardt K., 1994].

La competencia en el mercado de procesadores digitales de señales (DSPs) está incrementándose, por lo cual Texas Instruments ha mejorado sus series C4x y C5x.

Se desarrolló una versión más rápida para el TMS320C40 a 80 MHz lo cual incrementará la eficiencia de aplicaciones como graficación, imágenes, *internetworking*, control, telecomunicaciones, robótica, procesamiento de datos, prueba de equipo y sistemas militares. El C40 a 80 MHz soporta E/S masiva de datos y programas a través de 2 buses externos de 32 bits, seis puertos *byte-wide*, comunicación entre procesos bidireccional y E/S. Su espacio de direcciones es de 32 bits y su transferencia DMA se lleva a cabo cada ciclo sin requerir memoria caché [FYI, Febrero 1994].

Actualmente se está produciendo la versión del TMS320C40 a 50 MHz y la versión a 60 MHz.

Tabla 3.6 Versiones del procesador TMS320C40.

	50 MHz MBytes/s	80 MHz MBytes/s
Ancho de banda total	320	494
Ancho de banda del bus	200	320
Ancho de banda del puerto de comunicación	120	174
Ciclo de instrucción	40 ns	25 ns

Fuente: FYI Integration. New DPS speeds parallel processing. Texas Instruments. Vol. 11 Núm.2 Febreo 1994.

Para la serie C5X de 16 bits se incrementó su velocidad a 40 MIPS con un voltaje de 3.3V. Hasta el momento se ha introducido al mercado japonés el TMS320C540 (C540) y más recientemente el TMS320C57 (C57) y el TMS320C56 (C56) [Reinhardt K., 1994].

Referencias

FYI Integration. New DPS speeds parallel processing. Texas Instruments. Vol. 11 Núm.2 Febreo 1994.

Harp, G., 1989. Transputer Applications. Ed. Pitman Publishing.

Hottinger, K., "More DSP solutions hit the streets". Electronic News, Vol.40 Núm.2020 pág.64(2). Junio 27, 1994.

Texas Instruments, 1991. TMS320C5x User's Guide.

Texas Instruments, 1992. TMS320 Digital Signal Processor, Product Family Bulletin.

Inmos,. 1989. The Transputer Databook.

Papamichalis, P., 1990. Digital Signal Processing Applications with the TMS320 Family. Volumen 3. Ed. Prentice Hall & digital Signal Processing Series Texas Instruments.

Pountain, D. The Transputer Strikes Back. BYTE Agosto 1991. Vol. 16,8 pág. 265-275.

Priemer, R., 1991. Introductory Signal Processing (Advanced Series in Electrical and Computer Engineering). Ed. World Scientific Publishing.

Reekie, J. "Realtime DSP: The TMS320C30 Course". (1994). Revision 3, 20 February 1994. School of Electrical Engineering University of Technology, Sydney, Australia.

Reinhardt, K. "Latest IBM 3.3V DSPs Double Performance". Electronic News, Vol.40 Núm.2004 pág.54(1). Marzo 7, 1994.

Reinhardt, K. "TI unveils Multimedia Video Processor". Electronic News, Vol.40 Núm.2005 pág.8(2). Marzo 14, 1994.

Reinhardt, K. "TI pressing 3.3V DSPs to 40 MIPS". Electronic News, Vol.40 Núm.2023 pág.95(1). Julio 18, 1994.

Roberts, J., 1992. Transputer Assembly Language Programming. Ed. Van Nostrand Reinhold.

4 Estudio de alternativas para el diseño del nodo de procesamiento heterogéneo

Introducción

La interconexión de un sistema, es la parte más importante de un sistema integrado por varios elementos de procesamiento, debido a que de ésta depende el intercambio de información entre procesadores. Realmente no existe una técnica de diseño para determinar qué estructura es la más adecuada, porque la solución depende de las necesidades de la aplicación o del sistema.

El propósito de este capítulo es plantear las distintas alternativas de diseño disponibles para la implementación del Nodo de Procesamiento Heterogéneo.

Básicamente se analizaron dos alternativas para desarrollar la interfaz: memoria compartida y canales de comunicación. De este análisis se determina cuál es la mejor alternativa, presentándose las ventajas y las desventajas para cada caso estudiado.

4.1 Evaluación de procesadores

Los elementos de procesamiento seleccionados para el diseño e implementación del nodo de procesamiento heterogéneo, como se vió en el Capítulo 1, son el transputer IMS T805 de Inmos y el DSP TMS320C30 de Texas Instruments.

Dentro de las características que se pueden considerar para la comunicación entre los procesadores IMS T805 y TMS320C30 se tienen las siguientes:

- El transputer IMS T805 tiene 4 lazos de comunicación independientes, los cuales son una parte integral de la filosofía del transputer. Además, son la única interconexión en sistemas multitransputers o multiprocesadores. Las conexiones o lazos operan asíncronamente a velocidades de hasta 20 Mbits/seg. Esto da una capacidad total de entrada/salida de 80 Mbits/seg por transputer. Esta velocidad es independiente del procesamiento interno debido al diseño particular del transputer [Inmos, 1989].
- El DSP TMS320C30 tiene dos timers y dos puertos seriales. Los dos puertos seriales son modulares y totalmente independientes. Cada puerto serial puede ser configurado para transferir 8, 16, 24 y 32 bits de datos por ventana. Los pines del puerto serial pueden ser configurados como pines de E/S de propósito general. Posee un modo especial de protocolo que garantiza la sincronización de la información. El TMS320C30 proporciona dos interfaces externas: la interfaz paralela y la interfaz de E/S. La interfaz paralela consiste de un bus de datos de 32 bits, un bus de direcciones de 24 bits y un conjunto de señales de control. La interfaz de E/S tiene un bus de datos 32 bits, un bus de direcciones de 13 bits y un conjunto de señales de control. Ambos puertos soportan una señal externa para la generación de estados de espera y el uso de estados de espera para el control de software. También cuenta con dos banderas de E/S, XF0 y XF1, que pueden ser configuradas como pines de entrada o salida por software. Estos pines son también usados por instrucciones de interbloqueo para soporte de comunicación de sistemas multiprocesadores [Papamichalis, P., 1990].

Una característica común en el Transputer y el DSP es que ambos procesadores están provistos de una interfaz serial.

Para el diseño de la interfaz se utilizó el Módulo de Evaluación del TMS320C30 (EVMTMS30) y el Kit de desarrollo del IMS T805 (TEK805) (ver Capítulo 5), por la disponibilidad de estos elementos (procesadores o kits) a nivel de laboratorio y para reducir el tiempo de diseño.

Cabe señalar que al utilizar el EVMTMS30 para el desarrollo de la interfaz, la única ruta de interconexión disponible para el DSP es el puerto serial 1, a través de un conector de 10 pines.

4.2 Métodos de interconexión

Para interconectar los elementos de procesamiento existen dos métodos, el primero de ellos está basado en compartir el espacio de direcciones entre los procesadores, de tal manera que todos los procesadores de un sistema puedan acceder un área de memoria común. El segundo se refiere a los sistemas en los cuales los elementos de procesamiento no comparten memoria y son conectados a través de canales de datos de E/S. La velocidad de transmisión de datos varía de pocos Kbits por segundo hasta Mbits por segundo. Las estructuras con un espacio de dirección común son llamados sistemas *multiprocessors* o *tightly coupled* y las que no comparten memoria son definidas como sistemas *multiple computer* o *loosely coupled*.

De acuerdo a lo anterior para realizar el diseño de la interfaz tenemos dos opciones: Utilizando memoria compartida o utilizando canales de comunicación, figura 4.1.

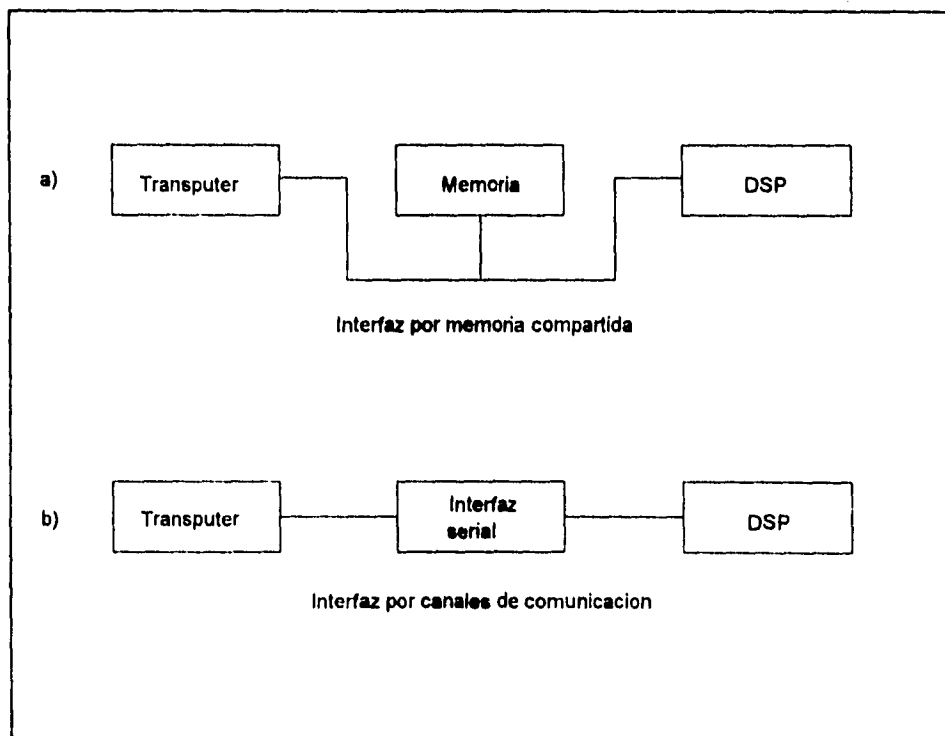


Figura 4.1 a) Interfaz por memoria compartida y b) Interfaz por canales de comunicación.

4.3 Interfaz por memoria compartida

Dentro del diseño de sistemas por memoria compartida existen diferentes soluciones para la interconexión de procesadores : *crossbar switches*, sistemas de memoria multipuerto, sistemas de bus compartido y redes multiestado [Conte, G., y Del Corso, D., 1985].

A continuación se presentan y evalúan estos esquemas de memoria compartida dentro del ámbito de diseño de la arquitectura del nodo heterogéneo.

Bus compartido. Un bus compartido es una simple ruta de comunicación para conectar unidades funcionales (tales como memorias y procesadores). Si es conectado únicamente un procesador al bus no hay problemas de contención. Cuando dos o más procesadores son conectados al mismo bus deben seguirse algunas reglas para establecer la comunicación como: asignación de un tiempo fijo a cada procesador y mecanismos de arbitraje para el manejo de requerimientos simultáneos.

Comentarios. Este tipo de interconexión no permite la transferencia simultánea entre diferentes pares de procesadores/memoria y desde luego una estructura de bus puede fácilmente llegar a ser el cuello de botella del sistema. Para el caso que nos ocupa esta alternativa resulta más compleja y costosa.

Memoria multipuerto. En estos sistemas el control y la lógica de arbitraje (que en un bus compartido reside en el procesador o en la interfaz de bus), están concentrados en módulos de memoria que presentan un número de interfaces (puertos) de comunicación a través de los cuales es posible acceder la información interna.

Cuando se tienen dos procesadores interconectados por una memoria de puerto dual o doblepuerto, la comunicación puede darse al escribir y leer datos en la memoria compartida, utilizándose interrupciones para solicitar la atención de un procesador hacia la memoria dual. Por ejemplo, si el procesador 1 desea comunicarse con el procesador 2, éste almacena la información y escribe a la bandera de interrupción de la memoria. Esto ocasiona que la línea de interrupción del procesador 2 se active. La interrupción es borrada por el procesador 2 al leer los datos. El procesador 2 entonces ejecuta una serie de tareas y sus resultados son enviados de

regreso a la memoria multipuerto donde son almacenados [Allen, A., y Wang, D., 1990].

Comentarios. Este esquema ofrece características que pueden ser aprovechadas en el diseño del nodo de procesamiento heterogéneo, sin embargo, el uso de este tipo de memoria resulta poco viable en sistemas donde se requiere que grandes cantidades de datos sean procesados, en virtud de la baja densidad actual de estas memorias.

Crossbar switches. En este esquema, un conjunto de rutas separadas son conectadas a cada banco de memoria y otra más a cada procesador. Un conjunto de interruptores puede conectar a varios procesadores a una memoria. El sistema soporta accesos simultáneos a todas las unidades de memoria. Se presenta contención sólo cuando el mismo banco de memoria es requerido por varios procesadores al mismo tiempo.

Comentarios. La implementación de un sistema de este tipo es poco costeable y además no resulta viable para una arquitectura que incluye sólo dos procesadores y un sólo espacio de memoria compartida, como es el caso que nos ocupa.

Red de interconexión multiestado. La red de interconexión de un sistema multiprocesador se construye usando un arreglo de bloques modulares. Cada elemento o bloque ejecuta una función de interruptor y puede ser configurado en una conexión directa o cruzada.

Para sistemas multiprocesador la terminal de entrada puede ser un elemento de procesamiento (una unidad de procesamiento asociado con una memoria local) y la terminal de salida puede ser un elemento de memoria global.

Si más de un par de terminales debe ser conectada simultáneamente pueden ocurrir conflictos en la ruta de comunicación, por esta razón la red de interconexión multiestado es dividida en tres clases: bloqueo, reordenación y no-bloqueo.

- Bloqueo. Si la red tiene conflictos es bloqueada.
- Reordenación. Si existe contención se ordena de otro modo las conexiones existentes.
- No-bloqueo. Si todas las interconexiones posibles son establecidas sin conflictos.

El uso de memoria compartida ocasiona contención entre los elementos de procesamiento cada vez que un mensaje es escrito (o leído) a memoria común, por lo que sólo un procesador puede acceder a memoria común en cada instante de tiempo.

Comentarios. Los mismos que se aplican al esquema de crossbar switches.

4.4 Interfaz por canal de comunicación

Como se vió en el inciso anterior, numerosos problemas se presentan cuando se usa un esquema de memoria compartida en un sistema multiprocesador. Al incrementar el número de procesadores, después de una mejoría inicial, causa una degradación en la eficiencia global debido a la competencia incrementada por el uso del bus de acceso a memoria. Esto ocasiona un tiempo ocioso en cada procesador puesto que tienen que esperar su acceso al bus. Por su parte, Inmos con el diseño del transputer facilita la comunicación entre procesadores a través de cuatro canales de comunicación serial, evitando de esta forma el cuello de botella en la comunicación que se presenta en sistemas con memoria compartida [Irwin, G., y Fleming P., 1992].

De lo anterior se determina que el método de interconexión por memoria compartida resulta poco práctico y viable, por lo cual, la implementación del nodo heterogéneo por medio de un canal de comunicación resulta ser la mejor opción de interconexión entre procesadores. De esta manera, se pueden aprovechar los elementos de comunicación (*links*) con los que cuenta el transputer y puede ser utilizado como cualquier otro nodo dentro de un sistema de procesamiento en paralelo de muy alta eficiencia.

Una conexión punto a punto para comunicación, involucra sólo dos unidades y mueve la información en una sola dirección. La **Figura 4.2** muestra los módulos que participan en una transferencia punto a punto. La unidad fuente corresponde a la unidad a la cual le pertenece la información (llamada también transmisor). La unidad destino por su parte, corresponde a la unidad de la cual se obtiene la información, gracias a la operación de transferencia (llamada también receptor).

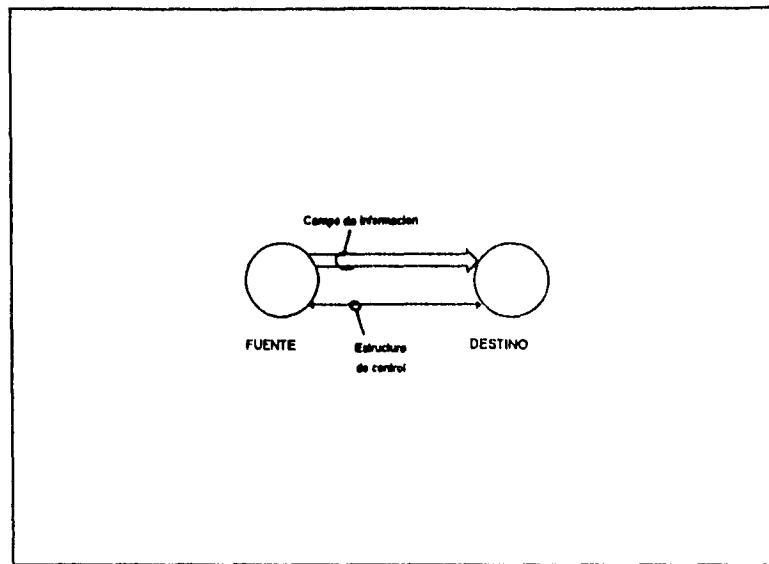


Figura 4.2 Transferencia punto a punto.

El canal de interconexión está compuesto por dos estructuras lógicas:

- un campo de información
- una estructura de control

Las operaciones en el canal de interconexión pueden ser descritas como una secuencia de acciones, tales como la habilitación de un buffer, activación de un registro, etc. El protocolo del canal especifica la secuencia de las acciones de control, las relaciones entre ellos y el flujo de información.

La principal ventaja de los canales de comunicación punto a punto sobre el uso de buses en sistemas multiprocesadores, es que no existe contención en los mecanismos de comunicación y éstos se van a ver incrementados al adicionar elementos de procesamiento (por ejemplo, transputers).

El canal de comunicación serial Inmos es un sistema de interconexión de alta velocidad y proporciona una comunicación full duplex entre los miembros de la familia de transputers. Es usado también como un interconector de propósito general donde no se utilizan transputers. Los adaptadores de canal (*link adaptors*) IMS C011 y IMS C012 son dispositivos de comunicación que habilitan el canal de comunicación serial Inmos

para ser conectado a puertos de datos paralelos y a los buses de un microprocesador.

El uso de un link adaptor en la implementación del canal de comunicación entre el transputer y el DSP resulta ventajoso en virtud de que el link adaptor maneja el protocolo de comunicación a través del link y los datos son leídos ó escritos sencillamente a través de un puerto. El **link adaptor** seleccionado fue el *IMS C011* (C011) el cual maneja dos modos de operación [Inmos, 1989]:

Modo 1: El C011 convierte un canal de comunicación en dos interfaces independientes, una de entrada y otra de salida. Puede ser utilizado por un dispositivo periférico para comunicarse con un transputer, por un procesador periférico Inmos, por otro link adaptor o proporcionar pines de entrada y salida programables para un transputer. Dos dispositivos C011 en este modo pueden ser conectados *back to back* vía los puertos paralelos para ser usado como un cambiador de frecuencia entre las diferentes velocidades de los links (5, 10, o 20 Mbits/seg).

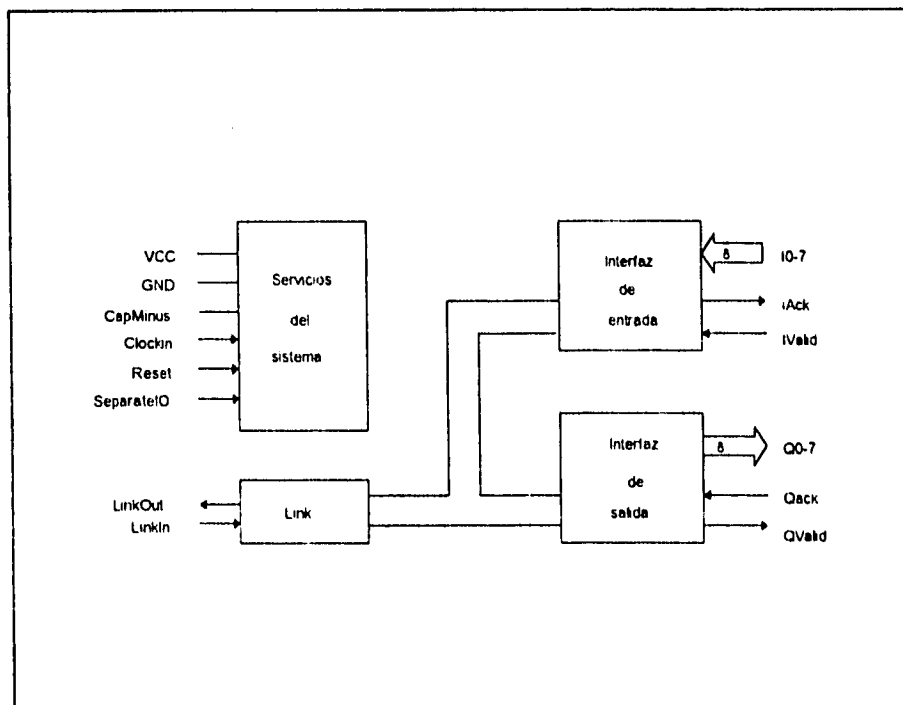


Figura 4.3 Diagrama de bloques IMS C011 MODO 1.

El C011 está configurado como una interfaz paralela con un protocolo de comunicación. Esta interfaz paralela comprende un puerto de entrada y un puerto de salida.

Puerto de entrada. El puerto paralelo de entrada de 8 bits (I0-7) puede ser leído por un dispositivo de la familia de transputers. Las señales IValid e IAck proporcionan el protocolo para este puerto. Cuando el dato es válido en I0-7, IValid es puesto a nivel alto por el dispositivo periférico para comenzar el protocolo de comunicación. El link adaptor transmite hacia afuera el dato presente en I0-7 a través del canal serial (*link*). Cuando el paquete de reconocimiento es recibido en el link de entrada, el C011 pone en alto la señal IAck para completar el protocolo y el dispositivo periférico regresa la señal IValid a nivel bajo. Finalmente, el link adaptor pone en un nivel bajo la señal IAck. El nuevo dato podrá ser puesto en I0-7 hasta que IAck se encuentre en un nivel bajo.

Puerto de salida. El puerto de salida paralelo de 8 bits (Q0-7) puede ser controlado por un dispositivo de la familia de transputers vía el canal de comunicación. Las señales QValid y QAck proporcionan el protocolo de comunicación para este puerto.

Un paquete de datos recibido en el canal de entrada es presentado en Q0-7. El link adaptor entonces pone en alto la señal QValid para iniciar el protocolo. Después de leer el dato de Q0-7, el dispositivo periférico pone en alto la señal QAck. Posteriormente, el C011 manda un paquete de reconocimiento fuera del canal de comunicación para indicar que la transacción ha sido completada y pone la señal QValid a un nivel bajo para completar el protocolo.

Modo 2. Permite tener una interfaz entre un canal de comunicación serial Inmos y el bus de un microprocesador. Los registros de datos y status para los puertos de entrada y salida pueden ser accedidos a través de la interfaz bidireccional de un byte. Tiene dos salidas de interrupciones, una para indicar que la entrada de datos está disponible y la otra para indicar que el buffer de salida está vacío.

La operación del link es controlada a través del bus de interfaz paralela (D0-7) por la escritura y la lectura de varios registros en el C011. Los registros son seleccionados por las señales RS0-1 y RnotW y el link adaptor es habilitado por la señal notCS.

D0-7. Los datos son comunicados entre un microprocesador (bus) y el link adaptor via el bus bidireccional (D0-7). El bus está en alta impedancia a menos que el link adaptor sea seleccionado y la línea RnotW esté en un nivel alto. El bus es usado por el microprocesador para acceder los registros de status y de datos.

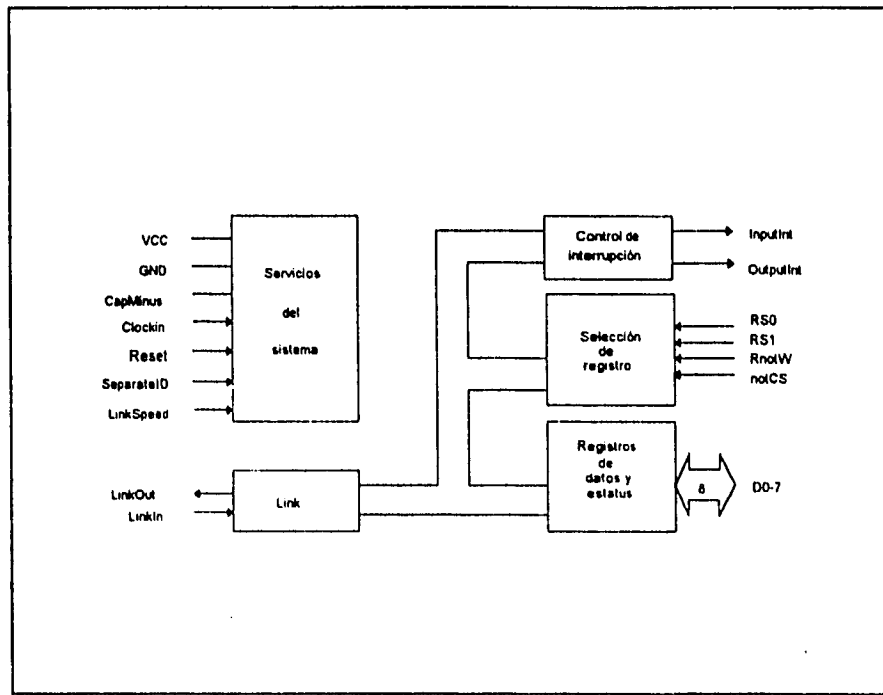


Figura 4.4 Diagrama de bloques IMS C011 MODO 2.

notCS. El C011 es seleccionado cuando la señal notCS está en un nivel bajo. El selector de registro RS0-1 y el RnotW deben ser válidos antes de que notCS este a nivel bajo; el bus de datos D0-7 debe ser válido si se esta escribiendo al C011 (RnotW nivel bajo). Los datos son leídos por el link adaptor al cambiar la señal de un nivel bajo a un nivel alto la señal notCS.

RnotW. Junto con la señal notCS, la señal RnotW selecciona los registros del C011 para lectura o escritura. Cuando RnotW está en un nivel alto el contenido del registro direccionado aparece en el bus de datos D0-7. Cuando RnotW está a un nivel bajo los datos en D0-7 ya han sido escritos al registro direccionado. El estado de la señal RnotW es almacenado en el

link adaptor al darse un nivel bajo en la señal notCS, éste puede cambiar antes de que notCS regrese a un nivel alto.

RSO-1. Uno de los cuatro registros es seleccionado por RSO-1. Un registro es direccionado poniendo en alto RSO-1 y en bajo notCS, el estado de RnotW cuando notCS esta en nivel bajo determina si el registro será de lectura o escritura. El estado de RSO-1 es tomado del C011 cuando la señal notCS está en nivel bajo, pudiendo cambiar antes de que notCS regrese a su estado alto. El registro puede incluir un registro de entrada de datos de sólo lectura, un registro de entrada de datos de sólo escritura y un registro de status de lectura/escritura para cada uno.

Registro de entrada de datos. Este registro mantiene el último paquete de datos recibido del canal de comunicación. El registro de entrada de datos no tiene paquetes de reconocimiento y el dato permanece en éste mientras la bandera del dato presente se encuentre en el registro de status de entrada. No se puede asumir que el dato está presente después de que el dato ha sido leído, una doble lectura puede o no regresar un dato válido en la segunda lectura. Si el dato presente es válido en una lectura subsecuente esto indica que un nuevo dato está en el buffer.

Registro de status de entrada. Este registro contiene la bandera del dato actual y el bit de control de habilitación de interrupción para InputInt. La bandera del dato actual es activada a nivel alto para indicar que el dato en el buffer de entrada es válido. El registro es borrado sólo cuando el dato del buffer de entrada es leído, o por la señal del Reset. Cuando se escribe a este registro, el bit del dato actual debe ser escrito como cero.

El bit de habilitación de interrupción puede ser activado e inicializado escribiendo al registro de status con el bit a nivel alto o bajo respectivamente. Cuando la habilitación de interrupción y la bandera del dato actual están en las dos señales en un nivel alto, la salida InputInt deberá estar en nivel alto. Al dar un reset a la habilitación de interrupción, InputInt estará en nivel bajo, habilitándolo de nuevo antes de leer el registro de entrada de datos el cual estará en nivel alto otra vez. El bit de habilitación de interrupción puede leerse para determinar este status.

Registro de Salida de datos. Los datos escritos a este registro son transmitidos fuera del C011 como un paquete de datos y esto se da únicamente cuando el bit de salida en el registro de status de salida está en nivel alto, de otra manera el dato puede ser erróneo.

Registro de status de salida. Este registro contiene la bandera de salida y el bit de control de habilitación de interrupción para la señal OutputInt. La bandera de salida en un nivel alto indica que el buffer de salida está vacío. El registro de status es inicializado en un nivel bajo sólo cuando el dato es escrito al buffer de salida de datos. Cuando se escribe a este registro, el bit de salida debe estar en cero.

El bit de habilitación de interrupción es activado o inicializado con sólo escribir al registro de status. Cuando las banderas de habilitación de interrupción y salida están ambas en nivel alto, la salida OutputInt deberá estar en nivel alto. Al inicializar la habilitación de interrupción OutputInt se pone en nivel bajo, mientras el registro de salida de datos esta vacío la señal OutputInt debe estar en un nivel alto. El bit de habilitación de interrupción puede ser leído para determinar este status.

InputInt. La salida InputInt es puesta a un nivel alto para indicar que el paquete de datos ha sido recibido del canal de comunicación. Esta señal se inhibe cuando el bit de habilitación de interrupción en el registro de status de entrada está en nivel bajo. La señal InputInt se desactiva cuando el dato es leído del registro de entrada de datos por un reset.

OutputInt. La salida OutputInt se activa en un nivel alto para indicar que el C011 está listo para recibir datos del microprocesador. Esta señal se inhibe si el bit de habilitación de interrupción en el registro de status está en nivel bajo. OutputInt es inicializado a un nivel bajo cuando el dato es escrito al registro de salida de datos.

Leer Dato. Un paquete de datos recibido en la entrada del C011 pone la bandera del dato actual en el registro de status de entrada. Si el bit de habilitación de interrupción es activado en el registro de status, la salida InputInt se activa en nivel alto. Entonces, el microprocesador podría responder ya sea a la interrupción (si el bit de interrupción está puesto) o bien, podría leer periódicamente el registro de entrada de datos hasta que el bit del dato actual éste en nivel alto.

Cuando el dato está disponible en el link adaptor, el microprocesador lee el paquete del registro de entrada de datos. Este puede inicializar la bandera del dato actual y el link adaptor transmite entonces un paquete de reconocimiento al canal de comunicación. La señal InputInt es

automáticamente inicializada con sólo leer del registro de entrada de datos; no es necesario leer o escribir el registro de status de entrada.

Escribir Dato. Cuando el buffer de salida del dato esté vacío, la bandera de salida en el registro de status de salida es activa en un nivel alto. Si el bit de habilitación de interrupción en el registro de status se activa, la salida OutputInt cambia a un nivel alto. Entonces, el microprocesador responde a la interrupción o puede leer periódicamente el registro de status de salida hasta que el bit de salida esté en nivel alto.

Cuando la bandera de salida está en un nivel alto, el microprocesador puede escribir el dato al buffer de salida del dato. El C011 entonces inicializa la bandera de salida y comienza la transmisión del paquete de datos fuera del link serial. El bit de status de salida puede permanecer bajo hasta que un paquete de reconocimiento es recibido a la entrada del link adaptor. Entonces, éste podría poner la bandera de salida en un nivel alto, si el bit de habilitación de interrupción esta activo y la señal OutputInt también estará en nivel alto.

El C011 soporta la velocidad de comunicación estándar de 10 Mbits/seg y también de 20 Mbits/seg. Su velocidad puede ser seleccionada de dos maneras. En modo 1 es modificada por la señal *SeparateIQ*. En modo 2 es seleccionada por la señal *LinkSpeed*, cuando la señal LinkSpeed está a nivel bajo, el link adaptor opera a 10 Mbits/seg y cuando está a nivel alto opera a 20 Mbits/seg [Inmos, 1989].

4.5 Diseño del canal de comunicación en base a un microcontrolador

Debido a que el uso de un microcontrolador en el diseño de un sistema disminuye los elementos de hardware utilizados, el tiempo de diseño y da mayor flexibilidad en cuanto a correcciones o modificaciones del sistema por ser dispositivos programables, se optó primeramente por utilizar el modo 2 de operación del link adaptor. Además, que ya existen microcontroladores que cuentan con un puerto serie, esto facilitaría su conexión al puerto serie del TMS320C30.

El microcontrolador utilizado fue el 80C51 de la familia MCS-51 de Intel. La **figura 4.5** muestra su diagrama de bloques.

El 80C51 es un dispositivo de tecnología CHMOS, el cual cuenta con un CPU de 8 bits optimizado para aplicaciones de control. El 80C51 realiza un procesamiento booleano extensivo (lógica de un bit), su espacio de direcciones en memoria de programa es de 64K al igual que su espacio de direcciones en memoria de datos. Cuenta asimismo con 4 Kbytes de memoria de programa dentro del chip y con una memoria RAM para datos de 128 bytes. El 80C51 tiene 32 líneas de E/S direccionables individualmente y bidireccionalmente, dos timers/contadores de 16 bits, un UART full duplex y una estructura de interrupciones con dos niveles de prioridad. Las fuentes de interrupción que proporciona el 80C51 son: 2 interrupciones externas, 2 interrupciones por timer y la interrupción por puerto serial [Intel, 1990].

Todos los dispositivos de la familia MCS-51 tienen separado el espacio de direcciones en memoria de programa y en memoria de datos. Esta separación lógica permite que la memoria de datos sea accesada por direcciones de 8 bits, con lo cual es más rápido el almacenamiento y manipulación con un CPU de 8 bits. Por su parte, el direccionamiento de la memoria de datos con 16 bits puede ser generado con el registro DPTR.

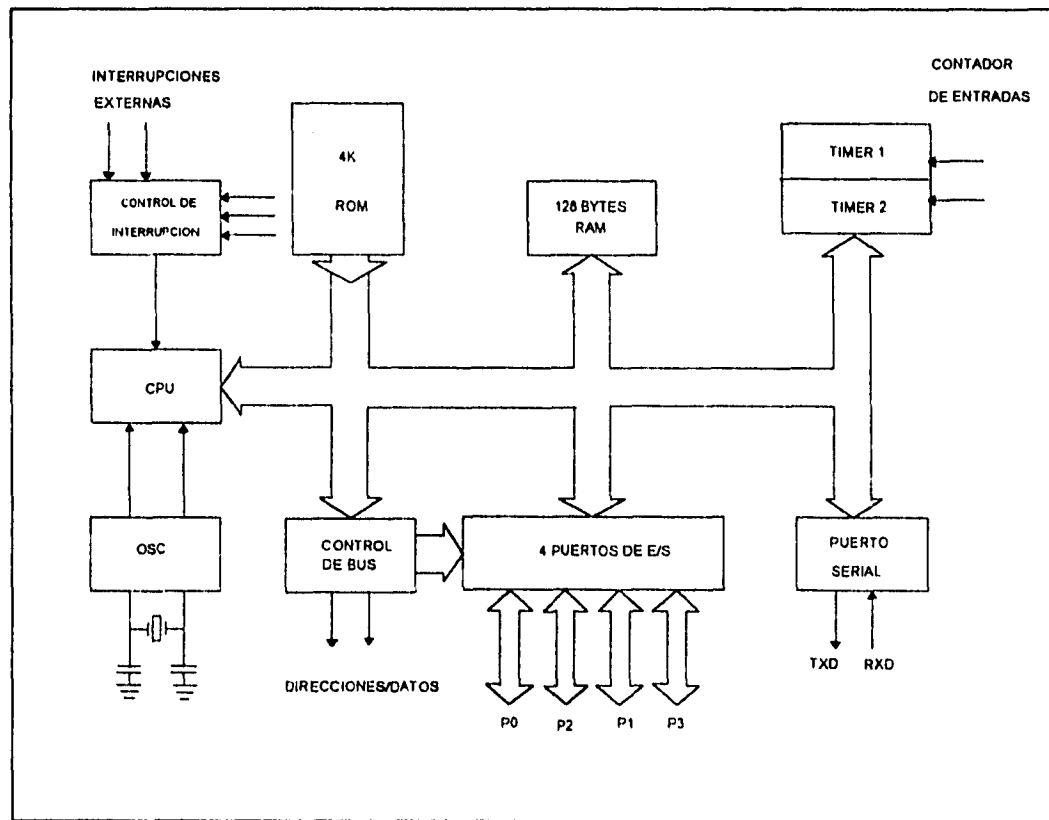


Figura 4.5 Diagrama de bloques del 80C51.

El diseño e implementación que se realizó con el 80C51 cuenta con una memoria EPROM externa en la cual se encuentra el programa que coordina la comunicación entre el transputer y el TMS320C30.

Para realizar la transferencia de datos entre el C011 y el DSP, se utilizó el puerto serie en modo 1 y el manejo de interrupciones externas del 80C51.

En la **figura 4.6** se puede observar que los puertos P0 y P2 son dedicados a la función de direccionamiento de la memoria de programa externa, funcionando el puerto P0 como un bus multiplexado de direcciones y de datos. Este emite el byte bajo del contador de programa (PCL) como una dirección y en un estado flotante espera la llegada del byte de código de la memoria de programa. Durante el tiempo que el byte bajo del contador de programa es validado en P0, la señal ALE (habilitación del latch de direcciones) manda este byte al latch de direcciones, mientras que el puerto P2 emite el byte alto del contador de programa (PCH). La señal NotPSEN habilita la EPROM y el byte de código es leído en el microcontrolador.

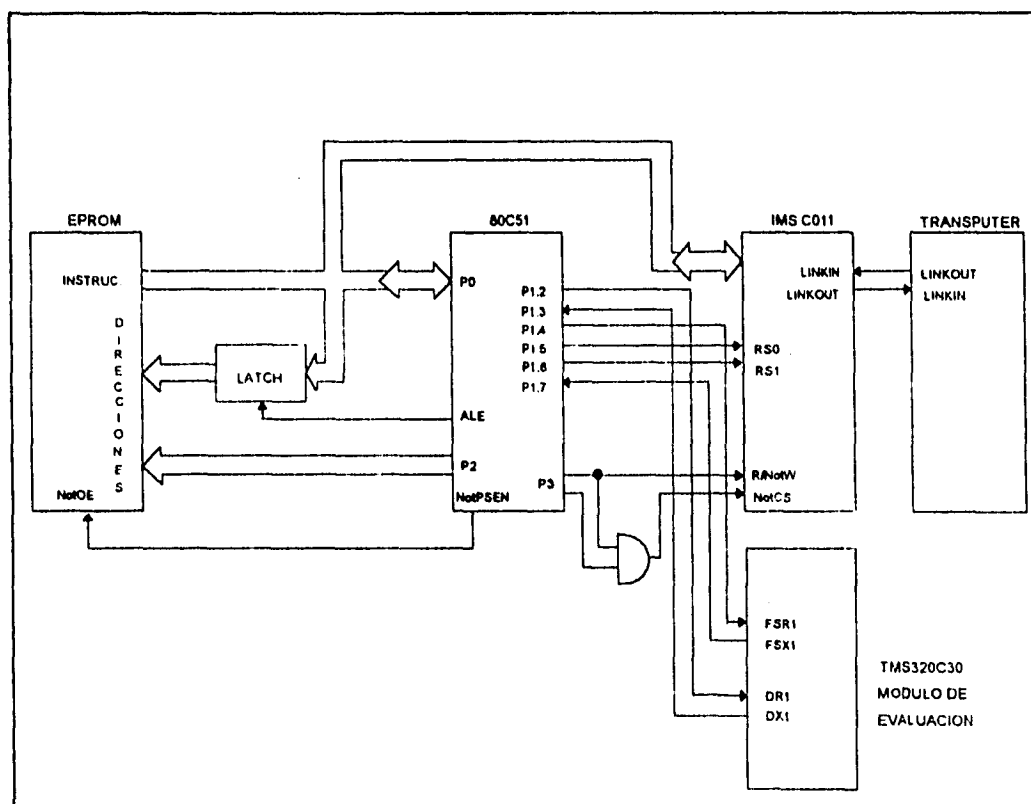


Figura 4.6 Diseño del nodo heterogéneo con un microcontrolador 80C51.

La habilitación del link adaptor depende de las señales de lectura y/o escritura emitidas por el 80C51 desde el puerto P3 y la habilitación de sus registros es controlada por el puerto P1.

Una vez que el dato ha sido transmitido por uno de los canales de comunicación del transputer al link adaptor, se detecta su presencia enmascarando el *registro de status de entrada* del link adaptor. Si el dato está presente, se lee el dato del *registro de lectura* del link adaptor, depositándolo en un registro del 80C51. Al tener el dato en el microcontrolador, por medio de una interrupción externa se transmite el dato al DSP. La señal P1.4 (FSR1 del TMS320C30) indica el comienzo de la transmisión del dato al C30 recibiéndolo este último por su puerto serie 1.

En cuanto a la transferencia de un dato al 80C51, el TMS320C30 envía la señal FSX1 (P1.7 del 80C51) indicando que empezará la transmisión en forma serial de un dato al microcontrolador. Una vez que el 80C51 tiene el dato en el registro de recepción del buffer de su puerto serial, lo transfiere a uno de sus registros y lo mantiene en éste. Mientras tanto el *registro de status de escritura* del link adaptor es enmascarado para determinar si el *registro de escritura* está disponible para enviar el dato al link adaptor y de éste al transputer.

Como el transputer IMS T805 y el link adaptor IMS C011 pertenecen a la misma familia (Inmos) la comunicación entre ambos es sincronizada. Cada byte de dato es transmitido como un bit de inicio en nivel alto, otro bit en nivel alto, seguido por 8 bits de datos y por un bit de paro en nivel bajo. Después de enviar un dato, el transmisor espera un reconocimiento del receptor que consiste de un bit de inicio en nivel alto seguido por un bit en nivel bajo.

4.6 Diseño del canal de comunicación en base a registros de corrimiento

El diseño basado en el link adaptor en modo 2 y un microcontrolador 80C51 fue implementado dando como resultado una velocidad de transferencia de datos muy baja. Sin embargo, éste esquema puede servir como base para el desarrollo de un puerto de comunicación serial estándar RS-232 para el transputer.

Por lo anterior se realizó otro diseño buscando incrementar la velocidad de transferencia entre los procesadores. Este nuevo esquema se basa también en el link adaptor IMS C011 pero considerando ahora el modo 1 de operación, ya que convierte un canal de comunicación (*link*) en dos interfaces independientes de 8 bits, una de entrada y otra de salida con un protocolo de comunicación.

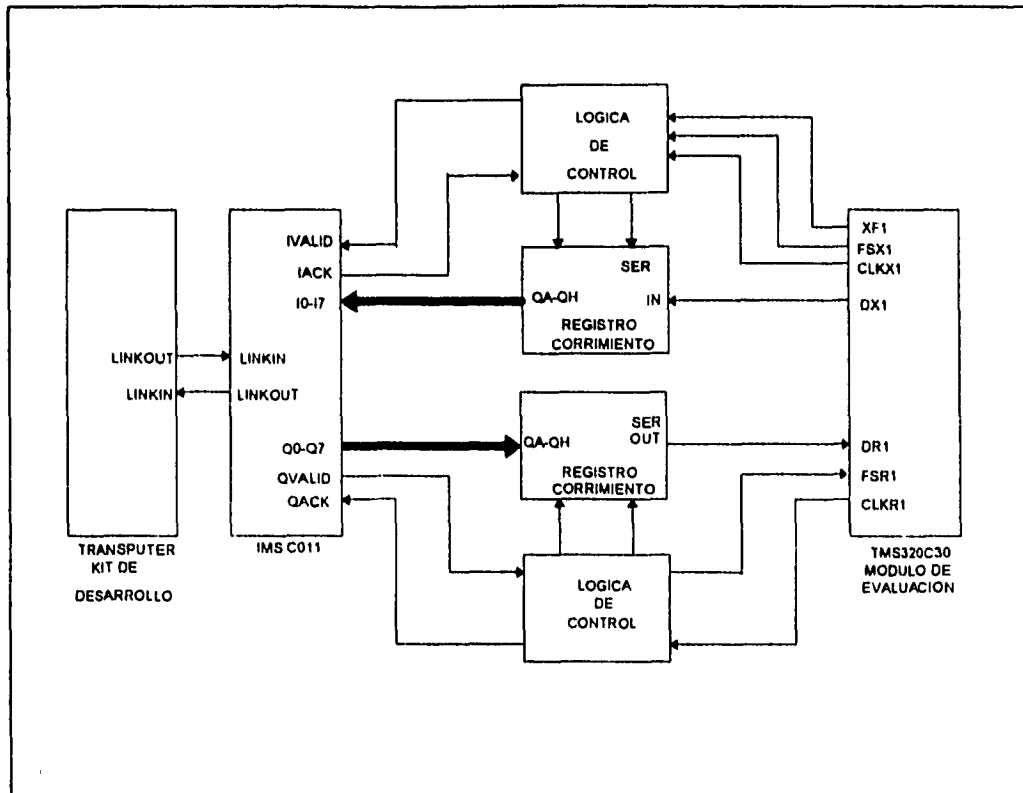


Figura 4.7 Diagrama de bloques del nodo heterogéneo utilizando lógica de control.

La figura 4.7 muestra un diagrama de bloques de la conexión del canal de comunicación (*link*) del transputer al puerto serie del DSP TMS320C30. El protocolo de comunicación entre el link adaptor y el DSP se logró utilizando registros de corrimiento y lógica de control dedicada (implementación detallada en el Capítulo 5).

Este último esquema fue utilizado en el presente trabajo debido a la alta velocidad de transferencia de datos obtenida, aunque presenta la desventaja de ser una interfaz muy poco flexible por la lógica de control, ya que si se desea hacer alguna modificación en hardware a la interfaz, esta tendría que rediseñarse.

Estudio de alternativas para el diseño del nodo de procesamiento... 4-18

Otra solución que resultaría óptima sería el realizar la interfaz de comunicación entre el transputer y el DSP, utilizando el link adaptor, lógica de control y en lugar de utilizar el puerto serie del DSP hacer un mapeo del link adaptor a memoria. Aún cuando esta solución resulta ser la más viable, se tuvo que optar por la solución a través del puerto serie del DSP, en virtud de que el sistema de evaluación EVMTMS30 del TMS320C30 no cuenta con opciones para el mapeo directo de componentes externas en memoria y sólo proporciona facilidades para acceder el puerto serie 1.

Referencias

Allen, A., Wang, D. "An application of ultrasonic signal processing in a mixed system of transputers and digital signal processor". Tools and Techniques for Transputers Applications by Stephen J. Turner. Proceedings of the 12th occam User Group Technical Meeting. 2-4 abril, 1990, Exeter, England.

Conte, G., Del Corso, D., 1985. Multi-microprocessor systems for real-time applications. Ed. D. Reidel Publishing Company.

García Nocetti, D. F., Solano J., Martínez J. "Heterogeneous Architecture for Parallel Real-Time Spectral Estimation in Doppler Blood Flow Instrumentation". Proceedings of the International Conference on Control '94. 21-24 March 1994, University of Warwick, Coventry, UK.

Inmos, 1989. The Transputer Databook.

Intel, 1990. 8-bit Embedded Controllers HandBook.

Irwin, G., Fleming, P., 1992. Transputers in Real-Time Control. Ed. John Wiley & Sons Inc.

Papamichalis, P., 1990. Digital Signal Processing Applications with the TMS320 Family. Volumen 3. Ed. Prentice Hall & digital Signal Processing Series Texas Instruments.

Sandler, M. "Interfacing the transputer to the TMS320 in an image processing enviroment". Microprocessors & Microsystems". Vol.12,9 Pág.490-496. Noviembre, 1988.

5 Implementación del Nodo de Procesamiento Heterogéneo en base a un transputer T805 y un DSP TMS320C30

Introducción

El tener un sistema de procesamiento heterogéneo proporciona la ventaja de contar con dos o más procesadores diferentes realizando la tarea más adecuada para cada uno, lo que permite lograr un incremento en la velocidad de procesamiento en comparación con los sistemas de procesamiento homogéneo.

Con base en la alternativa de diseño seleccionada del estudio presentado en el Capítulo 4, el presente capítulo busca describir física y operativamente el Nodo de Procesamiento Heterogéneo basado en el DSP TMS320C30. Asimismo se describe el transputer IMS T805, que es el tipo de procesador utilizado en la plataforma de procesamiento paralelo.

De esta manera, se dan las características más importantes de los procesadores utilizados en este trabajo de tesis. También se describen en forma breve los sistemas de desarrollo para el transputer T805 (TEK805) y del TMS320C30 (EVMTMS30), para posteriormente pasar a la descripción y teoría de operación de los circuitos implementados.

5.1 El transputer IMS T805 y el procesador de señales digitales TMS320C30

5.1.1 El transputer IMS T805

El Transputer IMS T805 [Inmos, 1989] es un procesador CMOS de 32 bits, el cual ejecuta operaciones aritméticas con una alta eficiencia y operaciones en punto flotante. La unidad de punto flotante (FPU) de 64 bits realiza operaciones de longitud simple y doble de acuerdo al

estándar para aritmética de punto flotante ANSI-IEEE 754-1985. La FPU ejecuta operaciones de forma concurrente con el procesador, manteniendo una velocidad de 2.2 MFLOPS a una velocidad del procesador de 20 MHz y de 3.3 MFLOPS a 30 MHz.

El T805 tiene 4 Kbytes de memoria RAM dentro del chip y puede acceder directamente a un espacio de direcciones de 4 Gbytes. La interfaz de memoria externa de 32 bits usa líneas multiplexadas de datos y direcciones, proporcionando una velocidad de datos hasta de 4 bytes cada 100 nanosegundos (40 Mbytes/seg) para procesadores de 30 MHz.

El transputer T805 cuenta con cuatro canales de comunicación estándar Inmos (*links de comunicación*). Estos canales de comunicación permiten construir redes de transputers, ya que facilitan la comunicación punto a punto sin adicionar lógica externa. Los links proporcionan una velocidad estándar de 10 Mbits/seg, operando también a 5 y 20 Mbits/seg. Cada canal de comunicación transfiere los datos bidireccionalmente a 2.35 Mbytes/seg.

El T805 también proporciona un ambiente gráfico. Este se realiza con instrucciones de movimiento de bloques de microcódigo, las cuales operan a la velocidad de la memoria. Existen instrucciones de movimiento de bloques de dos dimensiones que realizan movimiento de bloques continuos así como también copia de bloques. Las instrucciones de movimiento de bloques son utilizadas en las siguientes operaciones gráficas: manipulación de texto, manejo de ventanas, desplazamiento y actualización de pantalla.

Para flujos de datos de longitud arbitraria, el T805 ofrece el manejo de instrucciones de chequeo de redundancia cíclica (CRC), estas instrucciones proporcionan la detección de error donde la integridad de datos es crítica.

El conjunto de instrucciones realiza una eficiente implementación de lenguajes de alto nivel y proporciona soporte directo para el lenguaje Occam.

Una descripción más detallada del T805 se presenta en el Apéndice A.

5.1.2 El TMS320C30

El TMS320C30 (TMS30) es un procesador rápido (16.7 millones de instrucciones por segundo para un ciclo de instrucción de 60 nanosegundos) con un espacio de memoria de 16 millones de palabras de 32 bits y capacidad para aritmética de punto flotante. Además, tiene 6 mil palabras de 32 bits en memoria RAM y ROM dentro del procesador. Estas capacidades de memoria ofrecen soluciones a la implementación de algoritmos cuyos incrementos de memoria son significativos y a la vez reduce el uso de dispositivos periféricos.

Junto con las mismas líneas de memoria, el TMS30 tiene un bus interno por el cual se unen los dispositivos periféricos usando un mapeo de memoria dentro del mismo procesador. Dichos dispositivos incluyen dos puertos seriales síncronos de E/S que pueden operar a 2.5 Mbits/seg, dos timers y un controlador DMA. La modularidad de su diseño permite cambios fáciles, adiciones y eliminación de periféricos dependiendo de las necesidades de la aplicación.

El TMS30 cuenta también con un multiplicador de punto flotante de 40 bits y uno entero de 32 bits. Sus instrucciones, generalmente, se llevan a cabo en un sólo ciclo de reloj, con 2 ó 3 operandos por instrucción y cuenta con instrucciones para realizar operaciones paralelas.

Una descripción más detallada del TMS30 se presenta en el Apéndice B.

5.2 Descripción del nodo de procesamiento heterogéneo

Un sistema de procesamiento en paralelo está compuesto por varios elementos de procesamiento, los cuales operan de manera paralela, comunicándose unos con otros para intercambiar comandos y datos.

El principal beneficio del procesamiento en paralelo es el incremento de operaciones en el cálculo de funciones. El aumentar el número de elementos de procesamiento en un sistema redundante en un incremento en la velocidad de procesamiento.

El nodo de procesamiento heterogéneo surge como una propuesta para reducir los tiempos en el cálculo de funciones, el cual se integra como un elemento más en una plataforma de procesamiento paralelo. El nodo

heterogéneo realiza operaciones específicas, enfocadas principalmente al procesamiento de señales.

La plataforma en paralelo está formada por transputers (figura 5.1), y el nodo de procesamiento heterogéneo, incluido en la misma, está constituido por un DSP (Procesador de señales digitales) y por una interfaz de comunicación entre el transputer y el DSP.

Para el prototipo construido, se utilizó un sistema de desarrollo para el transputer [CSA, 1990] y un módulo de evaluación para el DSP TMS320C30 [TI, 1990].

La interfaz de comunicación utiliza uno de los canales de comunicación (*link*) del sistema de desarrollo del transputer y un puerto serie del DSP, disponible en el módulo de evaluación.

A continuación se describen brevemente los módulos de desarrollo TEK805 y EVMTMS30.

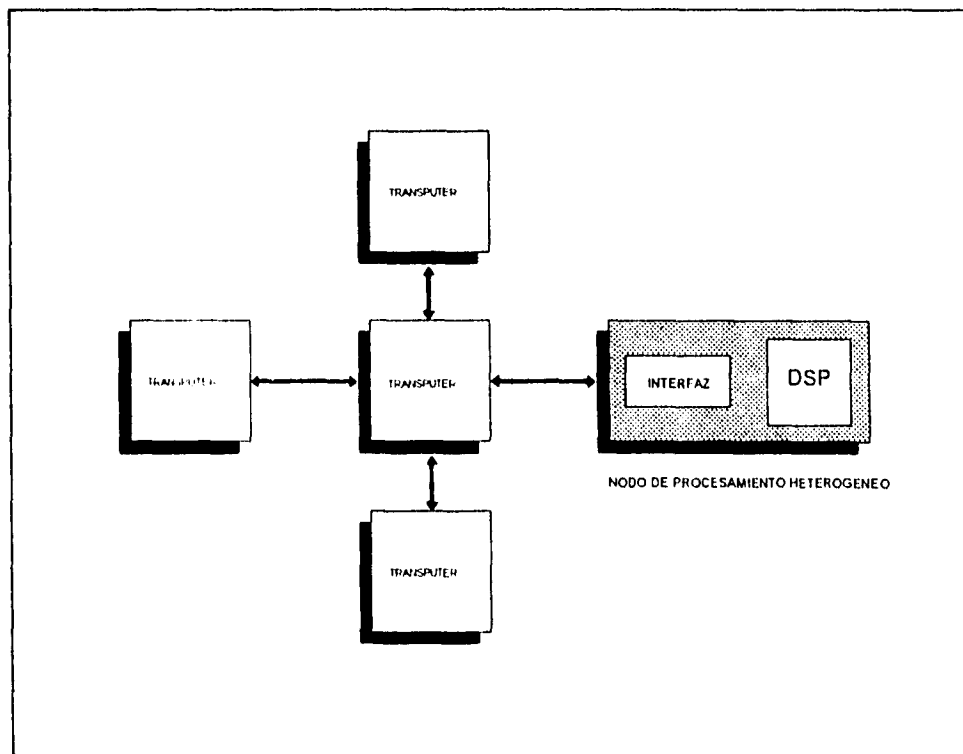


Figura 5.1 Diagrama a bloques del nodo de procesamiento heterogéneo.

5.2.1 Sistema de desarrollo para el transputer T805

La tarjeta del sistema de desarrollo del transputer T805 es fabricada por CSA (*Computer System Architects*) y está diseñada para experimentar con transputers y sistemas de multiprocesamiento.

El TEK805 se divide en tres secciones: el transputer T805, la interfaz PC-link y la interfaz externa. La interfaz PC-link ejecuta 2 funciones: Primeramente, es un medio de comunicación entre un link del transputer y la PC (computadora personal) y por otro lado permite que la PC genere y responda a un conjunto de señales de control conocidas como servicios del sistema (*system services*).

La interfaz PC-link tiene varios registros por los cuales la PC puede direccionar la lectura y la escritura de un dato del link y determinar el status de los registros de entrada y salida del mismo. La PC ve al link bidireccional del transputer como un dispositivo de E/S de un byte. Las señales del system services son: la señal *reset* del transputer, la señal *analyse* y la señal de *error*. La señal reset sirve para que la PC pueda iniciar o reiniciar al transputer o a la red de transputers unida a ella. La señal analyse proporciona un mecanismo para depurar programas en una red de transputers y la señal error es utilizada para notificar a la PC que existe una condición de error en uno o más de los transputers unidos a ella.

El PC-link está implementado con un link adaptor C012 de Inmos, que permite una interfaz para microprocesador de 8 bits convirtiendo los datos de un formato serie a uno paralelo y viceversa.

Por su parte, la interfaz externa consiste de un puerto de E/S programable, un puerto para LEDs y dos transistores de potencia opcionales. El puerto de E/S es de 8 bits y se utiliza para conectar hardware externo. El puerto para LEDs se emplea como una herramienta de depuración de software o como un conjunto de 8 señales de control, en unión con el puerto de E/S para una interfaz de hardware más versátil. Los transistores de potencia proporcionan una pequeña fuente de potencia conmutada para las interfaces unidas a través de un link.

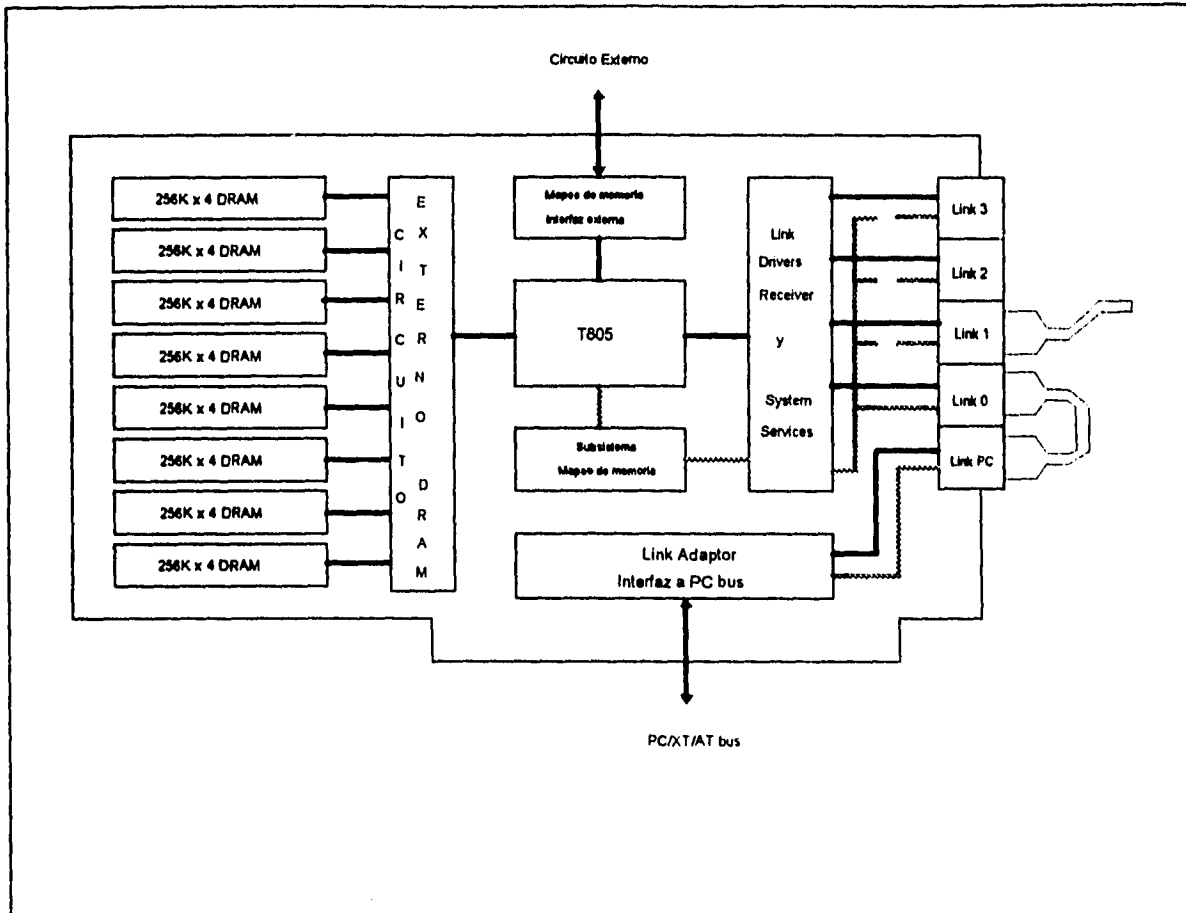


Figura 5.2 Sistema de desarrollo del T805.

Para comunicarse con otros transputers o formar redes de transputers, el TEK805 cuenta con varios conectores mini-DIN de 8 pines, los cuales son conectados a través de cables que llevan los datos del link bidireccionalmente y las señales de servicio de sistema (transputer reset, analyse y error).

El sistema de desarrollo del transputer, proporciona un conjunto de herramientas que permiten desarrollar software en lenguaje C, Occam y lenguaje ensamblador.

El compilador y el ensamblador para el lenguaje C corren ambos en la PC. Se requieren de 4 a 8 Kbytes de memoria en el TEK805 para ejecutar programas en C que usan librerías estándar de E/S. Mientras, que para el lenguaje Occam se requiere de 1 Mbyte de memoria en el sistema de

desarrollo para compilar los programas, en este caso los programas se compilan en el transputer y la PC actúa como un servidor anfitrión (*host server*). Por último, para desarrollar programas en lenguaje ensamblador se requieren únicamente de 2 Kbytes de memoria.

5.2.2 Módulo de evaluación para el TMS320C30

El Módulo de Evaluación del TMS320C30 (EVMTMS30) es una herramienta de desarrollo que permite ejecutar y depurar programas de aplicación usando un depurador de código fuente para lenguaje C y lenguaje ensamblador. Cuando se conectan entradas y salidas analógicas (micrófono o un altavoz) al sistema, el EVMTMS30 funciona como una herramienta de análisis de señales. Los datos analógicos pueden ser transferidos hacia o desde el *host PC* (o computadora personal anfitriona) a través del puerto de comunicación de 16 bits del EVMTMS30.

El poder de procesamiento del TMS320C30 lo hace un dispositivo popular para el diseño de sistemas de alta eficiencia que requieren megabytes de memoria y subsistemas de comunicación elaborados. Las mismas características se aplican al EVMTMS30, proporcionando también alta eficiencia con lógica mínima y bajo costo.

El EVMTMS30 se usa también como soporte de emulación en sistemas EISE (*embedded in-system emulation*).

Características del Módulo de Evaluación:

- DSP TMS320C30 de punto flotante (33 MFLOP)
- Memoria SRAM de 16 K palabras con cero estados de espera en el bus primario
- Adquisición de datos analógicos (con calidad de voz) vía el TLC32044
- Jack estándar RCA para entrada y salida analógica
- Puerto serial externo
- Puerto de comunicación bidireccional de 16 bits (PC host)
- Soporte de emulación *embedded* vía el controlador de prueba de bus 74ACT8990
- Tarjeta de 8 bits compatible con IBM PC/AT con cuatro direcciones de E/S disponibles.

En la figura 5.3 se muestra el diagrama de bloques y las interconexiones del EVMTMS30. El diagrama incluye la interfaz de emulación, la interfaz del host, una interfaz analógica, una interfaz de puerto serial y la memoria.

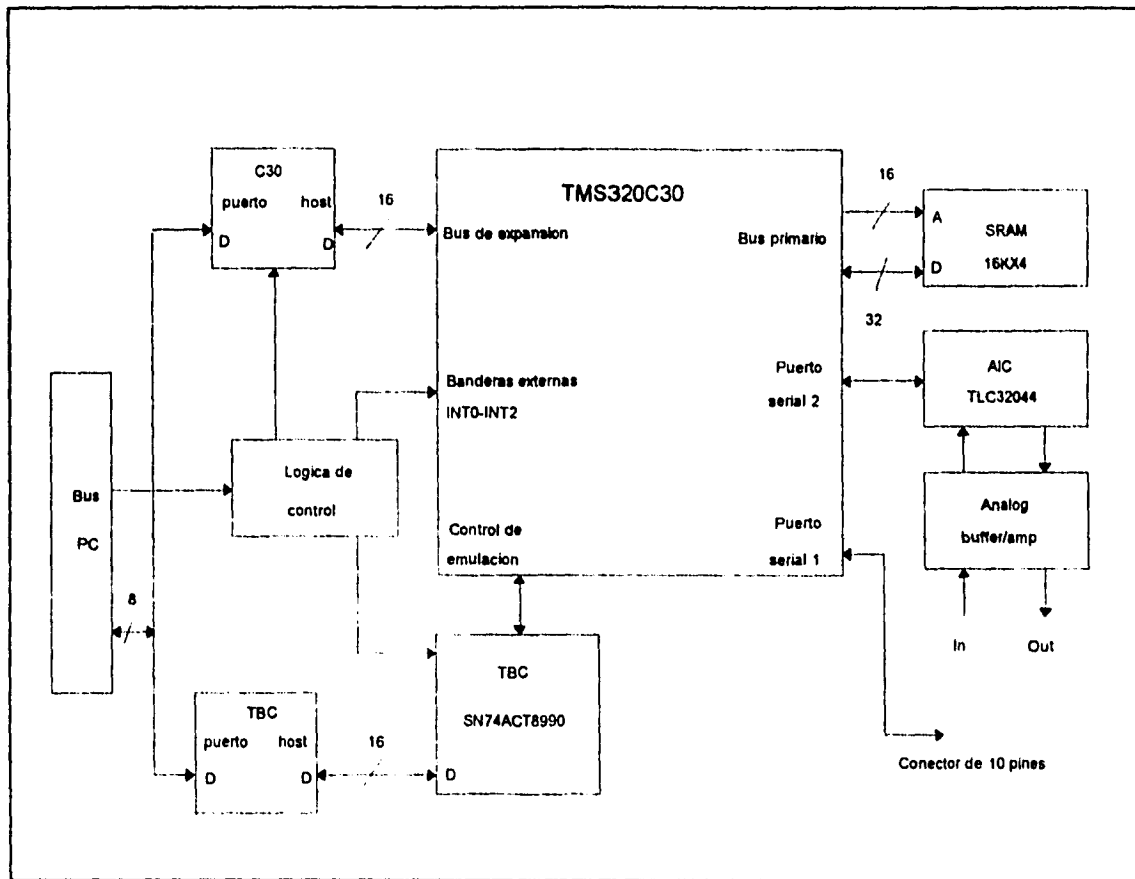


Figura 5.3 Diagrama de bloques del EVMTMS30.

El código a ser ejecutado por el EVMTMS30 es cargado a través del puerto de emulación del TMS320C30. Una vez que ha sido cargado el código, el host y el TMS320C30 se comunican por un registro compartido bidireccional de 16 bits.

La interfaz basada en el registro entre el host y el TMS30, proporciona una transferencia con un ancho de banda moderado de aproximadamente 200 Kbytes por segundo (KBPS). Esta interfaz está diseñada para ser usada con el canal DMA del TMS30.

La parte analógica del EVMTMS30 consiste de un AIC (*Analog Interface Circuit*) TLC32044 acoplado a un amplificador de audio de bajo ruido LM-386 para la salida y un TL072 como entrada. Las interfaces del AIC al TMS30 se realizan por el puerto serie 0. El puerto serie 1 queda disponible en un conector de 10 pines para uso externo.

La tarjeta del EVMTMS30 es instalada en un *slot* libre de la PC. El EVMTMS30 reside en el espacio de direcciones de E/S de la PC y requiere de 3 páginas cada una de 32 bytes (un total de 96 bytes) para el esquema de decodificación de las señales.

Puerto serial externo

El EVMTMS30 proporciona una interfaz de puerto serial externo por medio de un conector de 10 pines (J5). La tabla 5.1 muestra las señales de la interfaz y la figura 5.4 su ubicación física en la tarjeta.

Tabla 5.1 Señales del puerto serial externo.

Número de pin	Señal	Número de pin	Señal
1	GND	2	FSR1
3	CLKX1	4	DR1
5	DX1	6	TCLK1
7	FSX1	8	XF1
9	CLKR1	10	GND

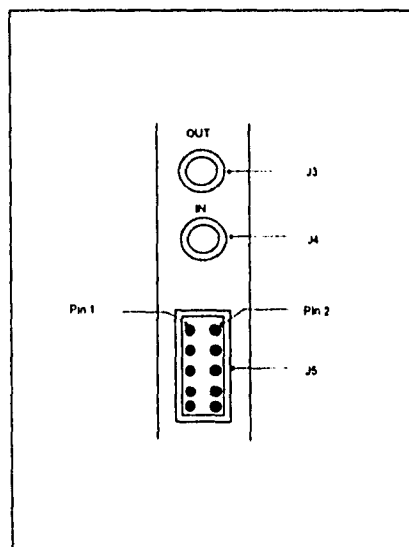


Figura 5.4 Señales del puerto serial externo.

5.3 Teoría de operación de la interfaz del nodo de procesamiento heterogéneo

La interfaz del nodo de procesamiento heterogéneo se puede dividir en 3 módulos: interfaz link adaptor, transmisión y recepción (figura 5.5).

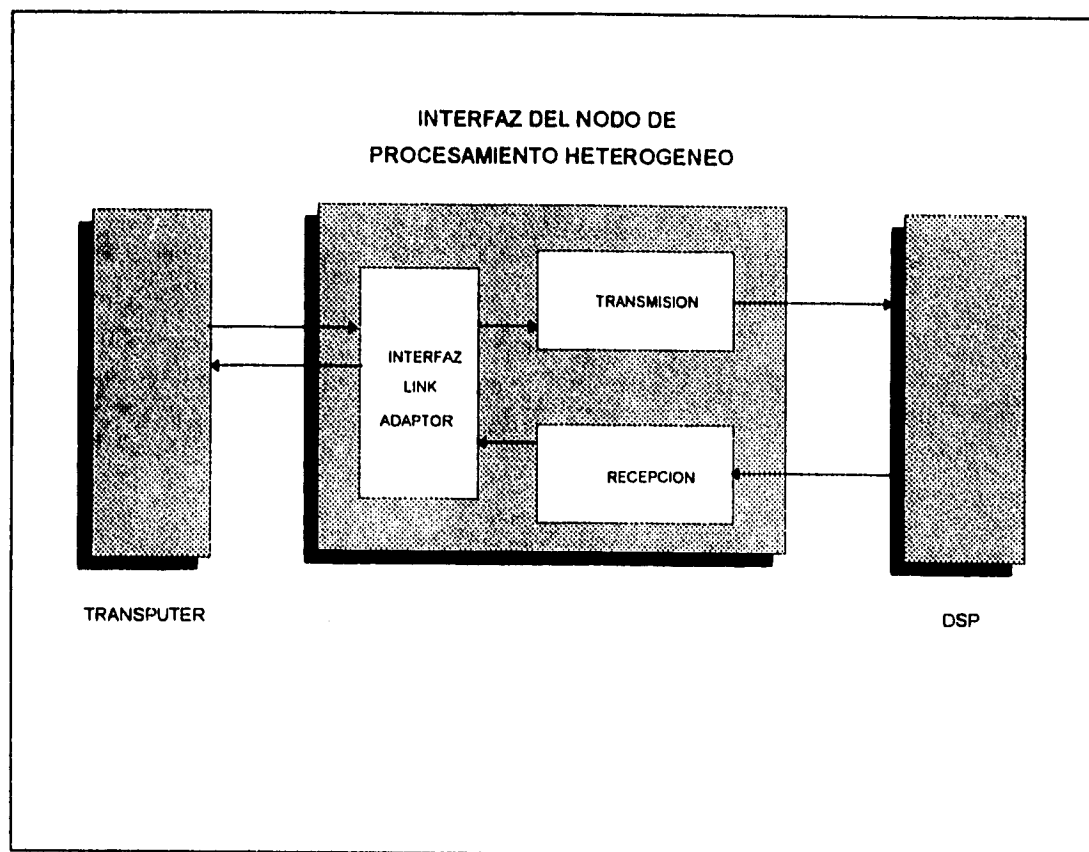


Figura 5.5 Diagrama a bloques de la interfaz del nodo de procesamiento heterogéneo.

En la interfaz del nodo de procesamiento heterogéneo, los datos a procesar son enviados por el transputer al módulo interfaz link adaptor por medio de uno de sus canales de comunicación serial. En éste módulo, los datos son transformados a un formato paralelo. Posteriormente, los datos en forma paralela son transferidos al módulo de transmisión y por medio de lógica de control se convierten a un formato serial requerido para ser enviado al puerto serie del TMS30. Una vez que los datos han sido procesados por el TMS30, éstos son enviados desde su puerto serie al

módulo de recepción, el cual convertirá los datos a un formato paralelo utilizando también lógica de control. Los datos con el formato paralelo son enviados de nuevo al módulo interfaz link adaptor el cual se encargará de transmitirlos serialmente al transputer.

Módulo Interfaz link adaptor

La interfaz link adaptor (ver diagrama electrónico 1/3 al final del capítulo), como se mencionó anteriormente, se refiere a la conversión a forma paralela de uno de los canales de comunicación serial del transputer. Esta interfaz está formada por los siguientes elementos: un *driver* diferencial DS8921, un oscilador de 5 MHz, un conector de E/S de 8 pines (Mini-DIN) y un Link adaptor IMS C011 (C011).

El C011 (ver Capítulo 4) es un dispositivo de comunicación configurado en modo 1, lo que permite conectar a un link del transputer con dos interfaces paralelas de 8 bits: una de entrada (I0-7) y otra de salida (Q0-7), teniendo, de esta manera, la conversión de un dato serie a uno paralelo o viceversa.

Dado que se seleccionó el modo 1 para la operación del C011, se consideró adecuado el uso de la velocidad estándar de 10 Mbits/seg, debido a que con una velocidad de 20 Mbits/seg se presentan problemas de comunicación entre el transputer y el C011. Estos problemas son ocasionados a la distancia relativamente grande (30 cm aproximadamente) que hay entre ellos. Por lo anterior la señal SeparatelQ está conectada a VCC.

El C011 tiene una entrada llamada ClockIn por la cual se conecta el oscilador con una frecuencia estándar de 5 MHz. Este oscilador es de cristal de cuarzo debido a que tiene mejor estabilidad que uno de tipo RC. Asimismo, como la potencia derivada internamente por los requerimientos del reloj requiere de una baja inductancia, es necesario conectar un capacitor de cerámica de 1 μ F (C02) entre el pin CapMinus y la señal de VCC.

La señal de Reset del C011 se activa con un nivel de VCC. Dicho nivel se proporciona por medio un circuito de reset que permite inicializar el C011 a través de un push-button o por medio de la señal de reset del transputer. El reset no debe exceder el voltaje máximo de VIH (VCC+0.5

volts). Después de presentarse la señal de VCC, la señal ClockIn debe alcanzar un período mínimo TDCVRL antes de terminar el reset y a su vez la señal LinkIn debe permanecer en un nivel bajo durante el mismo. La tabla 5.1 y la figura 5.6 muestra los tiempos de la señal de reset.

Tabla 5.2 Tiempos de la señal de Reset.

SÍMBOLO	PARAMETROS	MIN	NOM	MAX	UNIDADES
TPVRH	Potencia valida antes del Reset	10			ms
TRHRL	Pulso a nivel alto del Reset	8			ClockIn
TDCVRL	Duración de ClockIn antes de terminar la señal Reset	10			ms

La señal de Reset al inicializar al C011 deja a la señal LinkOut en un nivel bajo, al igual que las señales de control lAck y QValid, mientras que el estado del puerto de salida Q0-7 no es especificado.

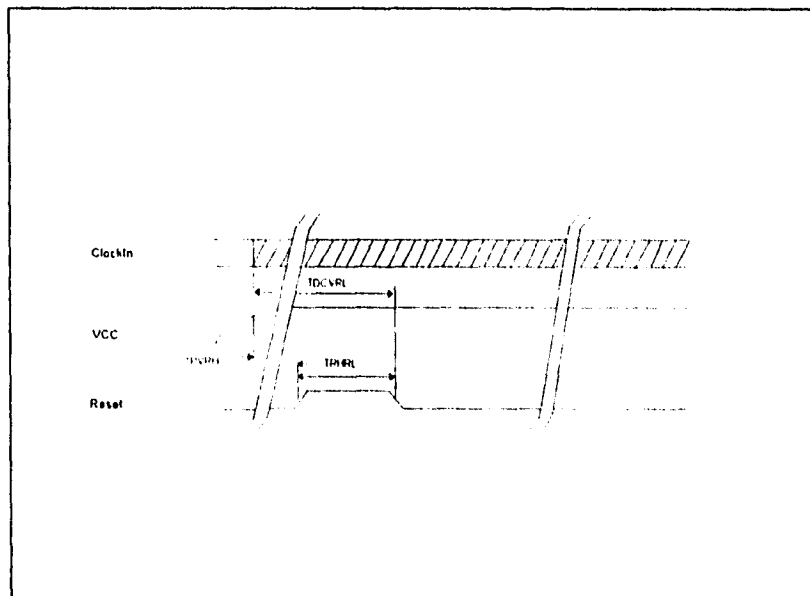


Figura 5.6 Tiempos de la señal reset.

El link del transputer se conecta al mini-DIN (JP01) por medio de un cable. De esta forma, los datos son transferidos bidireccionalmente al igual que las señales reset, analyse y error del transputer al link adaptor.

Para no tener problemas con la distancia entre el canal de comunicación y el C011, se utilizó un driver diferencial DS8921-RS422 que puede proporcionar una distancia de comunicación hasta de 30 metros aproximadamente.

Para llevar a cabo una transferencia sincronizada, el link del transputer maneja paquetes de datos y de reconocimiento. Siempre que un dato es enviado desde el transmisor, el receptor envía un paquete de reconocimiento de 2 bits, indicando con esto que recibió la información. El transmisor, mientras tanto, espera el paquete de reconocimiento, antes de enviar otro paquete de datos. Si todas las transmisiones son reconocidas de esta forma, ningún dato se pierde, pero si el receptor no está listo para recibir más datos, la comunicación se suspende.

Módulo de Transmisión

La función del módulo de transmisión (ver diagrama electrónico 2/3) es recibir un dato del puerto de salida del C011 (módulo interfaz link adaptor) y transmitirlo serialmente al puerto serie del TMS30. El módulo de transmisión está formado por los siguientes elementos: un contador síncrono de 4 bits (74LS191), un registro de corrimiento de 8 bits con entrada paralela y salida serial (74LS165), dos flip-flops tipo D (74LS74), un circuito de reset, inversores, compuertas ANDs, una compuerta OR y las señales de recepción del puerto serie 1 del TMS30.

Teoría de operación

Al ser transmitido por el transputer un dato, éste es recibido por el link de entrada **Linkin** del C011 el cual lo presentará en el puerto de salida **Q0-7**. Al presentarse el dato en Q0-7, el C011 inicia el protocolo de comunicación activando en nivel alto la señal **QValid**. Por su parte, el TMS30 genera internamente la señal de reloj **CLKX1** para la transmisión. La señal **CLKX1** es conectada a la señal de entrada **CLKR1** (señal de reloj de recepción) y sirve para sincronizar la transferencia de datos al DSP. Cabe señalar, que el puerto serie del TMS30 es programado en modo continuo con una velocidad de datos fija cuyo manejo de señales se ha hecho coincidir con el funcionamiento de la lógica del módulo de transmisión [TI, TMS320C3x User's Guide, 1992]. Con la señal de reloj **CLKX1** y la señal **QValid** en alto los flip-flops U10A y U10B generan un pulso a nivel alto en la señal **FSR1** que le indica al DSP el comienzo de la transmisión del dato

acorde a los siguientes pulsos de reloj. Simultáneamente, al pasar la señal FSR1 por el Inversor U07C (LOAD), se activa con un nivel bajo la carga paralela del registro de corrimiento U11, con lo cual se transmitirá serialmente el dato presente en Q0-7 al TMS30 por el pin DR1, con cada transición de bajo a alto en la señal de reloj de U11. Con el cambio de nivel de alto a bajo de la señal LOAD generada a la salida del inversor U07C, se habilita igualmente el contador U12, el cual después de 8 ciclos de reloj activa en alto su salida QD y junto con la señal QV" generan un pulso a nivel alto al pasar por la compuerta AND U09C. Con este pulso y la señal TCLK1 del puerto serie del TMS30 se activa en alto la señal QACK del link adaptor, la cual indica que el dato ya fue aceptado por parte del TMS30. La señal TCLK1 es controlada por software desde el TMS30 y permanece en nivel alto durante toda la transmisión del dato, lo que previene de un error de comunicación en el caso de que el TMS30 no estuviera listo para aceptarlo. El IMS C011 envía entonces un paquete de reconocimiento al link serial de salida LinkOut para indicarle al transputer que ha sido completada la operación de transmisión y la señal QValid se desactiva a nivel bajo para completar el protocolo de comunicación. En la **figura 5.7** se muestra el diagrama de tiempos para el módulo de transmisión.

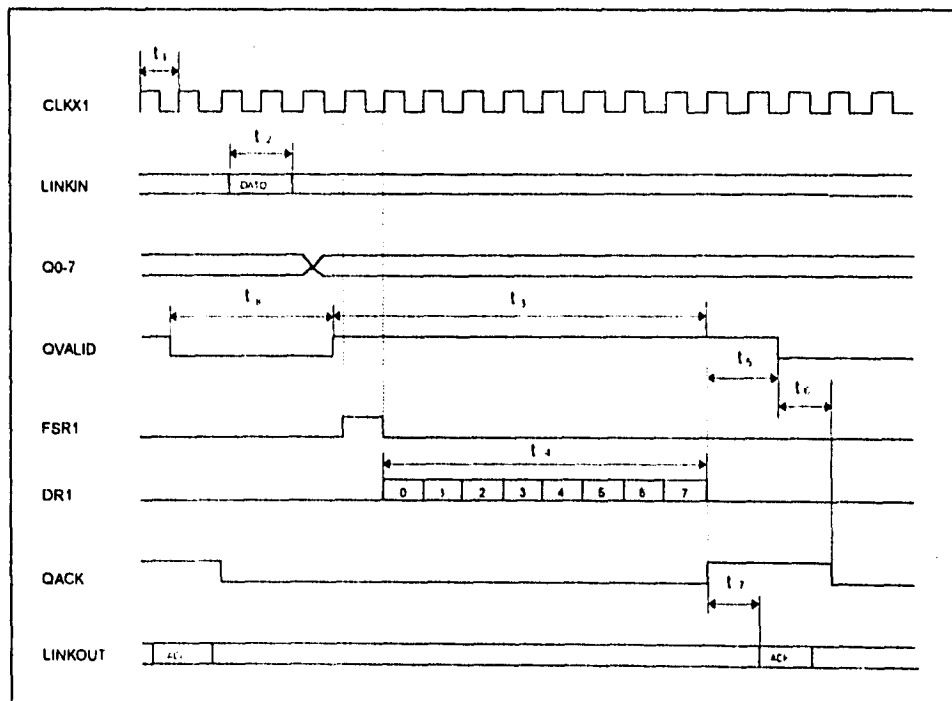


Figura 5.7 Diagrama de tiempos del módulo de transmisión.

Tabla 5.3 Tiempos del módulo de transmisión.

Símbolo	Tiempo (μ s)	Descripción
t_1	0.26	Duración del ciclo de reloj
t_2	0.9	Dato presente en LINKIN
t_3	2.6	Tiempo requerido para transmitir el dato al puerto serie del TMS30
t_4	$t_1 \times 8$	Tiempo de recepción del dato en el TMS30
t_5	0.35	Desactivación de QVALID después de la activación de QACK
t_6	0.4	Desactivación de QACK después de la desactivación de QVALID
t_7	0.08	Reconocimiento en LINKOUT después de la activación de QACK
t_8	1.65	Nuevo dato en Q0-7 después de concluida la transmisión del dato anterior

Nota:

$$t_{TX} = t_3 + t_5 + t_8 = 4.6 \mu s$$

donde t_{TX} representa el tiempo de transmisión por byte, medido al transmitir datos de manera continua.

Módulo de Recepción

El módulo de recepción tiene la función de recibir el dato transmitido por el puerto serie del TMS30 y convertir el dato de un formato serie a uno paralelo para posteriormente transmitirlo al módulo interfaz link adaptor. El módulo de recepción cuenta con los siguientes elementos: un registro de corrimiento de 8 bits con entrada serial y salida paralela (74LS164), un contador síncrono de 4 bits (74LS191), dos flip-flops tipo D (74LS74), inversores, compuertas AND's y por las señales de transmisión del puerto serie 1 del TMS30.

Teoría de operación

Para recibir datos en el transputer por parte del TMS30, la lógica del módulo de recepción pone en un nivel alto la señal IValid para indicar que existe un dato válido en el puerto de entrada I0-7 del link adaptor. En el diagrama electrónico 3/3 se muestra la lógica del módulo de recepción. Para activar la señal IValid, el TMS30 genera internamente la señal de sincronía FSX1 para la transmisión, que al pasar por el inversor U06D (LOAD) activa la carga del contador U01 con un cambio de transición de nivel alto a bajo. El contador permanece habilitado durante

8 ciclos de reloj y al noveno ciclo se deshabilita. La salida QD del contador activa la señal de reloj del flip-flop U03B. Con la activación de esta señal de reloj, se activa a nivel alto la señal IValid indicando al C011 que el dato está presente en I0-7. Cuando se presenta la señal FSX1, se habilita al mismo tiempo el flip-flop U03A generando en su salida Q la señal de control SHFCK_EN que junto con la señal de reloj CLKX1 a la salida del inversor U06 entran a una compuerta AND (U05A) y de esta forma activa la señal de reloj del registro de corrimiento U02. De esta forma el dato será transmitido serialmente desde el puerto serie del TMS30 a través del pin DX1 y será convertido a un formato paralelo por el registro de corrimiento. Después de 8 ciclos de reloj se presenta un cambio de transición en la señal de reloj del flip-flop U03A, lo que ocasiona que la señal de control SHFCH_EN a través del flip-flop U03A sea desactivada y por consiguiente el registro de corrimiento detiene el desplazamiento del dato, evitando de esta forma que la información sea errónea.

Una vez que el dato está presente en I0-7, el C011 envía el dato al transputer. Si el dato es recibido correctamente por el transputer, éste envía un paquete de reconocimiento por el link serial de entrada LinkIn del C011 activándose en un nivel alto la señal IAck. La señal IAck junto con la señal TCLK1 desactivan la señal IValid y poco tiempo después la señal IAck es desactivada por el C011 finalizando con esto el protocolo de comunicación.

Para asegurar, que la información transmitida por el TMS30 sea recibida por el transputer, la señal TCLK1 es controlada por software. Dicha señal se activa en un nivel alto únicamente cuando la señal IAck está también en un nivel alto y se desactiva cuando IAck cambia de nivel. Los cambios de nivel de la señal IAck son detectados por software con la señal XF1 del puerto serie.

Al igual que para el módulo de transmisión, el modo de operación del puerto serie del TMS30 para el módulo de recepción debe ser continuo con una velocidad de datos fija.

En la figura 5.8 se da el diagrama de tiempos para el módulo de recepción.

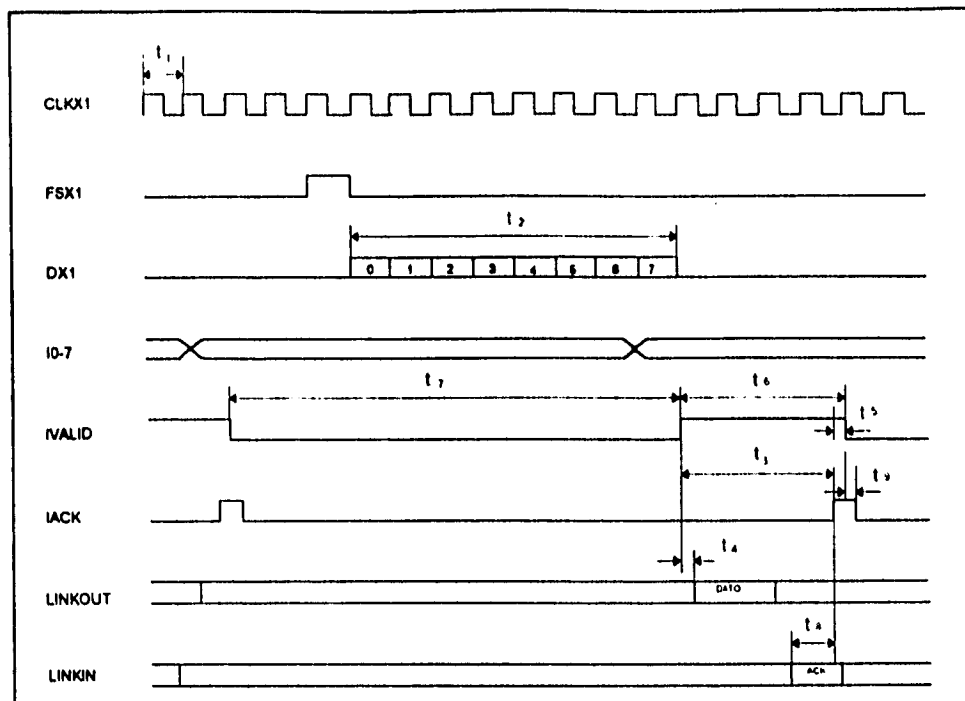


Figura 5.8 Diagrama de tiempos del módulo de recepción.

Tabla 5.4 Tiempos del módulo de recepción.

Símbolo	Tiempo (μs)	Descripción
t_1	0.26	Duración del ciclo de reloj
t_2	$t_1 \times 8$	Tiempo de transmisión del dato por el TMS30
t_3	1.3	Activación de IACK después de la activación de IVALID
t_4	0.2	Dato presente en LINKOUT después de la activación de IVALID
t_5	0	Desactivación de IVALID después de la activación de IACK
t_6	1.65	Activación de IVALID
t_7	2.55	Nuevo dato en el puerto serie del TMS30 después de concluida la recepción del dato anterior
t_8	0.25	Activación de IACK después de presentarse el reconocimiento en LINKIN
t_9	0.1 a 0.4	Desactivación de IACK después de la desactivación de IVALID

Notas:

- 1 Con una velocidad de comunicación de 10 Mbits/seg en el link, un bit tarda 100ns en ser transferido [Inmos, 1989].
- 2 $t_3 = T_{lvHLdV} + \text{dato} + T_{LaV}t_{aH}$
 $t_3 = 2\text{bits} + 8\text{bits} + 3\text{bits} = 1.3\mu\text{s}$
- 3 $t_4 = T_{lvHLdV} = 2\text{bits} = 0.2\mu\text{s}$

4 $t_5 = T_{LaHlvL} = 0 \eta s$

5 $t_9 = T_{lvLlaL} = 1 \text{ bit a } 4 \text{ bits} = 0.1-0.4 \mu s$

6 $t_{RX} = t_6 + t_7 = 4.2 \mu s$

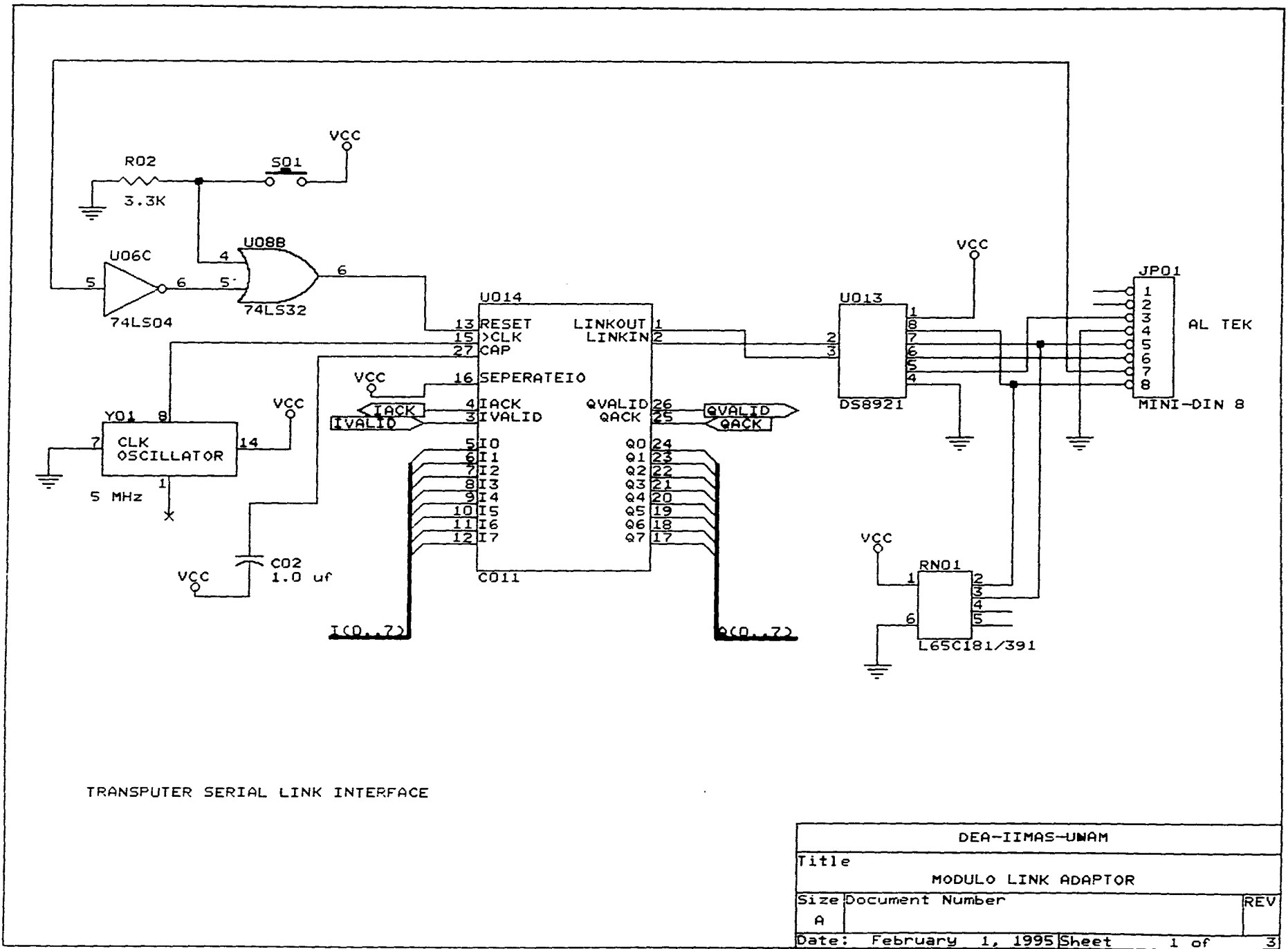
donde t_{RX} representa el tiempo de recepción por byte, medido al recibir datos de manera continua.

donde:

T_{lvHLdV} , T_{LaVlaH} , T_{LaHlvL} , y T_{lvLlaL} son obtenidos del manual [Inmos, 1989].

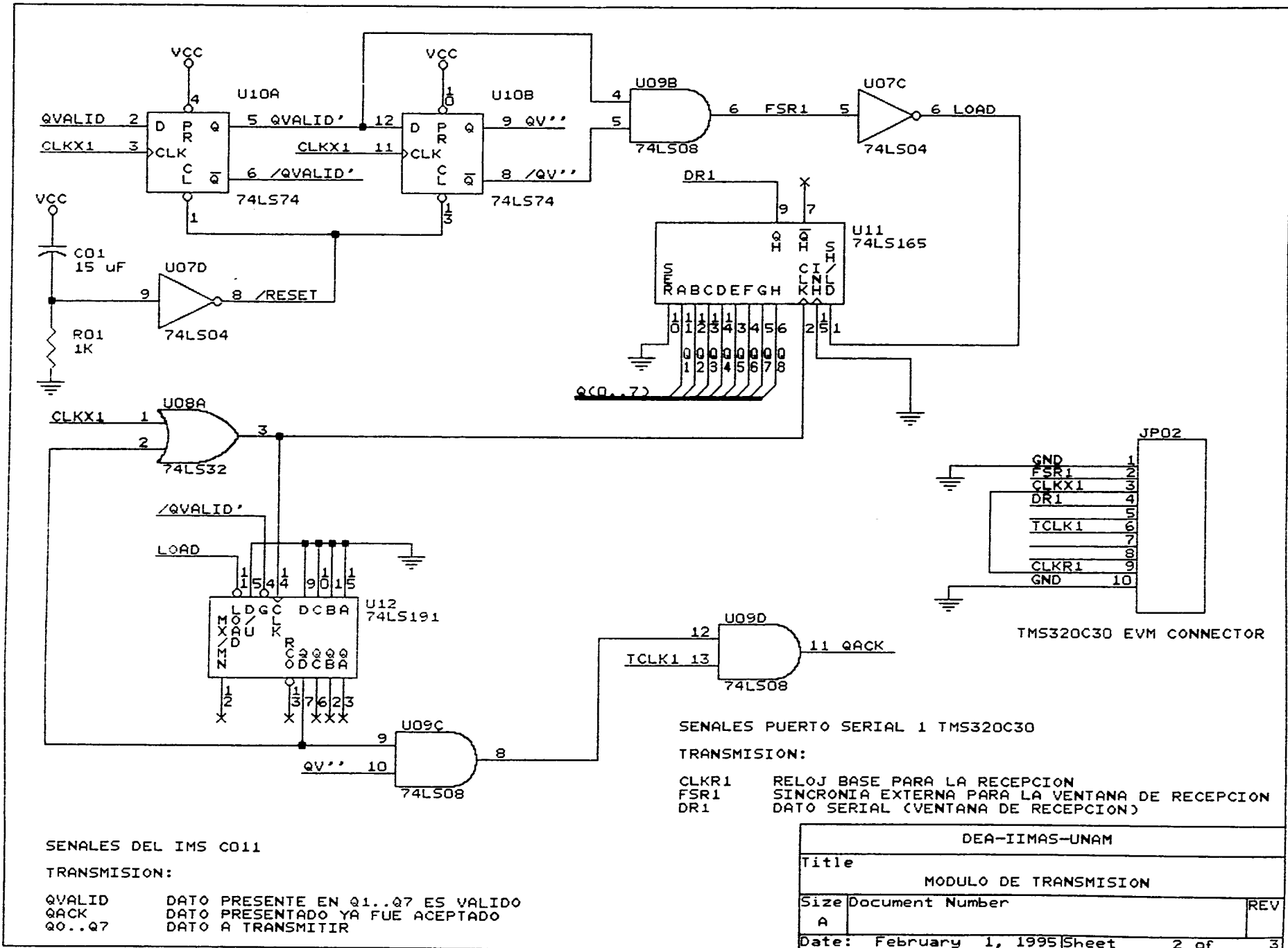
Tabla 5.5 Lista de componentes.

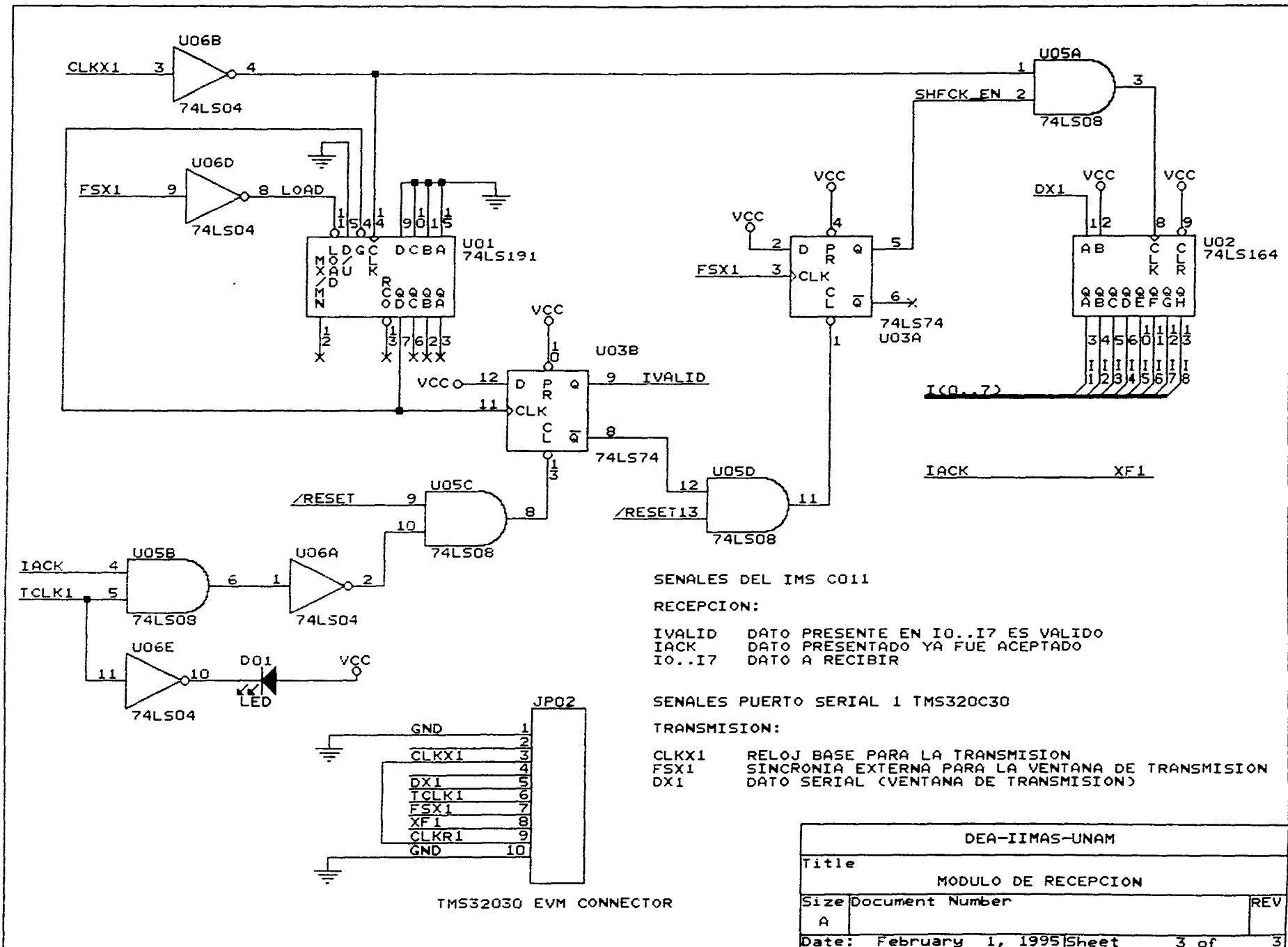
# de componentes	Descripción
S01	Push Button
JP01	Mini DIN 8 pines
JP02	EVMTMS30 Módulo de Evaluación
RN01	L65C181/391 Red de resistencias
C01	Capacitor 15 μF
C02	Capacitor 1.0 μF
D01	Led
R01	Resistencia 1K
R02	Resistencia 3.3K
U01	74LS191
U02	74LS164
U03	74LS74
U05	74LS08
U06	74LS04
U07	74LS04
U08	74LS32
U09	74LS08
U10	74LS74
U11	74LS165
U12	74LS191
U13	DS8921 Driver diferencial & receiver
U14	IMS C011 Link adaptor
Y01	Oscilador 5 MHz



TRANSPUTER SERIAL LINK INTERFACE

DEA-IIMAS-UNAM		
Title		
MODULO LINK ADAPTOR		
Size	Document Number	REV
A		
Date: February 1, 1995		Sheet 1 of 3





Referencias

Computer System Architects, 1990. Transputer Education Kit, User Guide, Theory of Operation, Installation, Schematics.

Computer System Architects, 1990. Transputer Education Kit, Occam and the Transputer, A Workbook.

García Nocetti, D. F., Solano J., Martínez J. Heterogeneous Architecture for Parallel Real-Time Spectral Estimation in Doppler Blood Flow Instrumentation. Proceedings of the International Conference on Control '94. 21-24 March 1994, University of Warwick, Coventry, UK.

Inmos, 1989. The Transputer Databook.

Martínez, J., Ramos, D. "Diseño e Implementación de un Nodo Heterogéneo de Procesamiento Digital de Señales utilizando un Transputer y un DSP". XVI Congreso Nacional Académico de Ingeniería Electrónica. 24-28 Octubre, 1994 Chihuahua, México.

National Semiconductor Corporation, 1987. LS/S/TTL Logic Databook.

Papamichalis, P., 1990. Digital Signal Processing Applications with the TMS320 Family, Volume 3. Ed. Prentice Hall and Digital Signal Processing Series Texas Instruments.

Texas Instruments, 1990. TMS320C30 Evaluation Module, Technical Reference.

Texas Instruments, 1992. TMS320C3x User's Guide.

6 Desarrollo de una aplicación con fines evaluativos

Introducción

El desarrollo de esta aplicación con el fin de evaluar el desempeño del nodo de procesamiento heterogéneo obedece al trabajo de investigación en el área de procesamiento en paralelo con aplicaciones en tiempo real, en el área de enfermedades cardiovasculares realizado en el Departamento de Electrónica y Automatización del IIMAS de la Universidad Nacional Autónoma de México.

Actualmente el incremento de enfermedades ocasionadas por la variación del flujo sanguíneo traen como resultado un gran número de muertes en la población. La detección temprana de enfermedades cardiovasculares, cerebrovasculares y la falta de suministro sanguíneo en el feto, entre otras, puede lograrse mediante el uso de *detectores de ultrasonido Doppler* [Sonicaid, 1989].

Las técnicas de ultrasonido Doppler se han usado extensivamente en enfermedades causadas por oclusiones o estrechamiento en las arterias, ocasionando que la distribución de la velocidad del flujo sanguíneo sea alterada. La medida de esta alteración y su relación con el grado de estenosis (estrechez patológica, congénita o accidental de un orificio o conducto) resulta de gran interés para el uso del análisis espectral de la señal Doppler en valoraciones no invasivas de las enfermedades arteriales. Para extraer información en términos cuantitativos de una señal Doppler, se debe establecer un esquema de procesamiento de señales digitales, derivado preferentemente de un modelo que sea una buena representación del proceso de generación de la señal.

La mayoría de los sistemas de ultrasonido Doppler comerciales emplean la transformada rápida de Fourier FFT como una herramienta en el procesamiento de señales reales. Las estenosis de moderadas a severas pueden ser detectadas de esta forma, sin embargo, los efectos de estenosis leves son difíciles de observar por esta técnica, por lo cual,

algunos métodos alternativos basados en estimación espectral, particularmente los métodos paramétricos (no-convencionales) pueden dar un incremento bastante significativo en la resolución dentro del espacio de tiempo y frecuencia.

Para la detección de enfermedades cardiovasculares por medio del procesamiento de señales Doppler, se han realizado diversos trabajos, algunos de los cuales han implementado arquitecturas basadas en transputers para obtener la estimación espectral de la señal Doppler, dando excelentes resultados en el manejo de operaciones en paralelo, pero mostrando deficiencias en las operaciones enfocadas al procesamiento de señales [García Nocetti, 1993]. También se han desarrollado arquitecturas en paralelo utilizando DSPs, las cuales resultan ser buenas en el cálculo de algoritmos como la FFT, correlación, etc., pero no así para el manejo de procesos en paralelo. El utilizar técnicas espectrales no convencionales para el procesamiento de la señal Doppler requiere de operaciones matemáticamente intensivas, lo cual puede llevarse a cabo en tiempo real con el uso de procesadores en paralelo. Por lo anterior, se ubicó el uso del **nodo de procesamiento heterogéneo** dentro de esta aplicación como una herramienta de procesamiento con amplias posibilidades de éxito, ya que es factible realizar el procesamiento de señales con el DSP y el manejo del procesamiento en paralelo con transputers.

6.1 Descripción de la aplicación

Con el fin de evaluar el nodo de procesamiento heterogéneo a partir de técnicas de análisis espectral en flujo Doppler (flujometría Doppler) es necesario manejar algunos conceptos como el efecto Doppler y los métodos de estimación espectral (convencionales y no-convencionales).

6.1.1 Efecto Doppler

El diagnóstico por ultrasonido ha jugado un papel muy importante en diversas áreas de la ciencia, particularmente en Obstetricia y Cardiología a partir de los años sesentas. El ultrasonido Doppler por su parte, no es, sino hasta los años ochenta, un método establecido de análisis clínico. Las razones de esta diferencia de aceptación son varias, destacando la dificultad de entender la generación del corrimiento Doppler en la

frecuencia causado por una estructura en movimiento y el tratamiento o procesamiento previo de los datos antes de que éstos sean desplegados [Atkinson, 1982].

Las técnicas de ultrasonido Doppler son importantes en la detección no invasiva y medición de la velocidad de estructuras en movimiento, particularmente de las células sanguíneas en el cuerpo humano. Se usan frecuentemente para valorar enfermedades circulatorias por la detección de anomalías en arterias, venas y corazón [Ruano, 1992] . La **figura 6.1** muestra un área de oclusión por la presencia de una placa arterial la cual ocasiona una reducción en el flujo y la presión sanguínea, ocasionando que cuando el diámetro no es suficiente, se emitan pequeñas partículas como resultado de la naturaleza pulsátil del flujo sanguíneo y por la flexibilidad de las paredes de la arteria. Estas partículas pueden bloquear zonas de la arteria ocasionando embolias (obstrucción brusca de un vaso sanguíneo arterial, venoso o capilar por un coágulo sanguíneo), cuando se trata de arterias que proporcionan sangre al cerebro.

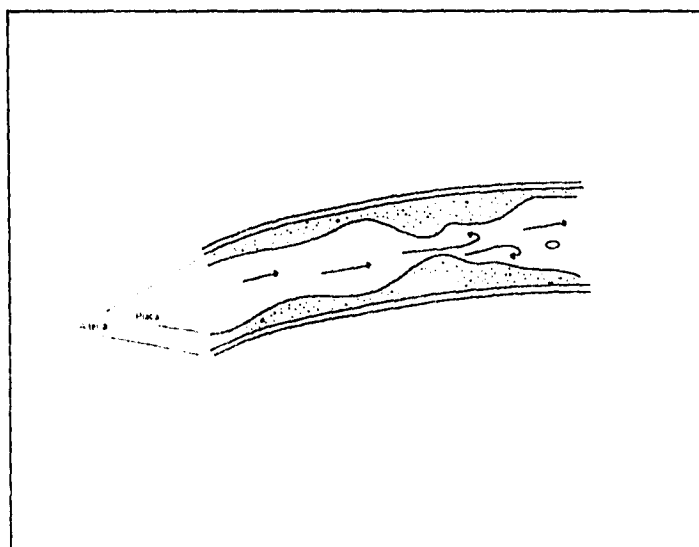


Figura 6.1 Placa en la pared de una arteria.

Para observar el estado hemodinámico (movimiento de la sangre) de la arteria, se requiere medir la velocidad del flujo sanguíneo y la forma más común de lograrlo, usando ultrasonido, es por medio del efecto Doppler.

Los instrumentos de ultrasonido Doppler miden la velocidad de la sangre detectando el corrimiento en frecuencia de la señal del ultrasonido generado por el movimiento de la sangre. Los instrumentos de ultrasonido Doppler más comunes son el de onda continua y el de onda pulsada. La diferencia entre uno y otro depende de la forma en la que la señal es transmitida, en forma continua o en modo ráfaga. Son preferibles los aparatos de ultrasonido Doppler pulsado debido a que es más fácil distinguir la señal de las arterias a diferentes niveles [Ruano, 1992].

El corrimiento en frecuencia de una señal Doppler se obtiene de un demodulador Doppler el cual combina la señal recibida con la señal de referencia (a la frecuencia transmitida) en un dispositivo no lineal (generalmente un multiplicador) para pasarla después por filtros paso-baja, los cuales eliminan las componentes de la señal por arriba de la frecuencia transmitida [Fish, 1992].

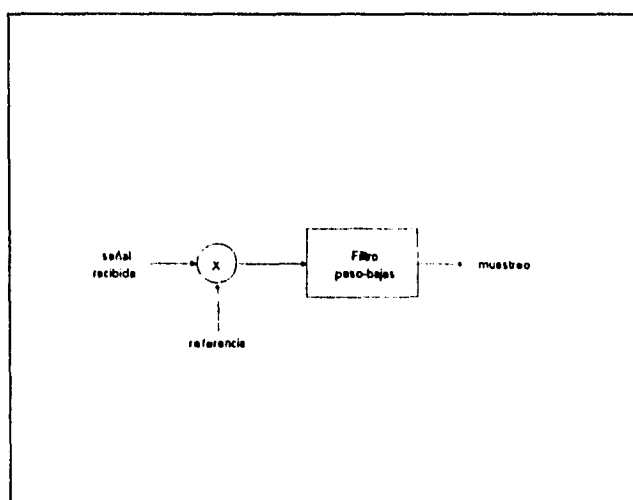


Figura 6.2 Demodulador Doppler.

La frecuencia f_d de la señal Doppler de un dispersor (*scatterer*) es proporcional a la velocidad del dispersor y a los cosenos de los ángulos entre la dirección de movimiento y la dirección de los vectores de los transductores de transmisión y recepción (figura 6.3).

$$f_d = -\frac{(\bar{k}_t + \bar{k}_r) \cdot \bar{v}}{2\pi}$$

$$= -\frac{v}{\lambda}(\cos \theta_t + \cos \theta_r)$$

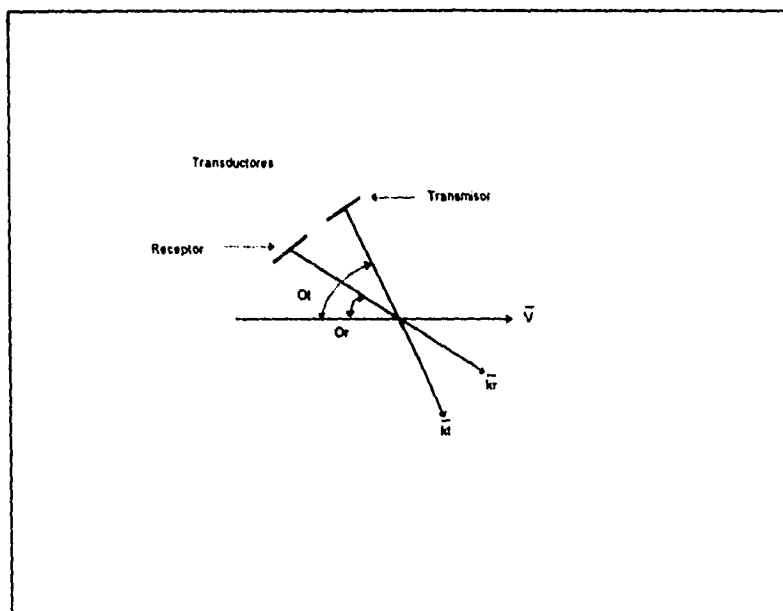


Figura 6.3 Relación entre el vector de velocidad reflejado y los vectores de los transductores de transmisión y recepción.

El efecto de una estenosis (o lesión) en la velocidad del flujo sanguíneo es mostrado en la figura 6.4. Los remolinos y turbulencias son creados adelante de la lesión, pudiéndose detectar y cuantificar por medio del ultrasonido Doppler. Las estenosis más significativas alteran la forma de onda y ancho de banda de la velocidad de la sangre. Este cambio en la forma de onda y ancho de banda es usado en el diagnóstico de lesiones que son accesibles directamente sólo por ultrasonido.

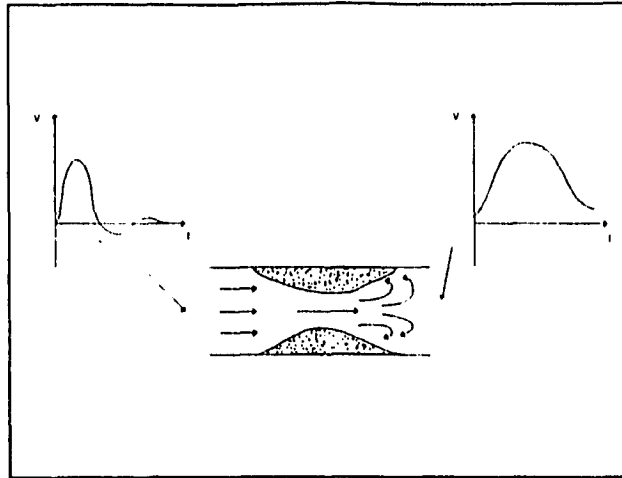


Figura 6.4 Efecto de estenosis en una arteria.

Una amplia gama de modelos para el proceso de generación de la señal Doppler han sido publicados, y todos estos concluyen que la señal Doppler en un flujo no turbulento es aleatoria con una función de probabilidad Gaussiana (pdf) [Mo and Cobbold, 86] y que su espectro es determinado por el intervalo de velocidades de flujo que pasa a través de un volumen muestra y por las características de éste. En un flujo turbulento, la señal Doppler incrementa su potencia y este aumento en retrodispersión (*backscatter*) se atribuye a un incremento de las fluctuaciones aleatorias de los parámetros acústicos, además se presentan variaciones en la densidad de la arteria como resultado de las aceleraciones locales dentro del flujo turbulento (figura 6.5).

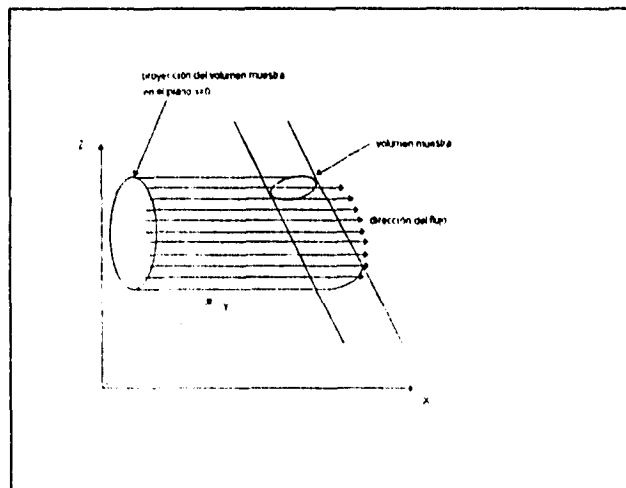


Figura 6.5 Líneas de flujo que contribuyen al espectro de la señal Doppler.

6.1.2 Espectro de la señal Doppler

El espectro de las señales Doppler en el flujo sanguíneo es determinado, en parte, por las características del volumen muestreado así como por el intervalo de las velocidades del flujo que pasa a través del volumen muestra en un flujo no turbulento. Con volúmenes de muestras relativamente pequeños y en arterias grandes, se obtiene un espectro Doppler reducido como se muestra en la **figura 6.6a**. Cuando el volumen muestreado está en una región turbulenta el ancho del espectro de la señal Doppler o la gama de frecuencias Doppler es incrementada por dos mecanismos (**figura 6.6b**). Primero, el aumento de la velocidad sanguínea y segundo el aumento de los ángulos entre la dirección de movimiento y el vector del ultrasonido (ver ecuación 1).

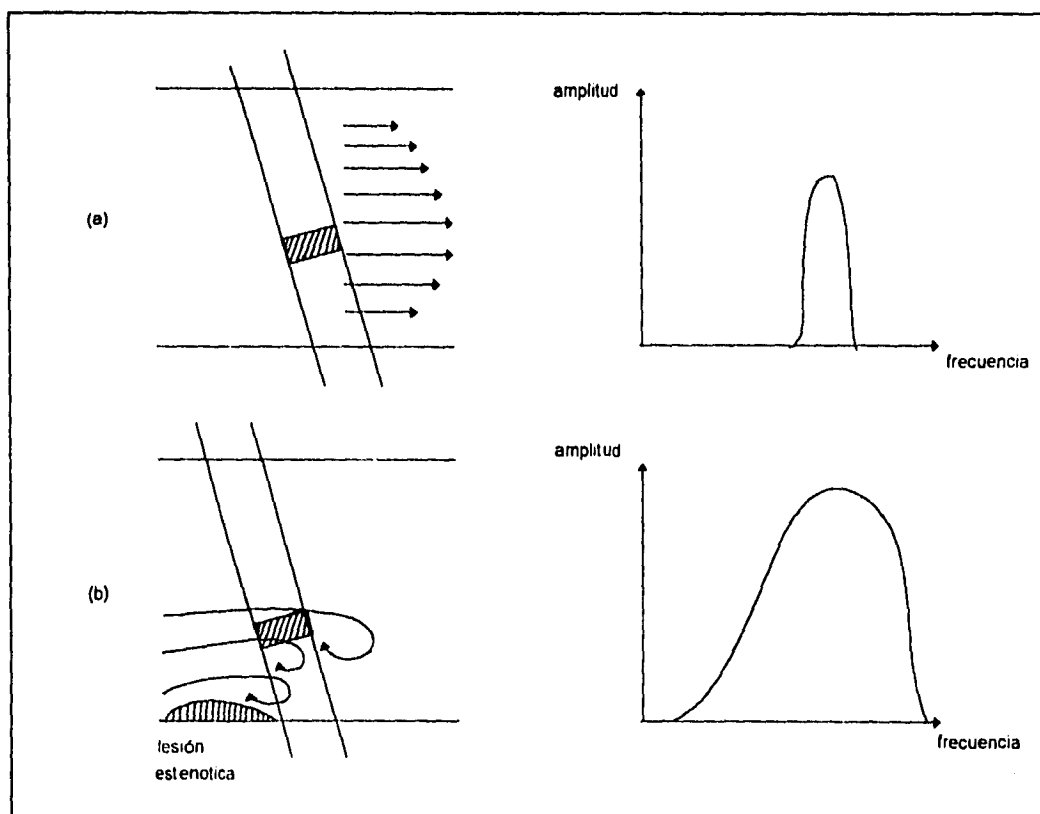


Figura 6.6 Espectro de frecuencia Doppler
a) Flujo laminar y b) Flujo turbulento.

Como el flujo en las arterias es pulsátil, se requiere un despliegue espectral en tiempo real para mostrar la variación en frecuencia de la señal Doppler, media o máxima, y la variación del ancho del espectro a lo largo de cada ciclo cardíaco (forma de onda de la velocidad). La intensidad en la frecuencia y el tiempo indican la potencia de los componentes de frecuencia de la señal Doppler en un tiempo particular. De esta manera, cuando el tamaño de una lesión se incrementa, el espectro se amplifica como resultado de la turbulencia que ocurre primeramente durante la desaceleración en la fase de la sístole y después se expande a todo el ciclo cardíaco. Por consiguiente, cuando la estenosis llega a ser severa, la velocidad del flujo sanguíneo en la región estenótica aumenta y sale en forma de chorro. Este aumento en la velocidad da una elevación en el corrimiento de la frecuencia Doppler que es muy útil para un diagnóstico médico (figura 6.7).

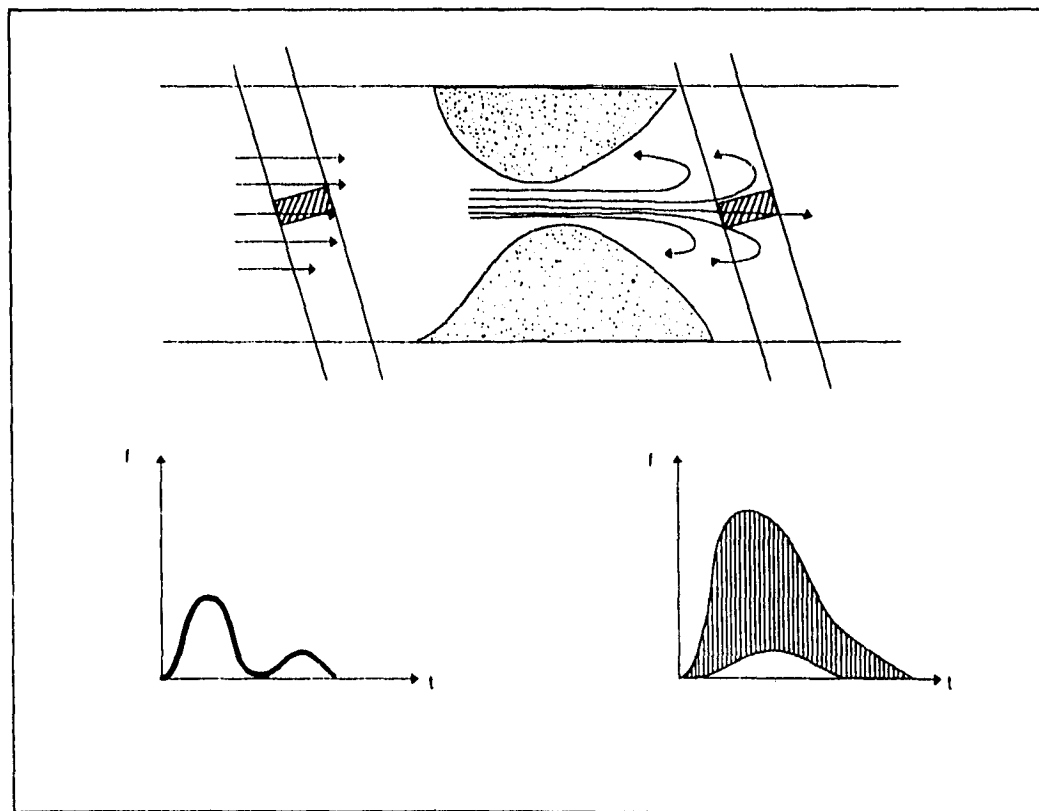


Figura 6.7 Cambio del espectro en tiempo real después de una lesión.

La **figura 6.8** ilustra el proceso convencional del análisis de la señal Doppler. El espectro de la señal Doppler dentro de segmentos secuenciales o traslapados, con un tiempo de duración de 2-30 ms es calculado multiplicando la señal de cada segmento de tiempo por una ventana reducida para suprimir los lados de los lóbulos en el espectro [Papoulis, 84]. Posteriormente, se calcula el módulo cuadrado de la transformada de Fourier (generalmente se usa el algoritmo de la Transformada Rápida de Fourier) de la ventana. Como la señal Doppler es aleatoria, los cálculos individuales del espectro también son aleatorios. Si se requiere un cálculo aproximado de la frecuencia media y el ancho espectral, se promedian los cálculos espectrales de los segmentos con tiempos similares en un número de ciclos consecutivos de onda de la arteria como se muestra en la **figura 6.9**.

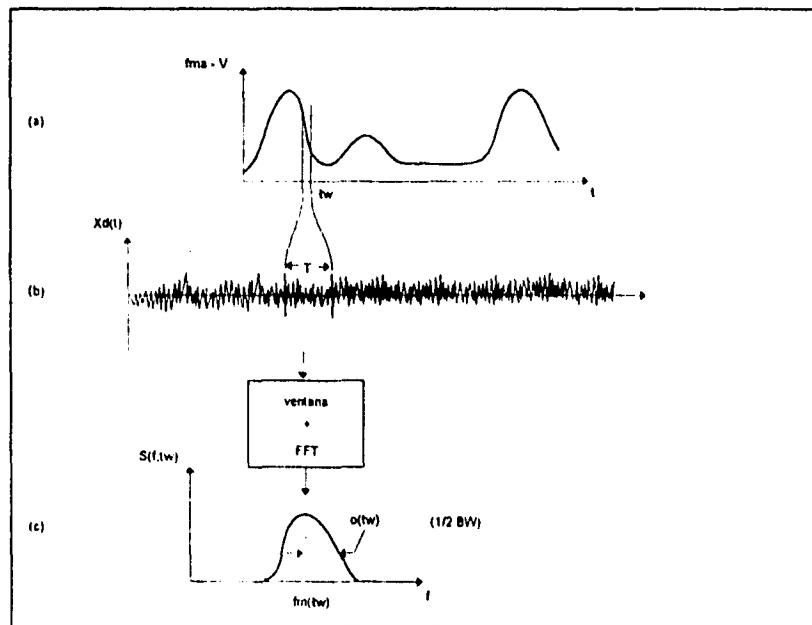


Figura 6.8 Método convencional de análisis espectral
 (a) La frecuencia media f_m corresponde a la forma de onda de velocidad V , (b) Señal Doppler y (c) $S(f, t_w)$ es $S_{XS}(f)$ en el tiempo t_w .

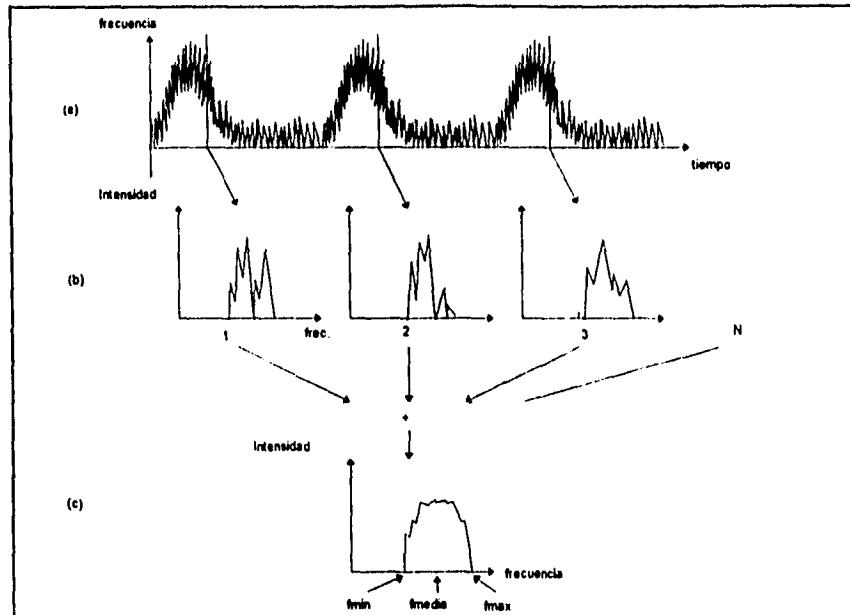


Figura 6.9 Promedio espectral (a) Despliegue espectral en tiempo real, (b) Espectro en ciclos consecutivos y (c) Promedio espectral.

Si una ventana de la señal es:

$$x_w(t) = w(t)x_d(t)$$

donde $x_d(t)$ es la señal aleatoria Doppler y $w(t)$ es la función de la ventana, entonces el cálculo del espectro de potencia de $x_d(t)$ es el espectro de energía $x_w(t)$ cuyo valor es

$$\xi |S_x(f)| = |W(f)|^2 * S_x(f)$$

donde $\xi |$ es el operador de índice, $W(f)$ es la transformada de Fourier de $w(t)$ y $S_x(f)$ es el espectro de potencia de $x_d(t)$. El espectro es ampliado por el espectro de energía de la función de la ventana, teniendo de esta forma que el ancho de éste es inversamente proporcional al ancho de $w(t)$.

6.1.2 Métodos convencionales

Los métodos de estimación espectral basados en el cálculo de la transformada de Fourier de la señal o de la función autocorrelación dentro de un intervalo medible son conocidos como **métodos convencionales** [Ruano, M., 1992].

La transformada rápida de Fourier (FFT) es una herramienta muy importante usada en aplicaciones de procesamiento de señales (DSP). Fue desarrollada por Cooley y Tuckey dando un fuerte impulso al establecimiento del DSP como una disciplina independiente.

Recientemente se han desarrollado algoritmos eficientes para incrementar la velocidad de la FFT y decrementar los requerimientos de memoria.

La transformada de Fourier de una señal analógica $x(t)$ esta dada por:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t} dt$$

la cual determina el contenido de frecuencia de la señal $x(t)$, es decir, para cada frecuencia, la transformada de Fourier $X(\omega)$ establece la aportación de frecuencia de una senoide en la composición de la señal $x(t)$. Cuando los cálculos se realizan en una computadora digital, la señal $x(t)$ es muestreada a intervalos discretos de tiempo. Si la señal de entrada es digitalizada, una secuencia de números $x(n)$ está disponible en lugar de la señal continua $x(t)$. Entonces, la transformada de Fourier toma la forma:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}$$

La transformada resultante $X(e^{j\omega})$ es una función periódica de ω y requiere ser calculada sólo por un período. El cálculo real de la transformada de Fourier de un flujo de datos presenta dificultades debido a que $X(e^{j\omega})$ es una función continua en ω . Como la transformada debe ser calculada en puntos discretos, las propiedades de la transformada de Fourier llevan a la definición de la transformada discreta de Fourier (DFT), la cual es representada por la siguiente expresión:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j \frac{2\pi kn}{N}}$$

Si $x(n)$ tiene N puntos $x(0), x(1), \dots, x(N-1)$, su representación en el dominio de la frecuencia está dado por un conjunto de N puntos $X(k), k = 0, 1, \dots, N-1$.

Entonces la ecuación anterior queda de la siguiente manera:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}$$

donde $W_N^{nk} = e^{-j2\pi nk/N}$. El factor W_N es conocido como el factor de giro (*twiddle*). Los requerimientos computacionales de la DFT incrementan rápidamente al aumentar el bloque de datos (N), teniendo esto un fuerte impacto en la eficiencia de los sistemas en tiempo real. Este problema ha disminuído con el desarrollo de algoritmos más rápidos conocidos como Transformada Rápida de Fourier (FFT). Con un algoritmo FFT, la carga computacional se incrementa lentamente al aumentar el número de datos y se mide en términos del número de multiplicaciones y sumas requeridas.

La definición de la FFT es idéntica a la de DFT, sólo cambia el método de cálculo. Para lograr la eficiencia de un algoritmo FFT, es importante que N sea un número compuesto. Generalmente la longitud de N es una potencia de 2: $N = 2^M$, y el algoritmo se convierte en una repetición de una transformación elemental conocida como mariposa (*butterfly*). Si N no es una potencia de 2, a la secuencia $x(n)$ se le añaden ceros hasta lograr que la longitud total sea una potencia de 2.

Diferentes formas de la FFT se han desarrollado con el fin de obtener mayor eficiencia en su cálculo. Para el caso de una secuencia de números reales han surgido algoritmos más sofisticados. Como ya se mencionó, el elemento principal de la FFT lo constituye la mariposa la cual se muestra en la siguiente figura:

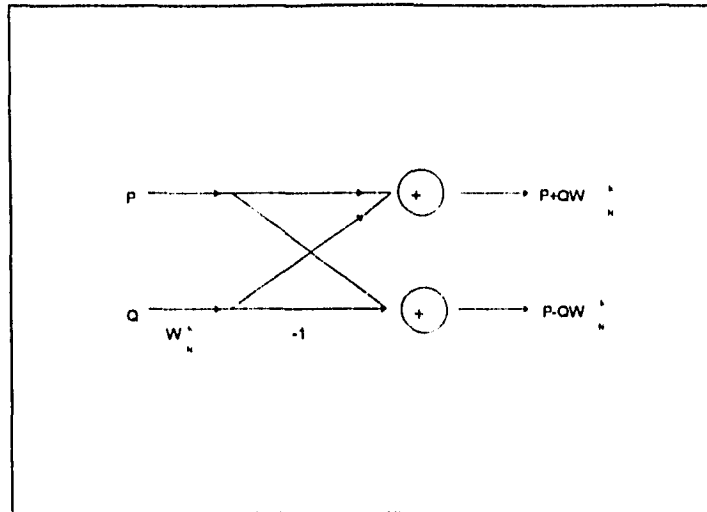


Figura 6.10 Mariposa Radix-2.

Si las entradas en la mariposa son números complejos P y Q , la salida serán dos números complejos P' y Q' . De esta manera:

$$P' = P + QW_N^k$$

y

$$Q' = P - QW_N^k$$

Los valores P , Q , P' y Q' representan diferentes puntos del arreglo a ser transformado y pueden o no ocupar localidades adyacentes en el mismo. Cuando se efectúa un cálculo en la misma localidad de memoria (*in-place*), el resultado P' será sobrescrito en P y Q' en Q . El valor W_N^k representa de nuevo el factor de giro y su exponente es determinado por la posición de la mariposa correspondiente en el algoritmo FFT. Usando la notación de la mariposa en la siguiente gráfica se muestra el flujo gráfico de una FFT de 8 puntos.

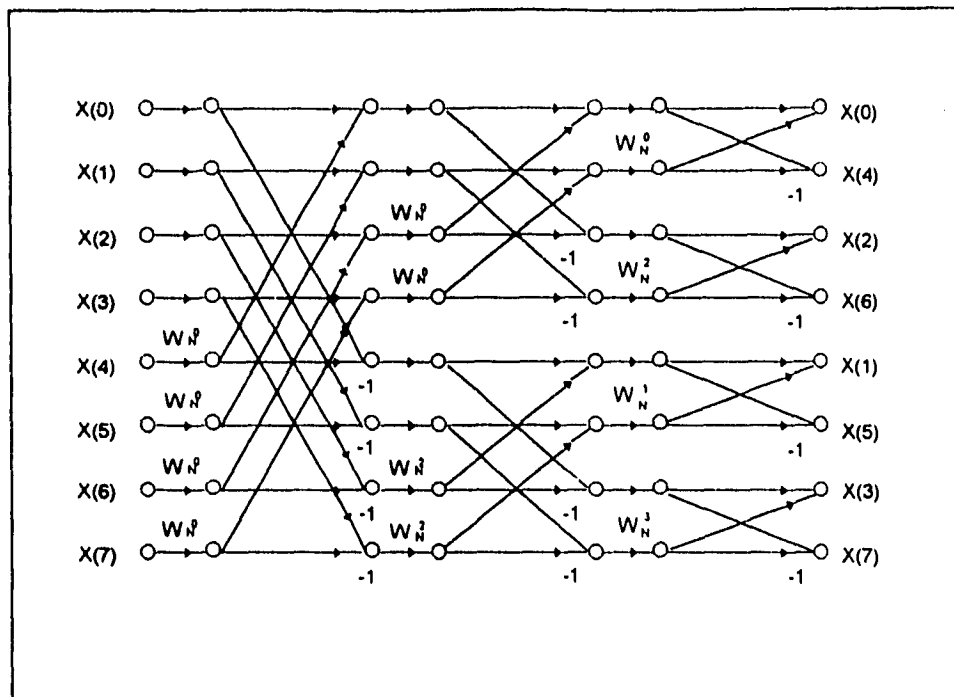


Figura 6.11 Ejemplo de una FFT con 8 puntos.

En la figura anterior, la secuencia de entrada $x(n)$ sigue un orden correcto, mientras que la salida $X(k)$ está fuera de orden. Esta reordenación ocurre de manera sistemática llamándose orden *bit-reversed*, que consiste en expresar los índices de una secuencia en forma binaria para después invertir estos números, obteniendo como resultado el orden que ocupan estos puntos. Por ejemplo, el valor $X(3)$ ocupa la sexta posición en la salida (contando a partir de cero). En forma binaria $3_{10} = 011_2$ y al realizar el bit-reversed se obtiene $102 = 6_{10}$, que es la posición que ocupa $X(3)$ a la salida. De igual forma la tercera posición es ocupada por $X(6)$ y para tener el orden correcto a la salida, únicamente se intercambian estos dos números.

De esta manera, si la secuencia de entrada $x(n)$ es reordenada para aparecer en forma bit-reversed, la salida $X(k)$ aparece en el orden correcto.

Cuando se usan procesadores de propósito especial como los DSPs (caracterizados por operaciones de multiplicación y acumulación en un sólo ciclo de instrucción, modo de direccionamiento bit-reversed e

instrucciones con un alto grado de paralelismo), la estructura del algoritmo es de igual importancia que la complejidad aritmética. Por ello, para este tipo de procesadores, los algoritmos FFT Radix-2 y Radix-4 son más eficientes en términos de velocidad y precisión.

El radix de la FFT representa el número de entradas que son combinadas en una mariposa. La FFT se conceptualiza mejor con el algoritmo Radix-2, sin embargo, se utilizan algoritmos Radix de mayor orden, porque se logra reducir el número de cálculos al incrementarse el mismo.

Por otro lado, la FFT se utiliza en la estimación espectral de potencia para obtener el periodograma (fotografía instantánea del espectro de potencia). El promedio de una serie de periodogramas de la señal es usado como el estimador del espectro de la señal en un tiempo particular. Los parámetros del promedio de la estimación espectral de potencia son:

1. *Frecuencia de muestreo*: determina la máxima frecuencia a ser calculada.
2. *Longitud de la FFT*: determina la resolución (diferencia más pequeña de frecuencia detectable).
3. *Ventana*: determina la cantidad de pérdida del espectro y afecta a la resolución y al ruido.
4. *Traslape entre espectros sucesivos*: determina la precisión del cálculo.
5. *Número de espectros promediados*: determina la velocidad máxima de cambio del espectro y también los efectos del ruido.

6.1.3 Métodos no convencionales o paramétricos

Los métodos convencionales de estimación espectral usan la transformada de Fourier con ventanas de datos. Cuando una ventana se amplifica, causa una pérdida en la resolución del espectro. Esto se debe esencialmente a que se considera que la señal fuera de la ventana tiene un valor cero, a diferencia de los estimadores paramétricos, los cuales consideran de gran interés el segmento de señal fuera de la ventana, ya que la señal tiene una forma que puede ser determinada, mejorando considerablemente la resolución espectral. Esta ventaja ha permitido el uso de los estimadores espectrales paramétricos en el análisis de señales Doppler.

El modelo general de los estimadores paramétricos se muestra en la **figura 6.12**. La señal generada en la fuente de banda ancha (señal de ruido blanco) es alimentada a un filtro digital cuya respuesta en frecuencia puede ser alterada ajustando los parámetros del filtro. La salida de este filtro es comparada con la señal muestreada, cuyo espectro es requerido y los parámetros del filtro son alterados hasta que las funciones de autocorrelación de las dos señales sean iguales. En este caso, el espectro de la señal a la salida del filtro es el mismo que el espectro de la señal de interés y como la señal de entrada al filtro tiene una densidad espectral constante, la forma de la respuesta en frecuencia del filtro es igual al espectro de potencia de la señal.

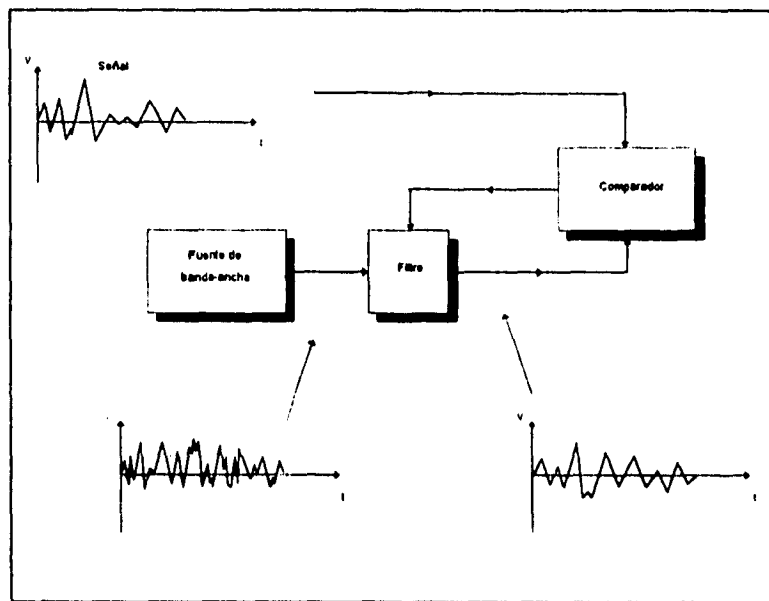


Figura 6.12 Modelo general de los estimadores paramétricos.

Los parámetros del filtro pueden ser calculados con la función de autocorrelación de la señal de interés. El tipo del filtro y el método para determinar sus parámetros determinan el tipo de estimador.

De esta manera, la estimación espectral para el modelado se compone de tres pasos. El primer paso consiste en seleccionar un modelo. El segundo estima los parámetros del modelo seleccionado usando los datos disponibles y el tercero es la obtención del estimador espectral sustituyendo los parámetros del modelo estimado en la función PSD (Densidad de Potencia Espectral), asociada al modelo [García Nocetti, 1993].

Parte del proceso de selección de los estimadores paramétricos es la elección del orden del filtro. El tamaño (u orden) del filtro se determina por el número de polos (p) y ceros (q) en su función de transferencia. Si $p=0$ es un filtro que tiene únicamente ceros o movimiento promedio (MA), si $q=0$ es un filtro sólo con polos o autoregresivo (AR) y si p y q son diferentes de cero es un filtro autoregresivo-movimiento promedio (ARMA).

En general, la naturaleza de la señal Doppler de un instrumento Doppler pulsado sugiere un filtro con polos y por esta razón se consideran únicamente los estimadores AR y ARMA [Fish, P., 1992].

Finalmente, los métodos de estimación paramétrica mejoran potencialmente la resolución espectral a base de un mayor número de cálculos que habrán de realizarse en tiempo real, requiriendo para ello el uso del procesamiento en paralelo. Algunos trabajos recientes [García Nocetti, 1993] han demostrado que con un número pequeño de procesadores en paralelo un estimador paramétrico moderadamente complejo (covarianza modificada AR) puede ejecutarse en tiempo real.

6.2 Software de la aplicación

Para evaluar el desempeño del Nodo de Procesamiento Heterogéneo (Capítulo 5) se desarrolló un programa que consiste en el despliegue gráfico de un espectrograma de ultrasonido Doppler en dos dimensiones. Como técnica de estimación espectral se eligió la FFT, aún cuando los métodos de estimación espectral paramétricos, como el algoritmo de covarianza modificada, dan una mejor resolución para señales Doppler [Ruano, M., 1992].

Con el fin de poder medir y comparar los resultados generados por el Nodo de Procesamiento Heterogéneo, se desarrolló paralelamente un programa para una arquitectura homogénea formada por dos transputers y cuyas funciones son básicamente las mismas que para el programa con la arquitectura heterogénea.

Los programas fueron realizados en lenguaje de programación C concurrente y en lenguaje ensamblador para el TMS320C30. Las funciones del lenguaje C concurrente para desarrollar programas en paralelo y los pasos para generar un archivo ejecutable están dados en el Apéndice C. Asimismo, los pasos necesarios para compilar y encadenar un programa en lenguaje ensamblador del TMS320C30 y su conjunto de instrucciones se muestran en el Apéndice B.

6.2.1 Programa para el sistema de procesamiento homogéneo

El sistema de Procesamiento Homogéneo consiste de dos nodos (transputers): el nodo host (**nodo_1**) y el nodo de muestreo y procesamiento (**nodo_2**). El nodo host es físicamente el sistema de desarrollo del transputer (TEK805), mientras que el nodo de procesamiento y muestreo es un transputer adicional o TRAM (*TRAnspuTer Module*). El TRAM es una tarjeta o módulo con transputer, que tiene la facilidad de integrar memoria de diferentes tamaños y periféricos con el fin de poder construir sistemas basados en transputers de una forma sencilla [Inmos, 1991]. Este sistema recibe la señal Doppler de un archivo de datos sintéticos (figura 6.13).

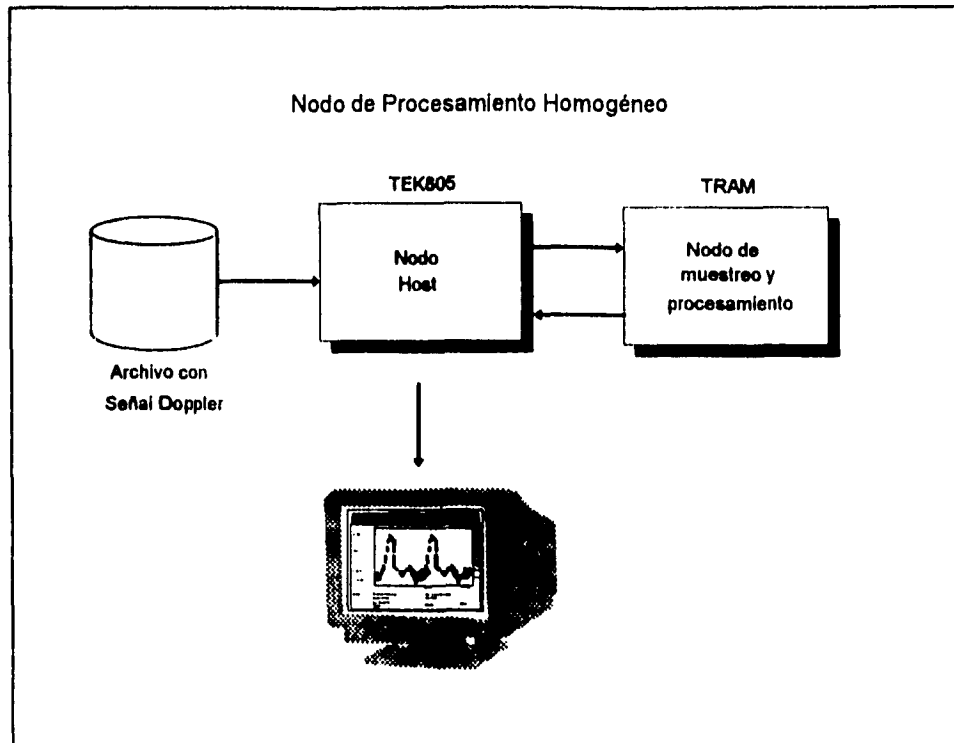


Figura 6.13 Nodo de procesamiento homogéneo.

Para cada nodo se desarrolló un programa. El programa para el nodo_1 cuenta con un menú principal el cual le permite al usuario seleccionar 3 funciones: cálculo de parámetros, despliegue de espectrograma y finalizar la sesión. Cuando se selecciona la función *Parámetros* es posible modificar el valor de los siguientes parámetros:

- Longitud de FFT (default 256),
- Frecuencia de muestreo (default 12.8 KHz),
- Período cardíaco (default 700 ms),
- Número de colores (default 8) y
- Eje de tiempo (default 0.20 seg/Div).

Si se selecciona la función *Espectrograma*, comienza el despliegue del espectro de la señal Doppler con los valores de default de los parámetros o con los nuevos valores seleccionados. El eje vertical del espectrograma indica frecuencia y el eje horizontal tiempo. La función espectrograma recibe datos del nodo_2, los cuales son agrupados en ventanas de acuerdo a la longitud de la FFT deseada. Dichos valores son codificados por el número de colores seleccionado. Antes de empezar a recibir los

datos de parte del nodo_2, esta función envía los valores de los parámetros a dicho nodo. Si se desea terminar el despliegue del espectrograma se presiona la tecla ESC o si se desea congelar el despliegue de la señal se presiona la tecla H (*Hold*). Por último, para finalizar la sesión se selecciona la función *Fin* dentro del menú principal.

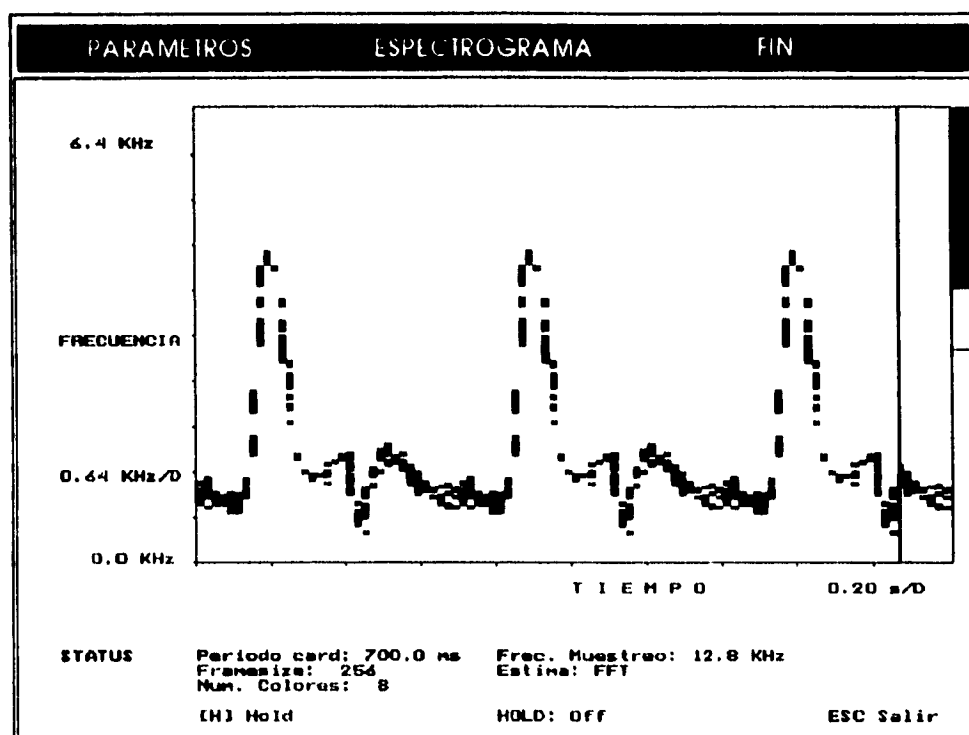


Figura 6.14 Pantalla de despliegue gráfico del espectrograma.

Cuando empieza la ejecución del programa del nodo_1, se lee el archivo que contiene los datos de la señal Doppler, los cuales son enviados al nodo_2. El envío y recepción de datos a los nodos se efectúa por medio de los canales de comunicación (*links*). Este programa también tiene la función de activar y desactivar el modo gráfico y el manejo del mouse para la selección de las funciones del menú principal y el cambio de parámetros. La siguiente figura (Figura 6.15) muestra a nivel de pseudo-código el listado del programa.

Para el nodo_2, el programa consta también de las tres funciones del nodo_1 en el menú principal, sólo que para la función *Parámetros* el programa no hace nada y únicamente espera que el usuario seleccione cualquiera de las otras dos funciones. Si se elige la función *Espectrograma* en el nodo_1, la función *Espectrograma* en el programa del nodo_2 espera que lleguen los parámetros del sistema y ejecuta de forma

concurrente la función de muestreo y la de procesamiento. En la función de *Muestreo* del nodo_2, se toman los datos del archivo de la señal Doppler enviados por el nodo_1 y se simula el muestreo de los datos con período de muestreo igual a $1/\text{frec_de_muestreo}$. Estos datos muestreados son enviados a la función *Procesamiento* donde se calcula con ellos la FFT y posteriormente la PSD (Densidad de Potencia Espectral). Por último los datos son enviados al nodo_1 para su despliegue gráfico. La Figura 6.16 representa el listado en pseudo-código del programa para el nodo_2.

```

Programa nodo_1
lee archivo datos señal Doppler
envía datos señal Doppler a nodo_2
activación modo gráfico y mouse
si la opción_menú_principal es
  Parámetros
    modifica parámetros
    longitud FFT
    frecuencia de muestreo
    periodo cardiaco
    número de colores
    eje del tiempo
  Espectrograma
    envía parámetros a nodo_2
    mientras no sea tecla_ESC
      gráfica señal Doppler
      recibe datos procesados de nodo_2
  Fin
    desactiva modo gráfico
fin de programa nodo_1
    
```

Figura 6.15 Programa nodo_1.

```

Programa nodo_2
recibe datos señal Doppler de nodo_1
si la opción_menú_principal es
  Parámetros
  Espectrograma
    recibe parámetros de nodo_1
    mientras no sea tecla_ESC
      PAR
        muestreo
        procesamiento
          calcula FFT
          calcula DPS
          envía datos procesados a nodo_1
    Fin
fin de programa nodo_2
    
```

Figura 6.16 Programa nodo_2.

6.2.2 Programa para el sistema con el nodo de procesamiento heterogéneo

El sistema con un nodo de procesamiento heterogéneo se muestra en la **Figura 6.17**. Consta de tres nodos de procesamiento: nodo host (**nodo_1**), nodo de muestreo (**nodo_2**) y nodo de procesamiento (**nodo_3**). Al igual que para el sistema de procesamiento homogéneo, físicamente el **nodo_1** corresponde al sistema de desarrollo del transputer (TEK805), el **nodo_2** a un TRAM adicional y el **nodo_3** al nodo de procesamiento heterogéneo.

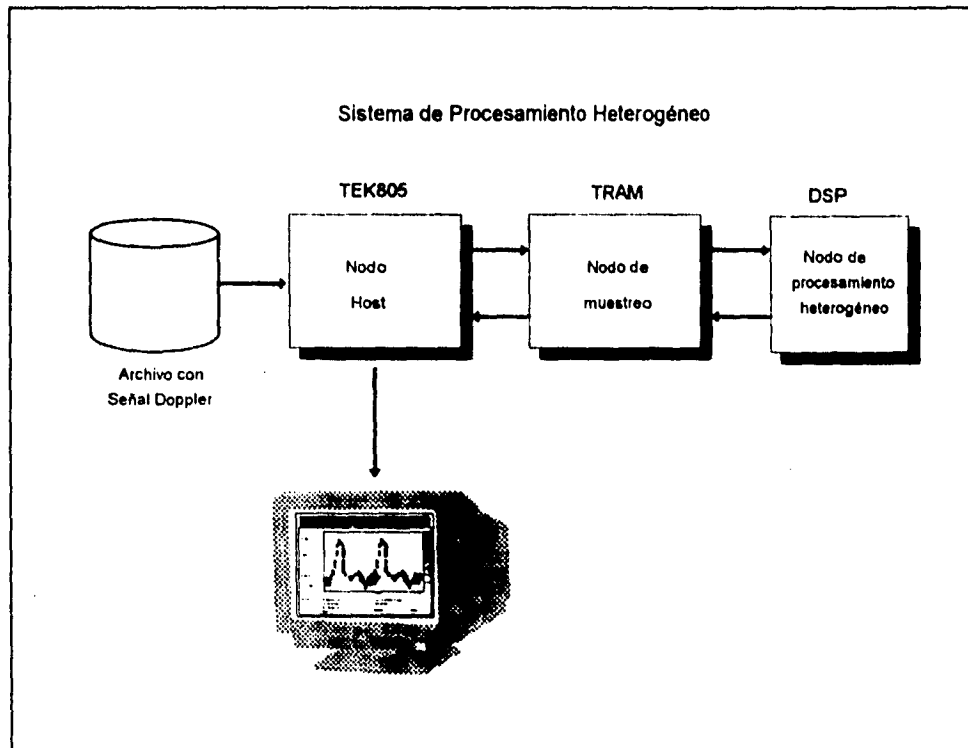


Figura 6.17 Nodo de procesamiento heterogéneo.

El programa para el **nodo_1** es el mismo que para el sistema homogéneo. El programa para el **nodo_2** básicamente es el mismo que para el **nodo_2** del sistema homogéneo. La diferencia radica en que la función de procesamiento ahora la realiza el **nodo_3**. Por lo tanto, el programa para el **nodo_2** únicamente realiza la simulación del muestreo de la señal,

manda los datos al nodo_3 y espera a que lleguen los resultados para enviarlos al nodo_1 y ser graficados por el mismo. Como el nodo_3 no es un elemento de procesamiento en paralelo (transputer) se maneja un byte de reconocimiento para sincronizar el arranque del nodo_3 (DSP). La figura 6.18 presenta el pseudo-código para el nodo_2.

```

Programa nodo_2
envía byte de reconocimiento a nodo_3
recibe datos señal Doppler de nodo_1
si la opción_menú_principal es
  Parámetros
  Espectrograma
    recibe parámetros de nodo_1
    mientras no sea tecla_ESC
      PAR
        muestreo
        envía longitud de FFT
        envía datos muestreados a nodo_3
        recibe datos procesados de nodo_3
        envía datos procesados a nodo_1
  Fin
fin de programa nodo_2

```

Figura 6.18 Programa nodo_2.

El programa del nodo_3 está escrito en lenguaje ensamblador del TMS320C30. Este programa tiene como función principal realizar el cálculo de la FFT y la PSD. Primeramente, espera la llegada de un byte de reconocimiento que permite indicar que el sistema está listo para procesar. Cuando esto sucede, queda en espera de recibir la longitud de los datos a transformar por la FFT para posteriormente aceptar los datos a procesar. Como el formato para números en punto flotante del transputer (ANSI-IEEE 754-1985) y el DSP son diferentes, se manejó la conversión de estos formatos. Para ello se utilizó el código de las rutinas FMIEEE y TOIEEE escrito por Gary A. Sifton [Papamichalis, P., 1990]. Estas conversiones son bastante rápidas y utilizan una tabla de constantes definida al inicio del programa. Después de recibir los datos y hacer la conversión de formato IEEE a formato TMS320C30, se efectúa el cálculo de la FFT. Para ello se utilizó el programa realizado por Panos E. Papamichalis [Papamichalis, P., 1990]. Dicho programa efectúa el cálculo de la FFT con el algoritmo Radix-2 para números reales. Se utilizó esta rutina debido a que los datos manejados son números reales (señal-Doppler) y con el objeto de tener la implementación óptima de la transformada rápida de Fourier. A

continuación se presenta una tabla con el tiempo de ejecución para diferentes transformadas incluyendo la transformada de Hartley.

Tabla 6.1 Tiempos de ejecución para diferentes algoritmos de transformada (ms).

Transformada Tamaño	Radix-2 Complejos FFT	Radix-4 Complejos FFT	Radix-2 Real FFT	Radix-2 Real Inversa FFT	Transformada de Hartley
64	0.165	0.124	0.077	0.085	0.081
128	0.370	-	0.174	0.193	0.181
256	0.816	0.624	0.387	0.434	0.403
512	1.784	-	0.857	0.964	1.132
1024	3.873	3.040	1.879	2.124	2.430

Como se observa en la tabla los tiempos de ejecución para diferentes tamaños, el algoritmo Radix-2 real FFT ofrece un menor tiempo de ejecución.

Para eficientar la FFT, los factores de giro (*twiddle*) necesarios para el cálculo de transformada son almacenados en una tabla de longitud $5N/4$.

Una vez realizado el cálculo de la transformada de Fourier, se implementó la parte del código para la PSD. La PSD requiere un cálculo de una raíz cuadrada para la cual se utilizó la rutina de Gary A. Sitton [Papamichalis, P., 1990], que calcula dicha raíz con una precisión de 8 dígitos decimales. Por último, los datos son convertidos a un formato IEEE y son enviados de regreso al nodo_2. La figura 6.19 muestra el pseudo-código para el programa del nodo_3.

```

Programa nodo_3
recibe byte de reconocimiento a nodo_2
si se envían datos del nodo_2
    recibe longitud de FFT
    recibe datos muestreados de nodo_2
    conversión de formato IEEE a TMS320C30
    cálculo de FFT
    cálculo de PSD
    conversión de formato TMS320C30 a IEEE
    envío de datos procesados a nodo_2
si no espera por datos
fin de programa nodo_3
    
```

Figura 6.19 Programa nodo_3.

Los listados completos de todos los programas se muestran en el Apéndice D, al igual que los archivos de configuración de los nodos de procesamiento.

6.2.3 Resultados

Con los programas desarrollados se obtuvieron los tiempos de ejecución para los nodos de procesamiento heterogéneo y homogéneo (ver tablas 6.2 y 6.3). Estos tiempos se alcanzaron utilizando la función `time()` del lenguaje C concurrente, que proporciona el valor del reloj en tiempo real. Los tiempos fueron tomados para ventanas de datos de 64, 128, 256, 512 y 1024. Para ello se utilizó una computadora AT a 12 MHz, donde se instaló el sistema de desarrollo del transputer (TEK805) a 20 MHz y el Módulo de Evaluación del TMS320C30 a 30 MHz.

Tabla 6.2 Tiempos de procesamiento para el sistema homogéneo (ms).

Función	Ventana de datos				
	64	128	256	512	1024
Muestreo	5.440	10.816	21.632	43.072	86.080
Cálculo FFT Y PSD	12.993	26.560	54.400	111.488	228.224
Despliegue gráfico	9.530	13.496	17.380	21.312	55.936

Tabla 6.3 Tiempos de procesamiento para el sistema heterogéneo (ms).

Función	Ventana de datos				
	64	128	256	512	1024
Muestreo	5.440	10.816	21.632	43.072	86.080
Cálculo FFT Y PSD	1.965	3.950	7.939	15.961	32.087
Despliegue gráfico	9.664	13.824	17.408	21.302	55.768

De acuerdo a los resultados obtenidos, se observa que el tiempo de procesamiento correspondiente al cálculo de la FFT y PSD para el nodo de procesamiento heterogéneo tiene un decremento significativo, lo que lo hace la mejor alternativa. La figura 6.20 muestra estos tiempos de manera comparativa presentándose el factor de incremento en la velocidad de procesamiento. Por ejemplo, para la ventana de 256 datos el sistema heterogéneo es 6.85 veces más rápido que el homogéneo. También se observa que al aumentar el número de datos en la ventana,

aumenta este factor. Asimismo, el tiempo de procesamiento para la función de muestreo es el mismo para ambos nodos como se puede apreciar en las tablas 6.2 y 6.3.

Por lo anterior se puede concluir que el nodo de procesamiento heterogéneo tiene un mejor desempeño que el nodo homogéneo en cuanto a tiempo de procesamiento, utilizando métodos convencionales de estimación espectral (FFT) y que el procesamiento de una señal Doppler puede realizarse en tiempo real debido a que el cálculo de la FFT y PSD sumado al despliegue gráfico resulta ser menor que el tiempo de muestreo como es el caso de la ventana de 512 datos.

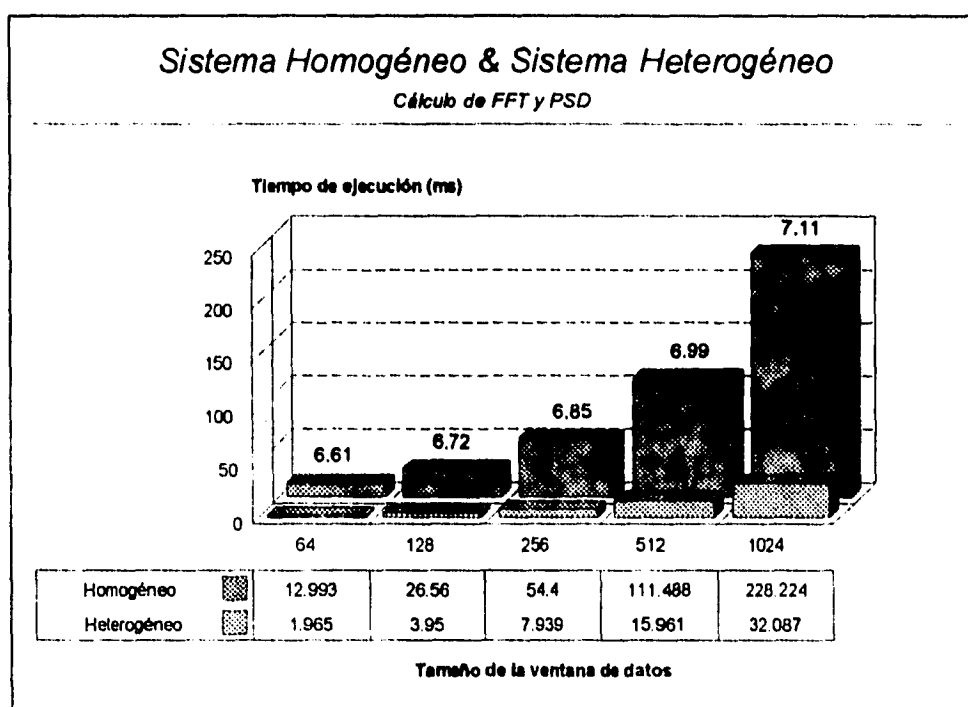


Figura 6.20 Comparación de tiempos de procesamiento (FFT y PSD) entre el sistema homogéneo y el heterogéneo.

Cabe mencionar, que el tiempo de procesamiento efectivo del cálculo de la FFT y PSD es mínimo en comparación con el tiempo de comunicación entre los procesadores. Las tablas 6.4 y 6.5 muestran el tiempo de transmisión y recepción para las ventanas de N datos (64, 128, 256, 512, 1024), considerando los tiempos t_{TX} y t_{RX} obtenidos al probar la interfaz del nodo de procesamiento heterogéneo (ver Capítulo 5).

Tabla 6.4 Tiempos de transmisión para las ventanas de datos.

Ventana de datos (números reales)	Tiempo de transmisión (μ s)
64	1177.6
128	2355.2
256	4710.4
512	9420.8
1024	18841.6

Nota:

$$t_{TX} = 4.6 \mu s / \text{byte}$$

$$\text{número real} = 4 \times 4.6 \mu s = 18.4 \mu s$$

Tabla 6.5 Tiempos de recepción para las ventanas de datos.

Ventana de datos (números reales)	Tiempo de transmisión (μ s)
64	537.6
128	1075.2
256	2150.4
512	4300.8
1024	8601.6

Nota:

$$t_{RX} = 4.2 \mu s / \text{byte}$$

$$\text{número real} = 4 \times 4.2 \mu s = 16.8 \mu s$$

De las tablas anteriores se observa que el tiempo de recepción se reduce debido a que sólo se transmiten N/2 datos del TMS30 al transputer y a que se tiene una respuesta más rápida en la lógica utilizada en el módulo de recepción.

Por otro lado, si se suman ambos tiempos, es decir el de transmisión y el de recepción, y se da esta suma en términos de porcentaje, se obtienen las siguientes tablas:

Tabla 6.6 Suma de los tiempos de transmisión y recepción para cada ventana de datos.

Ventana de datos (Números reales)	Tiempo de procesamiento ($t_{TX} + t_{RX}$) μ s	Tiempo de procesamiento (FFT y PSD) μ s	Tiempo de procesamiento Total μ s
64	1715.2	249.8	1965
128	3430.4	519.6	3950
256	6860.8	1078.2	7939
512	13721.6	2239.4	15961
1024	27443.2	4643.8	32087

Tabla 6.7 Suma de los tiempos de transmisión y recepción para cada ventana de datos en términos de porcentaje.

Ventana de datos (Números reales)	Tiempo de procesamiento ($t_{TX} + t_{RX}$) %	Tiempo de procesamiento (FFT y PSD) %
64	87.29	12.71
128	86.85	13.15
256	86.42	13.58
512	85.97	14.03
1024	85.53	14.47

Los resultados de la tabla 6.7 muestran que efectivamente el tiempo de comunicación es muy alto en comparación con el tiempo de procesamiento de FFT y PSD, ya que representa entre un 85 y un 87 por ciento del tiempo total. Sin embargo el tiempo de procesamiento del sistema heterogéneo resulta ser menor que para el sistema homogéneo.

Cabe notar que el tiempo de comunicación puede reducirse aproximadamente en un 10 por ciento utilizando una velocidad de 20 Mbits/seg en el link. La tabla 6.8 muestra el tiempo de procesamiento para la transmisión y recepción, y para el cálculo de FFT y PSD a 20 Mbits/seg.

Tabla 6.8 Suma de los tiempos de transmisión y recepción para cada ventana de datos en términos de porcentaje para 20 Mbits/seg.

Ventana de datos (Números reales)	Tiempo de procesamiento ($t_{TX} + t_{RX}$) %	Tiempo de procesamiento (FFT y PSD) %
64	77.44	22.56
128	76.75	23.25
256	76.09	23.91
512	75.39	24.61
1024	74.71	25.29

Referencias

Atkinson, P., Woodcock, J., 1982. Doppler ultrasound and its use in clinical measurement. Ed. Academic Press.

Computer System Architects, 1990. Logical System C for the Transputer: Version 89.1 User Manual.

Fish, P., "The doppler effect and blood flow: an instrument optimisation programme". *Developments in Acoustics and Ultrasonics*, Eds. M.J.W Povey and D.J. McClements, IOP Publishing, Bristol, 1992.

García Nocetti, D. F., Solano J., Martínez J. Heterogeneous Architecture for Parallel Real-Time Spectral Estimation in Doppler Blood Flow Instrumentation. Proceedings of the International Conference on Control '94. 21-24 March 1994, University of Warwick, Coventry, UK.

García Nocetti, D. F., Ruano, M. G., Fish, P. J., Fleming, P. J. Parallel Implementation of a Model-based Spectral Estimator for Doppler Blood Flow Instrumentation. Proceedings of the 8th International Parallel Symposium April 26-29 1994 Cancún, México. Págs. 810-814.

Inmos, 1991. The Transputer Development and Iq Systems Databook.

Papamichalis, P., 1990. Digital Signal Processing Applications with the TMS320 family, volumen 3. Ed. Prentice Hall and Digital Signal Processing Series Texas Instruments.

Ruano, M. G. "Investigation of real-time spectral analysis techniques for use with pulsed ultrasonic doppler blood-flow detectors", Thesis, 1992. University College of North Wales, Bangor, UK.

Sonicaid, Folleto: "Vasoflo 3 A unique blood-flow analysis system, Vascular Doppler Unit with Integral Spectrum Analyser & Optional Printer", 1989.

Texas Instruments, 1992. TMS320C3x User's Guide.

7 Conclusiones y comentarios para un trabajo futuro

Con base en los resultados obtenidos de la implementación y evaluación del nodo de procesamiento heterogéneo, se puede concluir lo siguiente:

- Se diseñó y se puso en funcionamiento un nodo de procesamiento heterogéneo basado en un DSP TMS320C30.
- El nodo de procesamiento heterogéneo puede fácilmente ser conectado dentro de un sistema multiprocesador formado por transputers, sin importar el tipo de transputer al cual es conectado directamente.
- El nodo de procesamiento heterogéneo permitió reducir los tiempos de procesamiento en el análisis espectral de una señal Doppler, utilizando métodos de estimación espectral convencionales (FFT), siendo posible realizar el procesamiento en tiempo real de este tipo de señales para ventanas de 512 y 1024 datos a una velocidad de transmisión de 10 Mbits/seg. Adicionalmente, se logró el procesamiento en tiempo real para ventanas de 256, 512 y 1024 datos a una velocidad de transmisión de 20 Mbits/seg.
- A pesar de que el tiempo de comunicación entre el T805 y el TMS30 ocupa un 80% del tiempo total procesamiento, este último resultó siete veces menor que el tiempo total de procesamiento, correspondiente a un nodo homogéneo y con el mismo tipo de tarea asignada.
- La utilización de un esquema basado en canales de comunicación para interconectar un transputer y un DSP resultó más viable que un esquema basado en memoria compartida. Asimismo, se conservaron las características de modularidad propias de los sistemas basados en transputers.

Conclusiones y comentarios para un trabajo futuro 7-2

- La implementación de la interfaz del nodo de procesamiento heterogéneo resultó sencilla pero poco flexible, debido a la lógica de control y registros de corrimiento utilizados.
- El uso de transputers y DSPs permite obtener un sistema de procesamiento en paralelo heterogéneo de alta eficiencia, debido a las características complementarias de ambos procesadores. Esto redundará en el desarrollo de aplicaciones que pueden utilizar los recursos disponibles de una manera más eficiente.
- Una plataforma de procesamiento paralelo heterogénea, permite una distribución de tareas entre los elementos de procesamiento más adecuada.
- El nodo de procesamiento heterogéneo puede resultar igualmente eficiente en otras aplicaciones como lo son el control adaptativo y los algoritmos de control y en general donde se requiera de funciones propias del procesamiento de señales.
- El uso del lenguaje C concurrente facilitó la programación de la aplicación, al hacerse uso de las librerías para graficación y manejo del *mouse* disponibles.
- La implementación de algoritmos para los tipos de procesadores utilizados, se reduce al desarrollo de programas en el ambiente correspondiente a los mismos, tomando en cuenta los requerimientos para la comunicación entre sí.

Con base en la experiencia obtenida en el diseño, implementación y puesta en marcha del nodo de procesamiento heterogéneo se proponen los siguientes puntos como base para la continuación de éste trabajo de tesis.

- Llevar a cabo la migración del nodo de procesamiento heterogéneo de una plataforma TEK805 CSA a una plataforma basada en TRAMS, en virtud de la flexibilidad, variedad y modularidad de este tipo de dispositivos. De igual forma, el soporte en cuanto a lenguajes de programación y a sistemas de desarrollo, así como la variedad de sistemas host (IBM PC, estación de trabajo SUN, etc.), hacen del uso de

Conclusiones y comentarios para un trabajo futuro 7-3

una plataforma compuesta por TRAMS un ambiente más apropiado de trabajo.

- Realizar un mapeo en memoria del link adaptor para incrementar la tasa de transferencia de datos. Esto significa el no poder utilizar el sistema de evaluación del TMS30 y tener que desarrollar un módulo especial para la aplicación.
- En cuanto al trabajo desarrollado en torno a la aplicación, es posible obtener una mejor resolución de la señal desplegada, utilizando métodos no convencionales (por ejemplo, covariancia modificada) y de igual forma beneficiarse, en cuanto a la reducción de tiempos de proceso, con el uso del nodo de procesamiento heterogéneo en el cálculo de la FFT y la PSD.
- Implementar una interfaz basada en un microcontrolador (como la presentada en el Capítulo 4), para el desarrollo de un canal estándar de comunicación RS-232, RS-422, etc., en virtud de que éstos manejan tasas bajas de transferencia de datos.

Apéndice A

El Transputer IMS T805

El transputer IMS T805 (T805) es un microprocesador CMOS de 32 bits de muy alta eficiencia, que permite realizar procesamiento en paralelo en un sólo chip. La figura A.1 muestra el diagrama de bloques del T805.

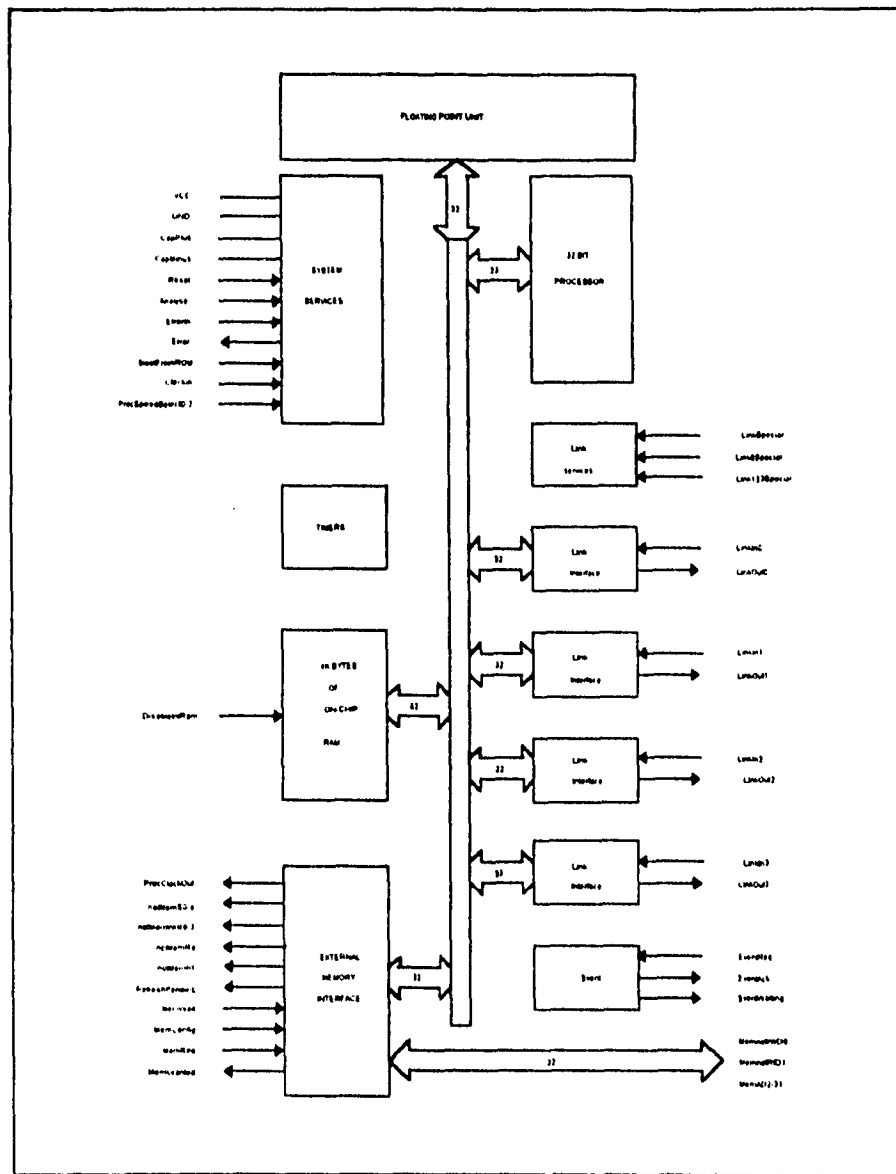


Figura A.1 Diagrama de bloques del IMS T805.

Características:

- Arquitectura de 32 bits
- Ciclo interno de 33 ns
- Velocidad de instrucción de 30 MIPS
- Velocidad de instrucción de 4.3 MFLOPS
- Compatible con el IMS T800, IMS T425 y el IMS T414
- Depurador
- Unidad de punto flotante de 64 bits
- Memoria RAM estática de 4 kbytes
- Velocidad de datos de 120 Mbytes/seg para memoria interna
- Memoria externa de 4 Gbytes
- Velocidad de datos para memoria externa de 40 Mbytes/seg
- Respuesta de interrupciones a 630 ns
- Cuatro canales de comunicación a 5/10/20 Mbits/seg
- Velocidad de datos bidireccional de 2.4 Mbytes/seg por canal de comunicación
- Ambiente gráfico con instrucciones de movimiento de bloques
- Buteo de ROM y de los canales de comunicación
- Reloj de entrada de 5 MHz
- Potencia de +5V +-5%

El set de instrucciones del T805 logra una eficiente implementación de lenguajes de alto nivel como C, Pascal y Fortran, además de proporcionar un soporte completo para OCCAM cuando se usa un transputer o una red de transputers.

Procesador

El procesador del T805 de 32 bits tiene instrucciones de procesamiento lógico, apuntadores de instrucción y de trabajo y un registro de operandos. Accesa directamente a una memoria interna de 4 Kbytes pudiendo almacenar datos o programa. Cuando son requeridas grandes cantidades de memoria, el procesador puede acceder un espacio de 4 Gbytes de memoria por medio de la interfaz de memoria externa (EMI).

Registros

Para explotar la rapidez de acceso a memoria interna, el T805 cuenta con 6 registros para la ejecución de procesos secuenciales:

Apuntador de espacio de trabajo. Apunta a un área de almacenamiento donde las variables locales son guardadas.

Apuntador de instrucción. Apunta a la siguiente instrucción a ser ejecutada.

Registro de operando. Se usa para la formación de los operandos de instrucción.

Registros A, B, y C. Forman una evaluación del stack.

Instrucciones

El conjunto de instrucciones del T805 ha sido diseñado para efectuar una eficiente y simple compilación de lenguajes de alto nivel. Todas las instrucciones tienen el mismo formato y están diseñadas para dar una representación compacta de aquellas operaciones que ocurren con mayor frecuencia en los programas.

Cada instrucción está formada por un byte dividido en dos partes de 4 bits. Los cuatro bits más significativos del byte corresponden al código de función y los cuatro bits menos significativos son el valor del dato. **Figura A.2.**

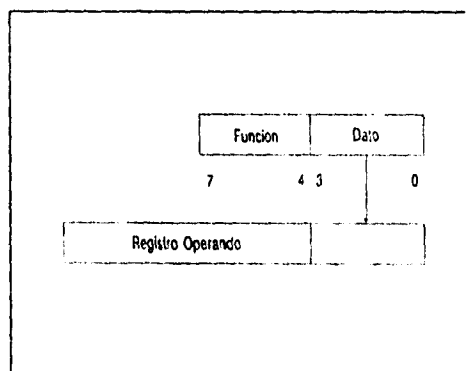


Figura A.2 Formato de instrucción.

Procesos y Concurrencia

Un proceso es una secuencia de instrucciones y puede tener prioridad alta o baja.

El T805 tiene un administrador de microcódigo que habilita a un número de procesos concurrentes para ser ejecutados juntos, compartiendo el tiempo del procesador.

En cualquier momento un proceso concurrente puede estar:

Activo	Si está siendo ejecutado En la lista de espera para ser ejecutado
Desactivo	Disponibile para entrar Disponibile para salir Esperando hasta un tiempo determinado

El administrador opera de tal manera que los procesos desactivos no consuman tiempo de procesador y asigna una porción del tiempo a cada proceso en turno. Los procesos que están esperando a ser ejecutados son colocados en dos listas de procesos, una de procesos de prioridad alta (0) y otra de procesos de prioridad baja (1). Cada lista tiene dos registros, uno de ellos, apunta al primer proceso de la lista y el otro al último.

Cada proceso es ejecutado hasta completar su acción, pero es sacado por el administrador mientras espera la comunicación con otro proceso, con un transputer, o espera un tiempo para terminar.

Para procesos que operan en paralelo, los procesos de baja prioridad se ejecutan durante un máximo de dos porciones de tiempo antes de ser forzados a salir del procesador. El período de cada porción de tiempo es de 5120 ciclos del reloj externo de 5 MHz.

Comunicaciones

La comunicación entre procesos se lleva a cabo por canales de comunicación en forma sincronizada. Un canal de comunicación entre dos procesos que se están ejecutando en el mismo transputer es implementado por una sola palabra en memoria, mientras que un canal entre procesos que se ejecutan en diferentes transputers es implementado por lazos de comunicación de punto a punto.

El T805 proporciona un número de operaciones para soportar el paso de mensajes, las más importantes son la entrada y salida de mensajes .

Las instrucciones de entrada y salida de mensajes usan las direcciones del canal para determinar si el canal es interno o es externo. De este modo, un proceso puede ser escrito y compilado sin conocer donde están conectados sus canales.

El proceso que esté listo, debe esperar a que un segundo proceso también lo esté. De esta forma, un proceso ejecuta una entrada o una salida cargando el stack con un apuntador a un mensaje, la dirección de un canal y un contador con el número de bytes a ser transferidos, para después ejecutar la instrucción de entrada o salida de mensajes. Los datos son transferidos si el otro proceso está disponible. Si el canal no está listo, el proceso podría ser abortado por el administrador.

Timers

El transputer T805 tiene dos timers de 32 bits. Los timers proporcionan un tiempo de proceso exacto, permitiendo a los procesos salir por ellos mismos después de un tiempo específico.

Uno de los timers está accesible sólo para los procesos de prioridad alta y es incrementado cada microsegundo. Un período es completado aproximadamente en 4295 segundos. El otro timer está accesible sólo para los procesos de prioridad baja y es incrementado cada 64 microsegundos dando exactamente 15625 marcas en un segundo, con un período completo de aproximadamente 76 horas.

Unidad de Punto Flotante

La unidad de punto flotante (FPU) de 64 bits proporciona una aritmética de longitud simple y doble de acuerdo al estándar de punto flotante ANSI-IEEE 754-1985. La FPU ejecuta aritmética de punto flotante en forma concurrente con la CPU, logrando hasta 3.3 Mflops 30 MHz. La comunicación de datos entre memoria y la FPU está bajo el control de la CPU.

El FPU tiene un stack para la manipulación de números en punto flotante. Los registros del stack son FA, FB y FC cada uno manteniendo datos de 32 ó 64 bits. Además tiene una bandera que se enciende cuando se carga un valor en punto flotante. La FPU puede ser usada en procesos de prioridad alta y baja.

Los puntos en una instrucción donde los datos necesitan ser transferidos a o desde la FPU son llamados puntos de sincronización.

Memoria

El T805 tiene 4 Kbytes de memoria interna estática para altas velocidades en el procesamiento de datos. Cada acceso a memoria interna lleva un ciclo de procesador. También se pueden acceder 4 Gbytes de espacio de memoria externa. La memoria interna y la memoria externa forman parte del mismo espacio de direcciones. Si la RAM interna es deshabilitada, entonces, todas las direcciones internas son mapeadas a la RAM externa. La tabla A.1 muestra el mapa de memoria del transputer T805.

La memoria del T805 es direccionada por byte, con palabras de 4 bytes. El byte menos significativo de una palabra es el byte con la menor dirección.

La memoria interna inicia en la dirección más negativa #80000000 terminando en #80000FFF. La memoria del usuario, por su parte, inicia en la dirección #80000070, dándole el nombre de *MemStart*. El área de memoria interna reservada abajo de *MemStart* es usada para implementar links y eventos en los canales.

Por otro lado, el espacio de memoria externa inicia en la dirección #80001000 y se prolonga de la dirección #00000000 a la dirección #7FFFFFFF. La configuración de la memoria de datos y el código de *bootstrapping* de memoria ROM deben estar en la dirección más positiva, iniciando en #7FFFFF6C y en la #7FFFFFFE respectivamente. El espacio direccionado inmediatamente abajo es usado convencionalmente por el código base de ROM.

Tabla A.1 Mapa de memoria del IMS T805.

hi	Machine map	lo	Byte address
	Reset Inst		#7FFFFFFE
	Memory configuration		# 7FFFFFF8 #7FFFFF6C
			#0 #80001000 - Start of external memory
			#80000070 MemStart
	Reserved for		#8000006C
	Extended functions		#80000048
	EregIntSaveLoc		#80000044
	STATUSIntSaveLoc		#80000040
	CregIntSaveLoc		#8000003C
	BregIntSaveLoc		#80000038
	AregIntSaveLoc		#80000034
	IptrIntSaveLoc		#80000030
	WdescIntSaveLoc		#8000002C
	TPtrLoc1		#80000028
	TPtrLoc0		#80000024
	Event		#80000020
	Link 3 Input		#8000001C
	Link 2 Input		#80000018
	Link 1 Input		#80000014
	Link 0 Input		#80000010
	Link 3 Output		#8000000C
	Link 2 Output		#80000008
	Link 1 Output		#80000004
	Link 0 Output		#80000000 Base of memory

Canales de comunicación

El transputer T805 tiene cuatro canales bidireccionales Inmos que proporcionan una comunicación sincronizada entre procesadores y el mundo exterior. Un canal comprende un canal de entrada y uno de

salida. Cada byte de datos enviado en un canal es reconocido a la entrada del mismo, de esta forma, cada línea lleva tanto datos como información de control.

El estado pasivo de un canal de salida es bajo. Cada byte de datos es transmitido con un bit de inicio en nivel alto, seguido por otro bit en nivel alto, por 8 bits de datos y por un bit de paro a nivel bajo. El bit menos significativo del dato es transmitido primero. Después de transmitir un byte de datos el transmisor espera el reconocimiento, el cual consta de un bit de inicio a nivel alto y un bit a nivel bajo. El reconocimiento significa que un proceso fue capaz de recibir el byte de datos y por tanto, el elemento receptor es capaz de recibir otro byte.

Los canales de comunicación del T805 soportan una velocidad estándar de 10 Mbits/seg. Pueden ser utilizados también a 5 o 20 Mbits/seg con dispositivos a 17, 20 y 25 MHz, y a 20 Mbits/seg para dispositivos más rápidos. La conexión directa puede ser hecha entre dispositivos separados por 300 milímetros. Para distancias mayores se une una resistencia RM de 100 ohms, lo cual ocasiona que el retraso de la línea sea menor a 0.4 bit y asegura que la reflexión regrese antes de que el siguiente dato sea enviado. La velocidad de datos es indicada por el uso de memoria interna y puede ser afectada por el número de accesos y duración del ciclo de memoria externa.

Referencias

Inmos, 1989. The Transputer databook.

Apéndice B

El DSP TMS320C30

El TMS320C30 (TMS30) es un eficiente procesador digital de señales de punto flotante. Su arquitectura pipeline permite que la cantidad de datos procesados sea muy alta. Cuenta con 16 millones de palabras de 32 bits, lo que permite desarrollar aplicaciones que requieren un espacio grande de memoria. Contiene un bus interno por el que se contentan los dispositivos periféricos: dos puertos seriales, dos timers y controlador DMA. Por lo anterior, el TMS30 constituye un sistema estable a buen precio en un sólo chip. La figura B.1 muestra un diagrama de bloques de la arquitectura del TMS30.

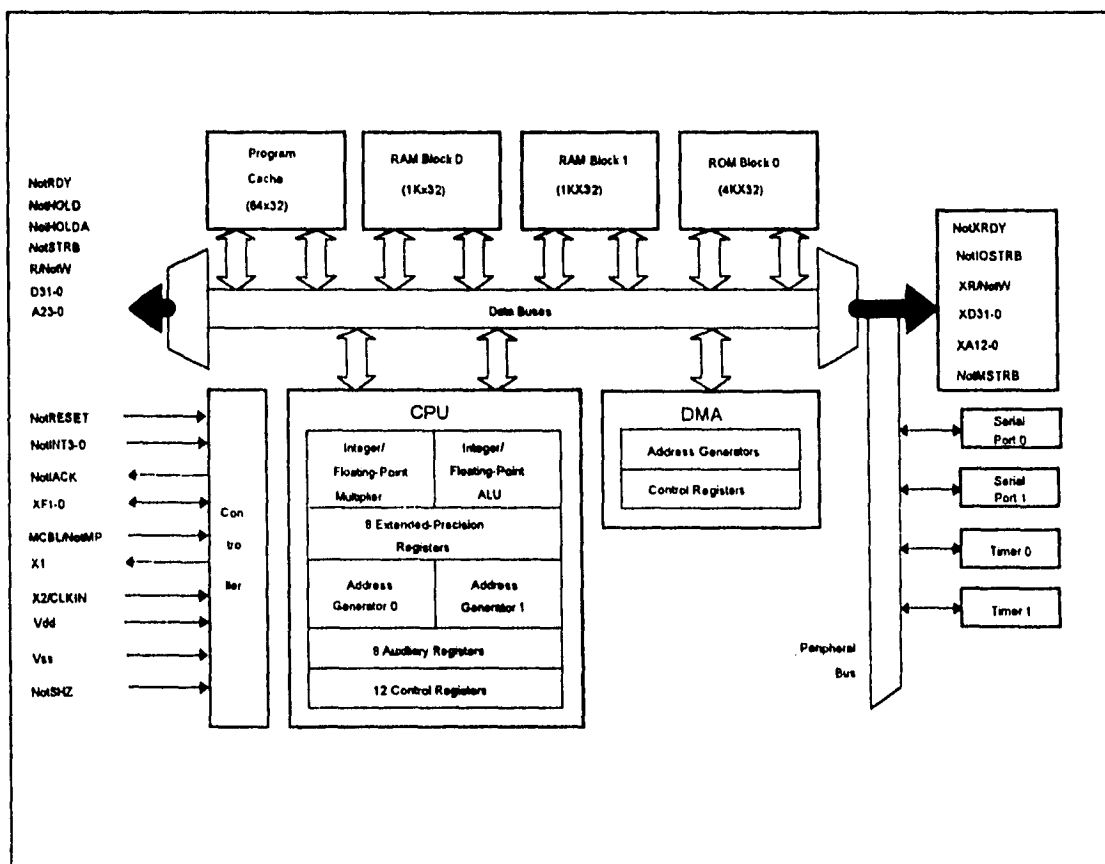


Figura B.1 Diagrama de bloques del TMS320C30.

Arquitectura

Unidad de Procesamiento Central (CPU)

La arquitectura de la CPU está basada en registros y la integran los siguientes elementos:

- Multiplicador de punto flotante y entero
- Unidad aritmética y lógica (ALU) para ejecutar aritmética de punto flotante, entera y operaciones lógicas
- Corrimiento de 32 bits
- Buses internos (CPU1/CPU2 y REG1/REG2)
- Unidades aritméticas de registros auxiliares (ARAUs)
- Registro de archivo CPU

Multiplicador

El multiplicador ejecuta multiplicaciones en un sólo ciclo con enteros de 24 bits y con valores de punto flotante de 32 bits. La aritmética de punto flotante permite realizar operaciones a velocidades de punto fijo de 50 ns así como alcanzar un alto grado de paralelismo.

Cuando el multiplicador ejecuta multiplicaciones en punto flotante, las entradas son números de 32 bits y el resultado es un número de 40 bits. Cuando el multiplicador ejecuta una multiplicación de enteros, la entrada del dato es de 24 bits y el dato de salida es de 32 bits.

Unidad aritmética-lógica (ALU)

El ALU ejecuta operaciones con enteros de 32 bits, operandos lógicos de 32 bits y datos en punto flotante de 40 bits, y de igual modo conversiones de enteros y datos a punto flotante en un sólo ciclo. Los resultados del ALU son mantenidos en formatos de 32 bits ó de 40 bits. El *registro barrel* se utiliza para hacer corrimientos hasta de 32 bits a la izquierda o a la derecha en un sólo ciclo.

Unidad aritmética de registros auxiliares (ARAUs)

Las dos unidades aritméticas de registros auxiliares (ARAU0 y ARAU1) pueden generar dos direcciones en un ciclo. Los ARAUs operan en paralelo con el multiplicador y con el ALU. Estas unidades toleran

direccionamientos con desplazamientos, con registros índice (IRO y IR1), circulares y bit-reversed.

Registro de archivo CPU

El TMS30 tiene 28 registros en un registro de archivo multipuerto que está estrechamente unido al CPU. Estos registros pueden ser operados por el multiplicador y por el ALU, y pueden ser usados como registros de propósito general. Realizan funciones especiales: 8 registros para precisión extendida, 8 registros que soportan varios modos de direccionamiento y los registros restantes proporcionan funciones del sistema tales como direccionamiento, administración del stack, status del procesador, interrupciones y repetición de bloques.

Registros de precisión extendida (R7-R0). Almacenan y soportan números enteros de 32 bits y de punto flotante de 40 bits.

Registros auxiliares de 32 bits (AR7-AR0). Estos registros son accedidos por el CPU y modificados por las dos unidades aritméticas de registro auxiliar (ARAUs). La principal función de los registros auxiliares es la generación de direcciones de 24 bits. También son utilizados como contadores de ciclos y como registros de propósito general 32 bits que pueden ser modificados por el multiplicador y por el ALU.

Apuntador de página de datos (DP). Es un registro de 32 bits, el octavo bit menos significativo del apuntador de página de datos se utiliza en el modo de direccionamiento directo como apuntador a la página de datos direccionada. Las páginas de datos son de 64 K palabras con un total de 256 páginas.

Registros índice (IRO, IR1). Son registros de 32 bits y tienen el valor usado por la unidad aritmética de registro auxiliar (ARAU) para calcular una dirección indexada.

Registro de tamaño de bloque (BK). Registro de 32 bits, es utilizado por el ARAU en un direccionamiento circular para especificar el tamaño del bloque de datos.

Apuntador del Stack (SP). Es un registro de 32 bits que tiene la dirección de la parte superior del stack. El SP permite apuntar al último elemento guardado dentro del stack. Un *push* realiza un preincremento y un *pop* un

postdecremento del apuntador del mismo stack. El SP es manipulado por interrupciones, *traps*, llamadas a subrutinas, retornos y por las instrucciones PUSH y POP.

Registro de Status (ST). En él se encuentra la información global del estado del CPU. Las operaciones modifican la condición de las banderas del registro de status de acuerdo al resultado si fue cero, negativo, etc.

Registro habilitador de interrupciones CPU/DMA (IE). Es un registro de 32 bits. Los bits de habilitación de interrupciones del CPU están en las localidades 10-0 y los bits de habilitación de interrupciones del DMA están en la localidades 26-16. Un '1' en un bit del registro habilitador de interrupciones habilita la interrupción correspondiente y un '0' la deshabilita.

Registro de bandera de interrupciones del CPU (IF). Este registro es de 32 bits. Un '1' en un bit del registro de bandera de interrupciones indica que la interrupción está activa y un '0' indica que la interrupción está desactiva.

Registro de banderas de I/O (IOF). Controla la función de los pines externos XF0 y XF1. Estos pines son configurados para entrada o salida y pueden ser de lectura o de escritura.

Contador de repetición (RC). Este registro de 32 bits se utiliza para especificar el número de veces que un bloque de código es repetido. Cuando el procesador está operando en modo de repetición, el registro de dirección de inicio de repetición (RS) de 32 bits, tiene la dirección de inicio del bloque de memoria de programa que se va a repetir y el registro de dirección de fin de repetición (RE) de 32 bits tiene la última dirección del bloque.

Contador de programa (PC). Registro de 32 bits, tiene la dirección de la siguiente instrucción a ser cargada. Este registro no forma parte del registro de archivo del CPU, pero puede ser modificado por instrucciones para modificar el flujo del programa.

Organización de memoria

El espacio de memoria total del TMS30 es de 16M de palabras de 32 bits. La memoria RAM se divide en dos bloques (0 y 1), cada uno de 1K x 32 bits. El bloque de memoria ROM es de 4K x 32 bits. Cada bloque de memoria

RAM o ROM soportan dos accesos del CPU en un ciclo. Los buses de programa separado, de datos y del DMA permiten cargas de programas en paralelo, leer y escribir datos y realizar operaciones con el DMA. Por ejemplo el CPU puede acceder dos datos de un bloque de memoria RAM y ejecutar la carga de un programa externo en paralelo con el DMA el cual puede estar cargando otro bloque de memoria RAM y todo esto en un ciclo.

El TMS30 cuenta además con una memoria caché de 64 x 32 bits para almacenar secciones de código repetidas frecuentemente, reduciendo de esta forma el número de accesos fuera del procesador.

Mapa de memoria

El mapa de memoria depende del modo en el que se está ejecutando el procesador, en modo microprocesador (MC/notMP o MCBL/notMP=0) (Tabla B.1) o en modo de microcomputadora (MC/notMP o MCBL/notMP=1) (Tabla B.2). A partir de la localidad de memoria 800000h hasta la 801FFFh son utilizadas para el bus expansión. Cuando esta región de memoria es accesada, la señal notMSTRB está activa. De la localidad 802000h a la 803FFFh son reservadas. Las localidades 804000h a 805FFFh son direccionadas para el bus de expansión y cuando esta sección es accesada, la señal notIOSTRB permanece activa. De la localidad 806000h hasta la 807FFFh son reservadas. El mapa de memoria para los registros de los dispositivos periféricos abarca de la localidad 808000h a la 8097FFh. En ambos modos, el bloque 0 de RAM está localizado a partir de la localidad 809800h a la 809BFFh y para el bloque 1 está localizado de la dirección 809C00h a la 809FFFh. Las localidades 80A000h a 0FFFFFFh son accesadas por el puerto de memoria externa (señal notSTRB activa).

En modo microprocesador los 4 K de memoria interna ROM no son direccionados en el mapa de memoria del TMS30. Las localidades 0h a 0BFh (192 en total) forman parte del vector de interrupciones, el vector trap y localidades reservadas. De la localidad 0C0h a la 7FFFFFFh es direccionada por el puerto de memoria externa (señal notSTRB activa).

En modo microcomputadora, los 4 K de memoria interna ROM son mapeados a partir de la localidad 0h a la 0FFFh. Las 192 localidades (0h a 0BFh) son asignadas al vector de interrupciones, al vector de trap y un espacio reservado. Las localidades 1000h a 7FFFFFFh son accesadas por el puerto de memoria externo (señal notSTRB activa).

Tabla B.1 Mapa de memoria del TMS30.
Modo microprocesador.

0h	Interrupt Locations and Reserved (192) (External STRB Active)
03Fh 040h	
	External STRB Active
7FFFFFh 800000h	Expansion Bus MSTRB Active (8K Words)
801FFFh 802000h	Reserved (8K Words)
803FFFh 804000h	Expansion Bus IOSTRB Active (8K Words)
805FFFh 806000h	Reserved (8K Words)
807FFFh 808000h	Peripheral Bus Memory-Mapped Registers (6K Words Internal)
8097FFh 809800h	RAM Block 0 (1K Word Internal)
809BFFh 809C00h	RAM Block 1 (1K Word Internal)
809FFFh 80A000h	External STRB Active
FFFFFFh	

Tabla B.2 Mapa de memoria del TMS30.
Modo microordenador.

0h	Interrup Locations and Reserved (192)
0BFh 0C0h	ROM (Internal)
0FFFh 1000h	External STRB Active
7FFFFFFh 800000h	Expansion Bus MSTRB Active (8K Words)
801FFFh 802000h	Reserved (8K Words)
803FFFh 804000h	Expansion Bus IOSTRB Active (8K Words)
805FFFh 806000h	Reserved (8 Words)
807FFFh 808000h	Peripheral Bus Memory-Mapped Registers (Internal) (6K Words Internal)
8097FFh 809800h	RAM Block 0 (1K Word Internal)
809BFFh 809C00h	RAM Block 1 (1K Word Internal)
809FFFh 80A000h	External STRB Active
FFFFFFh	

Modos de direccionamiento

Existen 5 modos de direccionamiento en el TMS30. La tabla B.3 muestra los modos de direccionamiento con ejemplos.

1) Modos de direccionamiento general.

Registro. El operando es un registro del CPU.

Inmediato Corto. El operando es un valor inmediato de 16 bits.

Directo. El operando es el contenido de una dirección de 24 bits.

Indirecto. Un registro auxiliar indica la dirección del operando.

2) Modos de direccionamiento de tres operandos.

Registro. El operando es un registro del CPU.

Indirecto. Un registro auxiliar indica la dirección del operando.

3) Modos de direccionamiento en paralelo.

Registro. El operando es un registro de precisión extendida.

Indirecto. Un registro auxiliar indica la dirección del operando.

4) Modo de direccionamiento de longitud inmediata.

Longitud inmediata. El operando es un valor inmediato de 24 bits.

5) Modos de direccionamiento de salto condicional.

Registro. El operando es un registro del CPU.

PC-relativo. Un desplazamiento de 16 bits signado es adicionado al PC.

Tabla B.3. Modos de direccionamiento del TMS30.

Modo	Ejemplo	Operación	Descripción
Registro	ADDF R0,R1		Operando en R0
Directo	ADDF @MEM,R1	Addr=MEM	Operando en MEM
Corto Inmediato	ADDF 3.14,R1		Operando =3.14
Largo Inmediato	BR LABEL		Salto a etiqueta LABEL
PC relativo	BGE LABEL		Salto a etiqueta LABEL
Indirecto	ADDF *+AR0(di),R1	Addr=AR0+di	Adiciona predesplazamiento sin modificación
Indirecto	ADDF *-AR0(di),R1	Addr=AR0-di	Resta predesplazamiento sin modificación
Indirecto	ADDF *++AR0(di),R1	Addr=AR0+di AR0=AR0+di	Adiciona predesplazamiento y modifica
Indirecto	ADDF *--AR0(di),R1	Addr=AR0-di AR0=AR0-di	Resta predesplazamiento y modifica
Indirecto	ADDF *AR0++(di),R1	Addr=AR0 AR0=AR0+di	Adiciona postdesplazamiento y modifica
Indirecto	ADDF *--AR0(di),R1	Addr=AR0 AR0=AR0-di	Resta postdesplazamiento y modifica
Indirecto	ADDF *AR0++(di)%,R1	Addr=AR0 AR0=circ(AR0+di)	Adiciona postdesplazamiento y modificación circular
Indirecto	ADDF *AR0--(di)%,R1	Addr=AR0 AR0=circ(AR0-di)	Resta postdesplazamiento y modificación circular
Indirecto	ADDF *AR0++(IR0)B,R1	Addr=AR0 AR0=B(AR0+IR0)	Adiciona después de indexar y modifica con bit-reversed

Nota: di es un entero entre 0 y 255 o uno de los registro índice IR0 e IR1.

Operaciones del bus interno

El TMS30 proporciona una alta eficiencia debido a su sistema de bus interno y a su paralelismo. Los buses de programa (PADDR y PDATA), de datos (DADDR1, DADDR2, y DDATA) y de DMA (DMAADDR y DMADATA) separados, permiten la carga de programas paralelos, acceso de datos y accesos de DMA.

El contador de programa (PC) es conectado al bus de direcciones de programa de 24 bits (PADDR) y el registro de instrucción (IR) es conectado al bus de datos de programa de 32 bits (PDATA). Estos buses pueden cargar una palabra de instrucción cada ciclo de máquina.

Los buses de dirección de datos de 24 bits (DADDR1 y DADDR2) y el bus de datos de 32 bits (DDATA) pueden acceder dos veces la memoria de datos cada ciclo de máquina. El bus DDATA lleva el dato al CPU a través de los buses CPU1 y CPU2 y a su vez estos buses pueden llevar dos operandos de memoria de datos al multiplicador, al ALU y al registro de archivo cada ciclo de máquina. De igual forma, los buses de registro REG1 y REG2,

pueden llevar dos datos del registro de archivo al multiplicador y al ALU cada ciclo.

Por otro lado, el controlador DMA es soportado por el bus de direcciones de 24 bits (DMAADDR) y por el bus de datos de 32 bits (DMADATA). Estos buses permiten al DMA ejecutar accesos a memoria en paralelo con los accesos de memoria ocurridos por los buses de datos y de programa.

Operación del bus externo

El TMS30 tiene dos interfaces externas: el bus primario y el bus de expansión. Ambos buses consisten de un bus de datos de 32 bits y un conjunto de señales de control. El bus primario tiene un bus de direcciones de 24 bits, mientras que el bus de expansión tiene un bus de direcciones de 13 bits. Se utilizan estos buses para direccionar memoria de datos, de programa o espacio de E/S. Además, cuentan con una señal externa RDY para la generación de estados de espera, que pueden ser dados por medio del software.

Interrupciones externas

El procesador de señales TMS30 soporta 4 interrupciones externas (notINT3-notINT0), varias interrupciones internas y una señal externa no mascarable notRESET. Se utilizan para interrumpir al DMA o al CPU. Cuando el CPU responde a la interrupción, el pin IACK indica un reconocimiento de la interrupción externa.

Instrucción de Interbloqueo

Las banderas externas de E/S, XFO y XF1 pueden configurarse como pines de entrada o salida por medio de software. Estos dos pines son utilizados por las operaciones de interbloqueo del TMS30. El grupo de instrucciones que realizan operaciones de interbloqueo soportan comunicación entre procesadores.

Periféricos

El TMS30 cuenta con 3 módulos periféricos: dos timers, dos puertos seriales y un controlador de acceso directo a memoria (DMA), los cuales son controlados por un bus dedicado. Este bus consta de un bus de datos de 32 bits y un bus de direcciones de 24 bits.

Acceso directo a memoria (DMA)

El DMA puede leer o escribir a cualquier localidad en el mapa de memoria sin interferir con la operación del CPU. Cuenta con generadores de direcciones, registros fuente y destino, y un contador de transferencia. Los buses de direcciones y de datos del DMA minimizan los conflictos entre el CPU y el controlador DMA. La operación del DMA consiste en transferir un bloque o una palabra hacia o desde memoria.

Timers

Los dos timer son contadores de tiempo y de eventos de 32 bits de propósito general con dos modos de señalamiento y un reloj interno o externo. Cada timer tiene un pin de E/S que puede ser usado como reloj de entrada al timer o como una señal de salida manejada por el timer. El pin también puede ser configurado como un pin de E/S de propósito general.

Puertos Serie

Los dos puertos seriales bidireccionales del TMS30 son totalmente independientes. Estos puertos tienen un conjunto de registros de control para cada uno. Ambos puertos seriales pueden ser configurado para transferir 8, 16, 24 o 32 bits de datos por palabra. El reloj para cada puerto serial puede ser originado interna o externamente. Internamente es generado un reloj con división hacia abajo (*divide-down*). Los pines del puerto serial son configurados como pines de E/S de propósito general o también como timers. La figura B.2 muestra el diagrama de bloques del puerto serie y la tabla B.4 el mapa de memoria.

Cada puerto serial tiene 8 registros de mapeo de memoria:

- Registro de control global
- Dos registros de control para 6 pines de E/S serial
- Tres registros timer de recepción/transmisión
- Registro de transmisión de datos
- Registro de recepción de datos

El *registro de control global* controla las funciones globales del puerto serial y determina su modo de operación. El buffer de transmisión tiene la siguiente palabra a ser transmitida y el de recepción tiene la última palabra recibida.

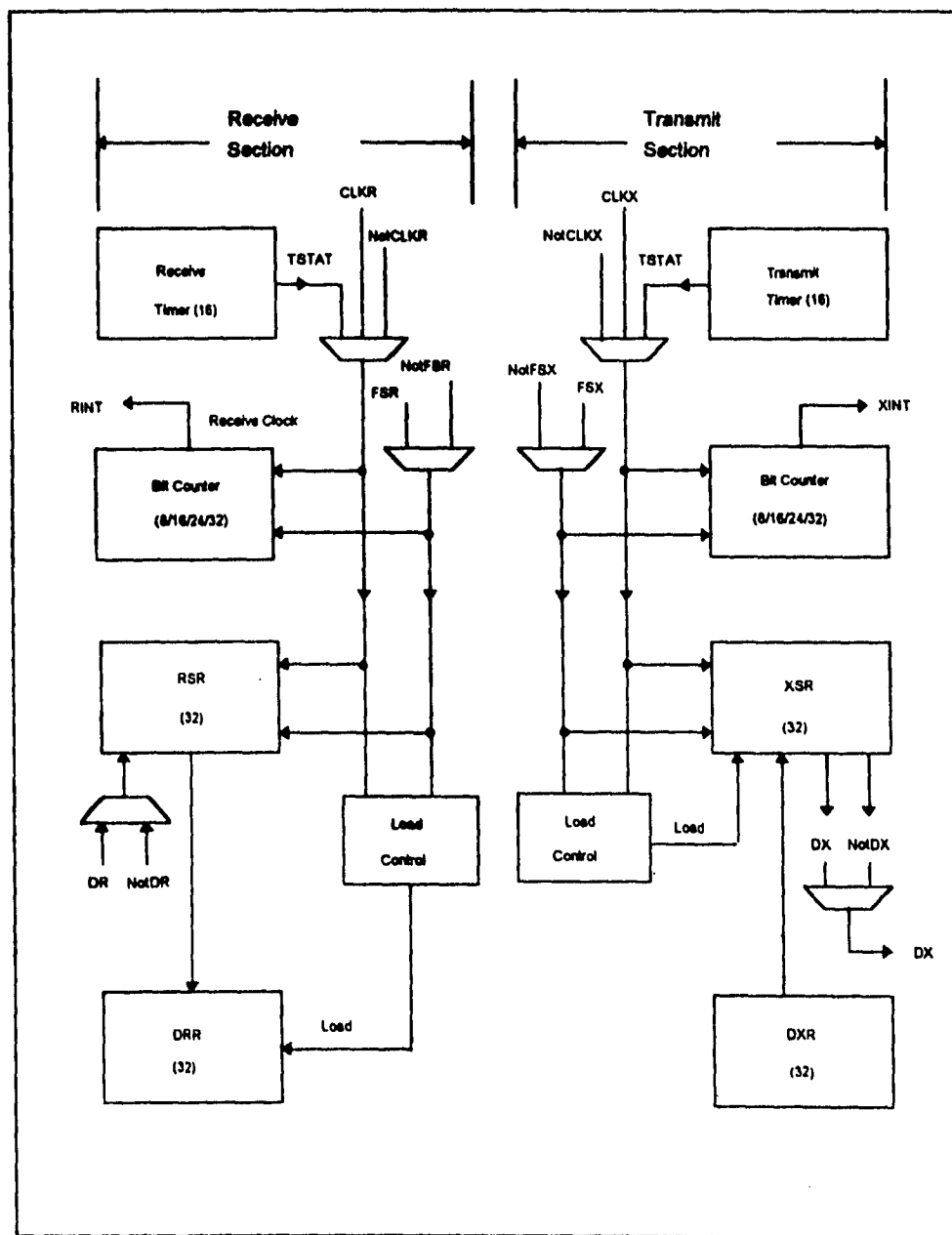


Figura B.2 Diagrama de bloques de puerto serial.

Tabla B.4 Mapa de memoria del puerto serial.

Register	Peripheral Address	
	Serial Port 0	Serial Port 1
Serial-Port Global Control	808040h	808050h
Reserved	808041h	808051h
FSX/DX/CLKX Port Control	808042h	808052h
FSR/DR/CLKR Port Control	808043h	808053h
E/X Timer Control	808044h	808054h
R/X Timer Counter	808045h	808055h
R/X Timer Period	808046h	808056h
Reserved	808047	808057h
Data Transmit	808048h	808058h
Reserved	808049h	808059h
Reserved	80804Ah	80805Ah
Reserved	80804Bh	80805Bh
Data Receive	80804Ch	80805Ch
Reserved	80804Dh	80805Dh
Reserved	80804Eh	80805Eh
Reserved	80804Fh	80805Fh

Registro de transmisión de datos. Cuando el registro de transmisión de datos es cargado (DXR), el transmisor carga la palabra en el registro de desplazamiento de transmisión (XSR) y los bits son desplazados. Cuando el DXR es cargado en el XSR, el bit XRDY es activado, indicando de esta forma que el buffer está disponible para recibir la siguiente palabra. Se requieren de 4 pasos para transmitir la palabra dentro del registro de desplazamiento ya que son 4 bytes de la palabra de datos. El desplazamiento es a la izquierda.

Registro de recepción de datos. Cuando los datos seriales son entrada, el receptor desplaza los bits al registro de desplazamiento de recepción (RSR). Cuando el número total de bits es desplazado, el registro receptor de datos (DRR) es cargado del RSR y el bit RRDY es activado. Si el DRR no ha sido leído y el RSR está lleno, el receptor es mantenido con la palabra. El DRR debe ser leído para permitir el nuevo dato en el RSR.

Modos de operación del puerto serie

Modos Continuos de Transmisión y Recepción. Cuando se elige el modo continuo, escrituras consecutivas no generan o esperan nuevas señales de pulsos de sincronía. Sólo la primer palabra de un bloque comienza con una sincronización activa. Después de esto, el dato es transmitido tan pronto como el dato es cargado en el DXR antes que la última palabra haya sido transmitida.

Tan pronto como el TXRDY es activado y todo el dato ha sido transmitido fuera del registro de desplazamiento, el pin DX es puesto en estado de alta impedancia y una escritura subsecuente al DXR inicia un nuevo bloque y un nuevo FSX.

Similarmente con FSR, el receptor continua desplazando el nuevo dato y cargando el DRR. Si el buffer de recepción de datos no está listo antes de que la siguiente palabra sea desplazada el dato podría perderse. El bit RFSM puede ser usado para terminar el modo continuo de recepción.

Modo Handshake. El modo handshake (HS=1) permite una conexión directa entre procesadores. En este modo todas las palabras de datos son transmitidas con un bit de inicio alto al inicio y no se transmite otra palabra hasta que no se mande un bit cero que funciona como bit de reconocimiento. Este esquema permite conectar procesadores de una manera sencilla sin conectar hardware adicional y garantiza una comunicación segura.

Fuentes de Interrupción del Puerto Serial

Un puerto serial tiene 4 fuentes de interrupción:

1. Interrupción del timer de transmisión
2. Interrupción del timer de recepción
3. Interrupción del transmisor
4. Interrupción del receptor

Todas las configuraciones de operación del puerto serial pueden ser clasificadas en 2 categorías: transferencia de datos a velocidad fija y transferencia de datos a velocidad variable.

En una transferencia de datos a velocidad fija existe un modo ráfaga y un modo continuo. En un modo ráfaga de operación, la transferencia de palabras es separada por períodos de inactividad del puerto serie. Los pulsos FSX/FSR inicializan la transferencia y cada transferencia involucra una palabra. En un modo continuo, no existen estos períodos de inactividad en la transferencia sucesiva de palabras, el primer bit de una nueva palabra es transferido en el siguiente pulso de reloj CLKX/R seguido por el último bit de la palabra previa. Esto ocurre continuamente hasta que el proceso termina.

En el modo de velocidad de datos fijo, la transferencia continua puede ser ejecutada siempre que $R/XFSM=0$ dure el mismo tiempo que el pulso de sincronía o si DXR es nuevamente cargado con un pulso FSX generado internamente.

Para recibir operaciones con un pulso de sincronía FSX generado externamente, una vez que la transferencia ha comenzado, los pulsos de sincronía son requeridos sólo durante el último bit de transferencia para iniciar otra transferencia continua, de otra manera los pulsos de sincronía son ignorados. La transferencia continua puede ocurrir si el pulso es mantenido alto. Con un FSX generado internamente hay un retraso de 2.5 ciclos de reloj (CLKX) contados desde que DXR es cargado hasta que se activa el pulso FSX. Este retraso ocurre cada vez que DXR es cargado. Durante una transmisión continua, la instrucción que carga DXR debe ser ejecutada en el bit N-3 para una transmisión de N bits.

Una vez que el proceso ha comenzado, una interrupción XINT y una RINT son generadas al comienzo de cada transferencia. La interrupción XINT indica que el registro XSR ha sido cargado de DXR y que DXR puede ser cargado nuevamente. Para mantener una transferencia continua en modo fijo con un pulso de sincronía interno FSX, DXR debe ser cargado nuevamente durante la transferencia.

La interrupción RINT indica que una palabra completa a sido recibida y transferida a DRR, RINT se utiliza para indicar el tiempo apropiado para leer DRR.

La transferencia continua es terminada al dejar de enviar los pulsos de sincronía o cuando FSX es generado internamente y no carga nuevamente DXR.

La transferencia continua puede ser realizada sin los pulsos de sincronía siempre que R/XFSM estén en nivel alto. En este modo, la operación del puerto serial es similar a la operación con el pulso de sincronía, excepto que el pulso es llamado únicamente en la primer palabra transferida y ningún pulso de sincronía es transferido posteriormente.

La transferencia de datos a velocidad variable también soporta el modo ráfaga y el modo continuo. La operación del modo ráfaga con velocidad de datos variable es similar al modo ráfaga con velocidad de datos fija. Con velocidad de datos variable, la diferencia de tiempo entre FSX/R y el dato es insignificante al comienzo y al final de la transferencia.

Cuando se transmite continuamente con una velocidad de datos variable con pulso de sincronía, el tiempo es el mismo que para el modo a velocidad fija. En este caso DXR debe cargarse nuevamente no más de N-4 bits para mantener la operación continua.

La operación en modo variable sin pulso de sincronía es similar a la operación continua con pulso de sincronía a velocidad de datos fija. DXR debe cargarse antes de N-4 bits, adicionalmente cuando R/XFSM se activa o se desactiva en un modo variable, este cambio no debe ocurrir en más de N-1 bits para que el resultado se dé en la transferencia actual.

La transferencia continua puede ser realizada sin los pulsos de sincronía siempre que R/XFSM estén en nivel alto. En este modo, la operación del puerto serial es similar a la operación con el pulso de sincronía, excepto que el pulso es llamado únicamente en la primer palabra transferida y ningún pulso de sincronía es transferido posteriormente.

La transferencia de datos a velocidad variable también soporta el modo ráfaga y el modo continuo. La operación del modo ráfaga con velocidad de datos variable es similar al modo ráfaga con velocidad de datos fija. Con velocidad de datos variable, la diferencia de tiempo entre FSX/R y el dato es insignificante al comienzo y al final de la transferencia.

Cuando se transmite continuamente con una velocidad de datos variable con pulso de sincronía, el tiempo es el mismo que para el modo a velocidad fija. En este caso DXR debe cargarse nuevamente no más de N-4 bits para mantener la operación continua.

La operación en modo variable sin pulso de sincronía es similar a la operación continua con pulso de sincronía a velocidad de datos fija. DXR debe cargarse antes de N-4 bits, adicionalmente cuando R/XFSM se activa o se desactiva en un modo variable, este cambio no debe ocurrir en más de N-1 bits para que el resultado se dé en la transferencia actual.

Desarrollo de programas en lenguaje ensamblador para el TMS320C30

Los procesadores de punto flotante de la familia TMS320 cuentan con un completo soporte de herramientas de desarrollo en software y hardware. Para compilar, ligar y generar archivos ejecutables a partir del lenguaje ensamblador, el TMS320C30 cuenta con herramientas que crean y usan archivos objeto con un formato común COFF (*Common Object File Format*). Estos archivos están formados por bloques separados de código y de datos, que pueden ser cargados en diferentes espacios de memoria.

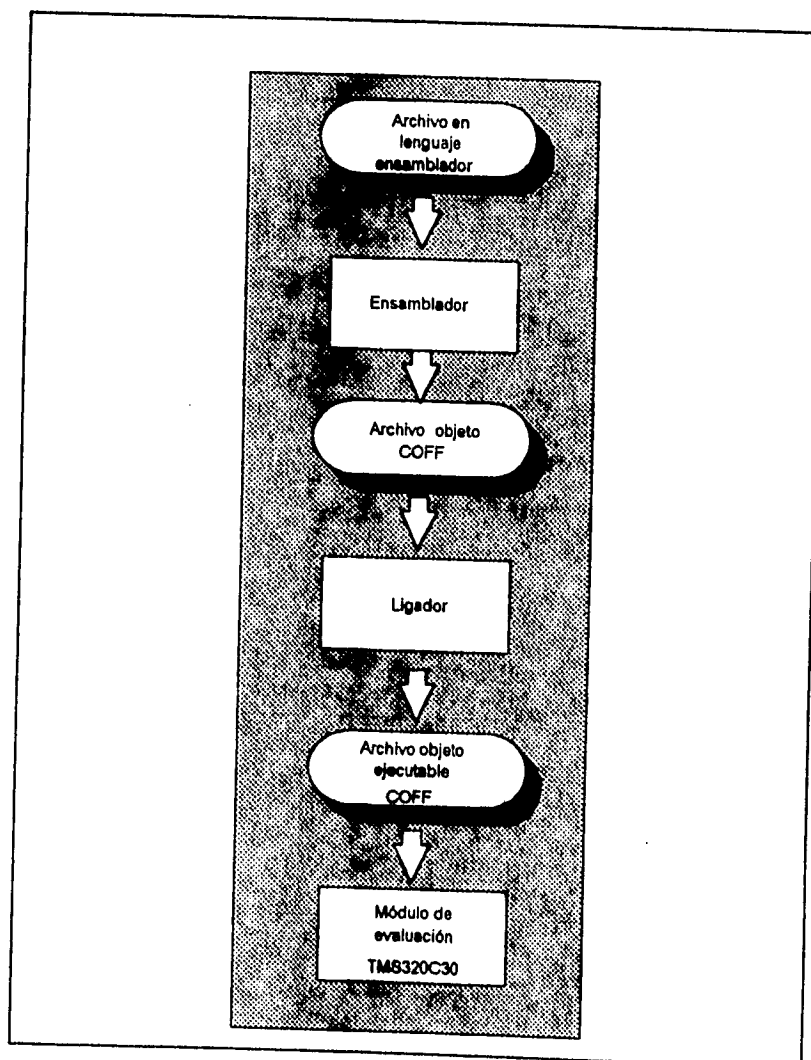


Figura B.3 Flujo de desarrollo de programas en lenguaje ensamblador.

La **figura B.3** muestra el flujo de desarrollo de un programa ensamblador. El ensamblador convierte el archivo fuente (en lenguaje ensamblador) en un archivo objeto COFF. Posteriormente el ligador combina los archivos objeto en un sólo módulo objeto ejecutable. Los pasos siguientes muestran la forma de invocar al ensamblador y al encadenador del TMS320C30:

- 1) Se escribe un archivo fuente en lenguaje ensamblador

archivo1.asm

- 2) Para ensamblar el archivo *archivo1.asm* se teclea

`asm30 archivo1 [-opciones]`

El comando `asm30` invoca al ensamblador generando un archivo objeto llamado *archivo1.obj*. En la misma línea se especifican las opciones con las que se desea compilar, por ejemplo, con la opción `-s`, el ensamblador pone todos los símbolos (constantes y etiquetas) en la tabla de símbolos del archivo objeto, generalmente sólo pone símbolos globales y con la opción `-l`, el ensamblador genera un archivo de listado.

`asm30 archivo1 -s -l`

- 3) Para ligar archivos se teclea

`lnk30 archivo1 archivo2 ... archivon [-opciones]`

El comando `lnk30` invoca al ligador, el cual combina los archivos objeto. De igual forma, se indican las opciones para ligar los programas. Para encadenar los archivos *archivo1.obj* y *archivo2.obj* se teclea la siguiente línea:

`lnk30 archivo1 archivo2 -o prog.out`

Con la opción `-o` se especifica el nombre del módulo objeto ejecutable de salida `prog.out`.

El formato COFF de los archivos objeto generados por el ensamblador y el encadenador, permite una programación modular, ya que un programa es escrito pensando en bloques de código y datos. Los bloques son

conocidos como secciones y pueden ocupar espacios continuos en el mapa de memoria. Los archivos COFF tiene tres secciones por default:

sección **.text** contiene código ejecutable
 sección **.data** contiene la inicialización de los datos
 sección **.bss** espacio reservado para variables no inicializadas

Existen dos tipos básicos de secciones:

1. Secciones inicializadas. Contiene datos o código. Las secciones **.text** y **.data** son secciones inicializadas, al igual que las secciones **.sect** y **.asect** que son creadas con las directivas del ensamblador.
2. Secciones no-inicializadas. Se refiere al espacio reservado en el mapa de memoria para datos no inicializados. Las secciones creadas con la directiva del ensamblador **.usect** son secciones no-inicializadas.

El ensamblador proporciona varias directivas que permiten asociar varias secciones de código y de datos con otras secciones más apropiadas. El ensamblador construye esas secciones durante el proceso de ensamblado, creando un archivo objeto que es organizado como se muestra en la **figura B.4**.

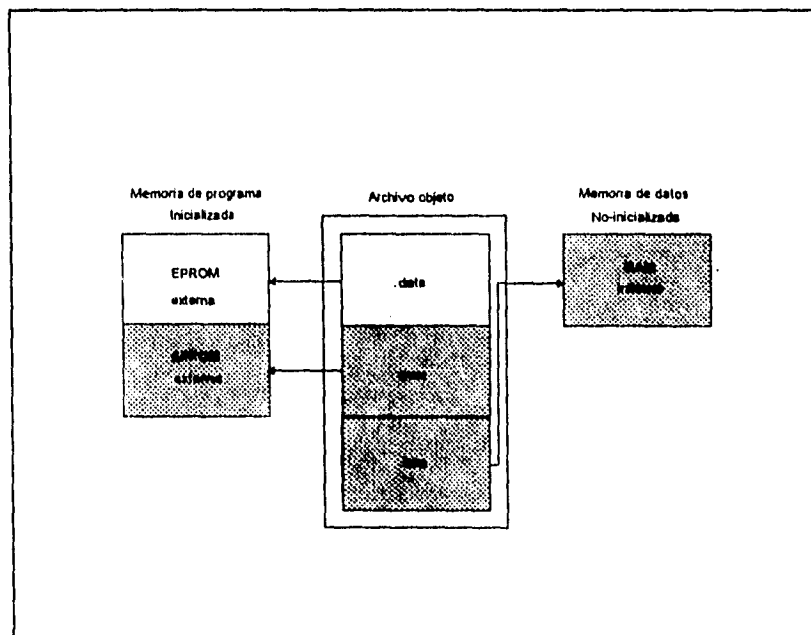


Figura B.4 Particionamiento de memoria en bloques lógicos.

La función principal del ensamblador es identificar las secciones de un programa que pertenecen a una sección particular y para realizar esto cuenta con las siguientes seis directivas:

- .bss
- .usect
- .text
- .data
- .sect
- .asect

El uso de secciones permite un manejo más eficiente del espacio de memoria. Las funciones principales del ligador son dos: 1) el ligador usa las secciones de los archivos COFF como constructores de bloques. Esto permite combinar secciones de entrada para crear secciones de salida en un módulo ejecutable y 2) el encadenador selecciona direcciones de memoria para las secciones de salida, es decir, asigna memoria a cada una de las secciones dentro del mapa de memoria. Las dos directivas que soportan estas dos funciones del ligador son:

MEMORY. Permite definir el mapa de memoria de un sistema destino. El mapa de memoria puede estar compuesto por varios tipos de memoria (DRAM, ROM, EPROM, etc.).

SECTIONS. Indica al encadenador como combinar secciones de entrada y donde poner las secciones de salida en la memoria.

Debido a que las directivas MEMORY y SECTIONS debe llamarse por medio de un archivo de comandos y a que la información del comando lnk30 puede darse también en un archivo de este tipo, toda esta información puede escribirse en un sólo archivo llamado archivo de comandos para ligar (*linker command file*) con la extensión *.cmd.

En un archivo de comandos de liga, se puede especificar también con la opción -m el nombre del archivo mapa, además del nombre de los archivos objeto (*.obj) y del archivo de salida ya ligado con la opción -o.

La figura B.5 muestra un ejemplo de un archivo de comandos de liga con las directivas MEMORY y SECTIONS.

```
/******  
/***** Archivo de comandos de encadenador (link.cmd) *****/  
/*****  
a.obj  
b.obj  
-o prog.out  
-m prog.map  
  
MEMORY  
{  
    VECS:  origin = 00000000h length = 40h  
    ROM:   origin = 00000040h length = FC0h  
    RAM0:  origin = 00801000h length = 400h  
    RAM1:  origin = 00801400h length = 400h  
}  
  
SECTIONS  
{  
    vectors: load = 00000000h  
    .text:   load = ROM  
    .data    load = ROM  
    .bss     load = Ram0  
    newvars:load = RAM1  
}
```

Figura B.5 Archivo de comandos de encadenador.

Para invocar al ligador con este archivo de comandos, se escribiría la siguiente línea:

```
lnk30 link.cmd
```

Después de que el encadenador genera los módulos objeto COFF ejecutables y se desea ejecutar un programa, los datos del módulo ejecutable deben ser transferidos o cargados al sistema de memoria destino. Las herramientas de depuración del TMS320C30 (simulador, emulador y sistemas de desarrollo) tienen un comando LOAD, el cual lee el archivo ejecutable y copia el programa en memoria.

Para el sistema de evaluación (EVMTMS30) el comando LOAD se encuentra en el programa `evmload.exe`, el cual carga el archivo de salida (archivo ejecutable) generado durante el encadenamiento y después inicia la ejecución del programa en el sistema de evaluación del TMS320C30.

Set de Instrucciones del TMS320C30

Syntax	Description	'C40		
ABSF src,dst [ABSF dst]	Absolute Value (Float)		CALL src [Call @src]	Call Subroutine
ABSI src,dst [ABSI dst]	Absolute Value (Int)		CALLcond src [Callcond @src]	Call Subroutine Cond
ADDC src,dst	Add Ints w/ Carry		CMPF src,dst	Compare Float
ADDC3 src2,src1,dst [ADDC src2,src1 ,dst]	Add Ints w/ Carry (3-Op)		CMPF3 src2,src1 [CMPF src2,src1]	Compare Float (3-OP)
ADDF src dst	Add Float		CMPI src,dst	Compare Ints
ADDF3 src2,src1,dst [ADDF src2,src1 ,dst]	Add Float (3-Op)		CMP13 src2,src1 [CMPI src2,src1]	Compare Ints (3-OP)
ADDI src,dst	Add Ints		DBcond ARn,src [DBcond ARn,@src]	Decrement & Branch Cond (standard)
ADDI3 src2,src1,dst [ADDI src2,src1,dst]	Add Ints (3-Op)		DBcondD ARn,src [DBcond ARn,@src]	Decrement & Branch Cond (Delay)
AND src,dst [AND src,dst]	Bitwise Logic AND		FIX src,dst [FIX dst]	Convert Float to Int
AND3 src2,src1,dst [AND src2,src1,dst]	Bitwise Logic AND (3-Op)		FLOAT src,dst [FLOAT dst]	Convert Int to Float
ANDN src, dst	Bitwise Logic AND w/ Complement		FRIEEE src,dst	Convert From IEEE Format ✓
ANDN3 src2,src1,dst [ANDN src2,src1,dst]	Bitwise Logic ANDN (3-Op)		IACK src	Interrupt Acknowledge
ASH count,dst	Arithmetic Shift		IDLE	Idle Until Interrupt
ASH3 count,src,dst [ASH count,src,dst]	Arithmetic Shift (3-OP)		LAJ src	Link & Jump ✓
Bcond src [Bcond @src]	Branch Cond (standard)		LAJcond src	Link & Jump Cond ✓
BcondAF src	Branch Cond Delay & Annul if False	✓	LATcond N	Link & Trap Cond ✓
BcondAT src	Branch Cond Delay & Annul if True	✓	LBb(0,1,2,3) src,dst	Load Byte, Signed ✓
BcondD src [BcondD @src]	Branch Cond (Delay)		LBUb(0,1,2,3) src,dst	Load Byte, Unsigned ✓
BR src [BR @src]	Branch UnCond		LDA src,dst	Load Address Register ✓
BRD src [BRD @src]	Branch UnCond Delay		LDE src,dst	Load Float Exponent
			LDEP src,dst	Load Int From Expansion-RF to Primary-RF ✓
			LDF src,dst	Load Float
			LDFcond src,dst	Load Float

Apéndice B EI DSP TMS320C30 B-24

	Cond			
LDFI src,dst	Load Float, Interlocked		MPYUHI src,dst	Multiply Unsigned Ints & Produce 32 MSBs ✓
LDHI src,dst	Load 16 MSBs w/ 16 Bit Immediate	✓	MPYUHI3 src1, src2, dst	Multiply Unsigned Ints & Produce 32 MSBs, 3-Op ✓
LDI src,dst	Load Int		NEGB src,dst [NEGS dst]	Negate Int w/ Borrow
LDIcond src,dst	Load Int Cond		NEGF src,dst [NEGF dst]	Negate Float
LDII src,dst	Load Int, Interlocked		NEGI src,dst [NEGI dst]	Negate Int
LDM src,dst	Load Float Mantissa		NOP	No Operation [NOP src]
LDP src,dst	Load Data Page Pointer		NORM src,dst [NORM dst]	Normalize Float
LDPE src,dst	Load Int From Primary-RF to Expansion- R F	✓	NOT src,dst [NOT dst]	Bitwise Logic Complement
LDPK src,dst	Load Data Page Pointer Immediate	✓	OR src,dst	Bitwise Logic OR
LHw(0,1) src,dst	Load Half-Word	✓	OR3 src2,src1,dst [OR src2,src1,dst]	Bitwise Logic OR (3-Op)
LHUw(0,1) src,dst	Load Half-Word Unsigned	✓	POP dst	Pop Int From Stack
LSH count,dst	Logic Shift		POPF dst	Pop Float From Stack
LSH3 count,src,dst LSH count,src,dst]	Logic Shift (3 Op)		PUSH src	Push Int on Stack
LWLct(0,1,2,3) src,dst	Load Word Left-Shifted	✓	PUSHF src	Push Float on Stack
LWRct(0,1,2,3) src,dst	Load Word Right-Shifted	✓	RCPF src,dst	Reciprocal Float ✓
MBct(0,1,2,3) src,dst	Merge Byte	✓	RETIcond	Return From Interrupt Cond or Trap Cond
MHct(0,1) src,dst	Merge Half-Word	✓	RETIcondD	Return From Interrupt Cond or Trap Cond Delay ✓
MPYF src,dst	Multiply Float		RETScnd [RETS]	Return From Subroutine Cond
MPYF3 src2,src1,dst [MPYF src2,src1,dst]	Multiply Float (3-Op)		RND src,dst [RND dst]	Round Float
MPYI src,dst	Multiply Ints		ROL dst	Rotate Left
MPYI3 src2,src1,dst [MPYI src2,src1,dst]	Multiply Ints (3-Op)		ROLc dst	Rotate Left
MPYSII src,dst	Multiply Signed Ints & Produce 32 MSBs	✓		
MPYISII3 src2,src1,dst	Multiply Signed Ints & Produce 32 MSBs (3-Op)	✓		

	Through Carry	
ROR dst	Rotate Right	
RORC dst	Rotate Right Through Carry	
RPTB src	Repeat Block of Instructions	
RPTBD src	Repeat Block of Instructions Delay	√
RPTS src	Repeat Single Instruction	
RSQR src,dst	Reciprocal of Sqr R1 Float	√
SIGI src,dst	Signal & Read Int Interlocked	
STF src,dst	Store Float	
STFI src,dst	Store Float, Interlocked	
STI src,dst	Store Int	
STII src,dst	Store Int, Interlocked	
STIK src,dst	Store Int Immediate	√
SUBB src,dst	Subtract Ints w/ Borrow	
SUBB3 src2,src1,dst {SUBB src2,src1,dst}	Subtract Ints w/ Borrow (3-Op)	
SUBC src,dst	Subtract Ints Cond	
SUBF src,dst	Subtract Float	
SUBF3 src2,src1,dst {SUBF src2,src1,dst}	Subtract Float (3-Op)	
SUBI src,dst	Subtract Ints	
SUBI3 src2,src1,dst {SUBI src2,src1,dst}	Subtract Ints (3-Op)	
SUBRB src,dst	Subtract Reverse Int w/ Borrow	
SUBRF src,dst	Subtract Reverse Float	
SUBRI src,dst	Subtract Reverse Int	
SWI	Software Interrupt	

TOIEEE src,dst	Convert To IEEE Format	√
TRAP cond N {TRAP N}	Trap Cond	
TSTB src,dst	Test Bit Fields	
TSTB3 src1,src2 {TSTB src1,src2}	Test Bit Fields (3-Op)	
XOR src,dst	Bitwise Exclusive OR	
XOR3 src2,src1,dst {XOR src2,src1,dst}	Bitwise Exclusive OR (3-Op)	

Macro Directives

Creating Macros

Mnemonic and Syntax	Description
macname .macro [parameter 1]...[parameter n]	Define macro
.mlb filename	Identify library containing macro definitions
.mexit	Go to .endm
.endm	End macro definition

Manipulating Substitution Symbols

.asg ["character string"] substitution symbol	Assign character string to substitution symbol
.eval well-defined expression, substitution symbol	Perform arithmetic on numeric substitution symbols
.var substitution symbol [...]substitution symbol]	Define local macro symbols

Conditional Assembly

.if well-definedexpression	Begin conditional assembly
.elseif well-defined expression	Optional conditional assembly block
.else	Optional conditional assembly block
.endit	End conditional assembly
.loop [well-defined expression]	Begin repeatable block assembly
.break [well-defined expression]	Optional repeatable block assembly

.endloop End repeatable block assembly

Producing Assembly-Time Messages

.emsg Send error message to standard output

.wrmsg Send warning message to standard output

.mmsg Send warning or assembly-time message to standard output

Formatting the Listing

.fcllist Allow false conditional block listing (default)

.fcnolist Inhibit false conditional block listing

.mlist Allow macro listings (default)

.mno list Inhibit macro listings

.scllist Allow expanded substitution symbol listing

.snolist Inhibit expanded substitution symbol listing (default)

OR3 || STI Bw OR value and store integer

STF || STF Store floats

STI || STI Store integers

SUBF3 || STF Subtract and store float

TOIEEE || STF Convert to IEEE format and store

SUB3 || STI Subtract and store integer

XOR3 || STI Bw XOR values and store integer

Parallel Load Instructions

Mnemonic	Description
LDF LDF	Load float
LDI LDI	Load integer

Parallel Multiply and Add/Subtract Instructions

Mnemonic	Description
MPYF3 ADDF3	Multiply and add floating-point
MPYF3 SUBF3	Multiply and subtract floating-point
MPYI3 ADDI3	Multiply and add integer
MPYI3 SUBI3	Multiply and subtract integer

Summary of Parallel Instructions

Parallel Arithmetic With Store Instructions

Mnemonic	Description	C40
ABSF STF	Absolute value and store float	
ABSI STI	Absolute value and store integer	
ADDF3 STF	Add floats and store float	
ADDI3 STI	Add integers and store integer	
AND3 STI	Bw logical-AND and store integer	
ASH3 STI	Arithmetic shift and store integer	
FIN STI	Convert float to integer and store	
FLOAT STF	Convert integer to float and store	
FRIEEE STF	Convert IEEE format and store	✓
LDF STF	Load and store float	
LDI STI	Load and store integer	
LSH3 STI	Logical shift and store integer	
MPYF3 STF	Multiply floats and store float	
MPYI3 STI	Multiply and store integer	
NEGF STF	Negate and store float	
NEGI STI	Negate and store integer	
NOT3 STI	Complement and store integer	

Referencias

Papamichalis, P., 1990. Digital Signal Processing Applications with the TMS320 Family, Volume 3. Ed. Prentice Hall and Digital Signal Processing Series Texas Instruments.

Texas Instruments, 1990. TMS320C30 Evaluation Module, Technical Reference.

Texas Instruments, 1991. TMS320 Floating-Point DSP Assembly Language Tools, User's Guide.

Texas Instruments, 1992. TMS320C3x User's Guide.

Apéndice C

Lenguaje C Concurrente

El lenguaje C concurrente es una extensión compatible del lenguaje C. Esta extensión incluye mecanismos para la creación y declaración, sincronización e interacción y terminación o finalización anormal de procesos. El lenguaje C concurrente no proporciona herramientas para la abstracción de datos, pero sus facilidades de programación en paralelo pueden usarse en unión con el lenguaje C++, lo cual da por resultado un lenguaje C++ concurrente.

La programación concurrente presenta las siguientes ventajas [Gehani, N., Roome, W., 1989]:

- Una programación conveniente y conceptualmente elegante para escribir sistemas en los cuales los eventos ocurren concurrentemente, por ejemplo en sistemas operativos, sistemas en tiempo real y bases de datos.
- Los algoritmos se expresan mejor cuando la concurrencia está declarada explícitamente, de otra manera, la estructura del algoritmo puede perderse.
- Utilización eficiente de arquitecturas multiprocesador.
- Reducción del tiempo de ejecución de un programa aún en sistemas uniprocador, debido al manejo de operaciones de E/S ejecutadas en paralelo con otros cálculos.

Para compilar y ejecutar un programa en C concurrente en un transputer con el software *Logical System C for the transputer*, versión 89.1, se requieren efectuar los siguientes pasos:

- 1) Escribir un archivo fuente en lenguaje C, *archivo.c*

- 2) Ejecutar el preprocesador indicando el nombre del archivo

```
pp archivo.c
```

El **pp** lee el código fuente (*archivo.c*) y realiza manipulaciones léxicas (compilación condicional, de macros, etc.) para generar una versión simplificada del código fuente.

- 3) Compilar el archivo generado por el preprocesador *archivo.pp* con el siguiente comando:

```
tcx archivo.pp
```

El **tcx** (compilador de C) convierte el código generado por el **pp** al lenguaje ensamblador del transputer con la extensión **.tal*.

- 4) Ensamblar el código generado por el compilador con el comando **tasm** (ensamblador).

```
tasm archivo.tal
```

El código generado puede incluirse en una librería o bien ligarlo a un archivo ejecutable.

- 5) Si se requiere ligar el archivo generado por el comando **tasm** debe emplearse el comando **tlnk**.

```
tlnk archivo.trl
```

El comando **tlnk** (ligador/cargador) soporta también múltiples librerías y lleva el control completo de la carga de direcciones para cada programa o fragmento del código de una librería. Antes de invocar este comando deben usarse los comandos **tasm**, **tlib** y **tcx**.

Por facilidad, los comandos **pp/tcx/tasm/tlink** pueden ejecutarse en un archivo por lotes:

```
lsc8 archivo.bat
```

Los archivos ejecutables generados por el encadenador (tlnk) tienen la extensión ".tld". Para cargar el archivo ejecutable en un transputer se utiliza el comando `ld-one`:

`ld-one archivo.tld cio`

El parámetro `cio` es un manejador de E/S que comunica a un programa en C que se está ejecutando en el transputer raíz con el sistema operativo del host (PC). De esta manera, mientras el transputer está ocupado en algún cálculo, la PC está disponible como servidor de E/S, ya que sin la PC el transputer no podría acceder a la pantalla de video, al teclado y a los archivos del disco duro. Los requerimientos de E/S pasan del transputer a la PC a través del canal de comunicación del transputer llamado PC/Link.

Por otro lado, si se tiene una red de transputers se utiliza el comando `ld-net` para cargar los archivos ejecutables en la red.

`ld-net archivo_cfg`

el "`archivo_cfg`" contiene una lista de parámetros, la topología de la red y la lista de programas a ser cargados en la red de transputers. El archivo de configuración tiene una extensión ".nif".

Los parámetros del archivo de configuración son:

`buffer_size`: especifica el espacio en el cual el código *bootstrap* puede almacenar los componentes de los programas que están siendo cargados. El intervalo de valores es de 16 a 255 bytes (el default es de 255).

`host-server`: Especifica el manejador de E/S del host a ser usado al momento de ejecutar el programa en el transputer. El default es "cio.exe", para manejo de ambiente gráfico el host-server es "tgio.exe".

También están los dos siguientes parámetros que constituyen un esquema de *timeout* para el diagnóstico de la configuración y manejo de errores en las redes de transputers:

`decode_timeout`: el valor de default es de 1000 milisegundos, el intervalo de valores es de 50 a 20000 milisegundos.

level_timeout: valor de default 500 milisegundos, intervalo de valores es de 25 a 1000 milisegundos.

Un ejemplo de los parámetros de configuración son:

```
buffer_size      128;
host_server      cio.exe;
decode_timeout   2000;
level_timeout    400;
```

Para describir la topología de la red se debe declarar una línea por cada nodo de la siguiente manera:

- Número de nodo
- Programa a ser cargado en el nodo
- Número de nodo que proporciona la señal de reset y si la señal se origina del sistema (R) o del subsistema (S).
- Nodos que se comunican a través de los links (canales de comunicación) con el nodo de cual se describe su configuración.

ejemplo:

1,archivo,R0,0,...:

El lenguaje C para el transputer tiene dos librerías de funciones. La primera de ellas se refiere a todas las rutinas estándar del lenguaje C de acuerdo a ANSI "C" y la segunda contiene rutinas específicas que permiten acceder a un amplio conjunto de funciones primitivas para comunicación y procesamiento en paralelo soportadas por el transputer.

La librería *conc.h* tiene todas las rutinas en lenguaje C para el manejo de concurrencia.

El transputer tiene un conjunto grande de instrucciones para implementar sistemas concurrentes. Este conjunto incluye instrucciones para inicio, terminación y paso de mensajes entre procesos. El hardware incluye características para bloquear y reiniciar la comunicación entre procesos y seleccionar nuevos procesos después de un tiempo predeterminado, y si es el caso colocarlos también al final de una cola activa. A continuación

level_timeout: valor de default 500 milisegundos, intervalo de valores es de 25 a 1000 milisegundos.

Un ejemplo de los parámetros de configuración son:

```
buffer_size      128;
host_server      cio.exe;
decode_timeout   2000;
level_timeout    400;
```

Para describir la topología de la red se debe declarar una línea por cada nodo de la siguiente manera:

- Número de nodo
- Programa a ser cargado en el nodo
- Número de nodo que proporciona la señal de reset y si la señal se origina del sistema (R) o del subsistema (S).
- Nodos que se comunican a través de los links (canales de comunicación) con el nodo de cual se describe su configuración.

ejemplo:

```
1,archivo,R0,0,...;
```

El lenguaje C para el transputer tiene dos librerías de funciones. La primera de ellas se refiere a todas las rutinas estándar del lenguaje C de acuerdo a ANSI "C" y la segunda contiene rutinas específicas que permiten acceder a un amplio conjunto de funciones primitivas para comunicación y procesamiento en paralelo soportadas por el transputer.

La librería *conc.h* tiene todas las rutinas en lenguaje C para el manejo de concurrencia.

El transputer tiene un conjunto grande de instrucciones para implementar sistemas concurrentes. Este conjunto incluye instrucciones para inicio, terminación y paso de mensajes entre procesos. El hardware incluye características para bloquear y reiniciar la comunicación entre procesos y seleccionar nuevos procesos después de un tiempo predeterminado, y si es el caso colocarlos también al final de una cola activa. A continuación

se describen las funciones para el manejo de concurrencia, comunicación entre canales y otras funciones.

Procesos y manejo de concurrencia

Antes de que un proceso pueda ser ejecutado, debe asignársele un espacio del *stack*. El espacio del *stack* para un proceso se asigna usando la función `malloc()`. La asignación de espacio para un nuevo proceso se hace por medio de la función `ProcAlloc()`, la cual genera un apuntador a una función que contiene el código del proceso y usa la función `malloc()` para asignar espacio a la estructura del proceso y al *stack*.

Con la función `Proclnit()` se inicializa la estructura del proceso y el espacio de trabajo. Después de que a un proceso se le ha asignado un espacio en memoria o es inicializado, es posible alterar sus parámetros con la función `ProcParam()`. La siguiente lista muestra las rutinas para manejo de concurrencia en el transputer:

<code>ProcAlloc</code>	Asignación dinámica de procesos
<code>ProcFree</code>	Liberación de asignación dinámica de procesos
<code>Proclnit</code>	Inicialización de procesos
<code>ProcPar</code>	Inicia la ejecución de un proceso o procesos
<code>ProcParam</code>	Modificación de parámetros de un proceso
<code>ProcParList</code>	Inicia ejecución de una lista de procesos
<code>ProcPriPar</code>	Inicia ejecución de procesos con prioridades mezcladas
<code>ProcRun</code>	Inicia ejecución de procesos con la prioridad actual
<code>ProcRunHigh</code>	Inicia ejecución de procesos con alta prioridad
<code>ProcRunLow</code>	Inicia ejecución de procesos con baja prioridad
<code>ProcStop</code>	Para los procesos
<code>ProcToHigh</code>	Proceso actual cambia a alta prioridad
<code>ProcToLow</code>	Proceso actual cambia a baja prioridad

Canales de comunicación

El transputer soporta un protocolo de paso de mensajes para la comunicación entre procesos. Un canal es un flujo de mensajes unidireccional entre dos procesos. Cuando un proceso ejecuta una entrada o una salida a un canal, el proceso es bloqueado hasta que el otro proceso al cual se intenta comunicar ejecute una salida o una

entrada. De esta forma los canales se usan como un mecanismo de sincronización dentro de la comunicación. Existe únicamente una restricción al usar los canales de comunicación, esta se refiere a que los dos procesos a los cuales se intenta comunicar deben ejecutar operaciones de la misma longitud de datos. Las siguientes rutinas se refieren a la comunicación entre canales:

ChanAlloc	Asignación dinámica de canales
ChanFree	Liberación de la asignación dinámica
ChanIn	Leer mensaje de un canal
ChanInChanFail	Leer mensaje de un canal con un reset
ChanInChar	Leer byte de un canal
ChanInInt	Leer una palabra de un canal
ChanInTimeFail	Leer mensaje de un canal con un timeout
ChanOut	Escribir un mensaje a un canal
ChanOutChanFail	Escribir un mensaje a un canal con un reset
ChanOutChar	Escribir un byte a un canal
ChanOutInt	Escribir una palabra a un canal
ChanOutTimeFail	Escribir un mensaje a un canal con timeout
ChanReset	Inicializar un canal

Los cuatro canales de comunicación del transputer se asocian a 8 apuntadores para facilitar su manejo a nivel de software, cuatro apuntadores para entrada y cuatro para salida, cada uno con una dirección específica de hardware. Estas direcciones se encuentran definidas en la librería conc.h.

```
#define LINK0OUT ((Channel *) 0x80000000)
#define LINK1OUT ((Channel *) 0x80000004)
#define LINK2OUT ((Channel *) 0x80000008)
#define LINK3OUT ((Channel *) 0x8000000c)
#define LINK0IN  ((Channel *) 0x80000010)
#define LINK1IN  ((Channel *) 0x80000014)
#define LINK2IN  ((Channel *) 0x80000018)
#define LINK3IN  ((Channel *) 0x8000001c)
```

Para determinar el status de los canales y la posibilidad de esperar hasta que una canal esté listo para una entrada, existen las siguientes rutinas:

ProcAlt	Bloquea los procesos hasta que uno de los canales de entrada esté disponible
---------	--

ProcAltList	Bloquea los procesos hasta que uno de los canales de entrada esté disponible
ProcSkipAlt	Checan si un canal de entrada está disponible
ProcSkipAltList	Checan si un canal de entrada está disponible
ProcTimerAlt	Bloquea los procesos hasta que uno de los canales está disponible para entrada o el valor del reloj especificado sea alcanzado
ProcTimerAltList	Bloquea los procesos hasta que uno de los canales está disponible para entrada o el valor del reloj especificado sea alcanzado

Otras funciones

Existen otras funciones como la función `time()` que proporciona el valor del reloj. Este valor es diferente para procesos de prioridad alta y procesos de prioridad baja. El reloj para prioridad baja es incrementado cada 64 μ s y para prioridad alta cada 1 μ s. La ejecución de un proceso puede ser bloqueada durante un tiempo específico usando la función `ProcAfter()` o puede suspenderse por un número de períodos de reloj con la función `ProcWait()`. Otras funciones son `ProcGetPriority()` para obtener la prioridad de un proceso y `ProcReschedule()` para reorganizar un proceso en una cola activa de procesos.

Por otro lado, para desarrollar un ambiente gráfico en un transputer o plataforma de transputers con lenguaje C concurrente, existe una librería gráfica llamada `t8graph.ttl`. Esta librería maneja tres archivos que deben incluirse al inicio del archivo fuente si se desea manejar funciones en modo gráfico, modo texto, manejo de mouse o sonido.

<code>#include "graph.h"</code>	Modo gráfico
<code>#include "text.h"</code>	Modo texto y sonido
<code>#include "mouse.h"</code>	Manejo del mouse

Modo gráfico: Soporta los modos EGA y VGA, para el modo EGA la resolución es de 640x350 pixeles y para VGA de 640x480 pixeles.

Modo texto: Permite la ejecución de operaciones tipo texto en color y en blanco y negro, con 40 o 80 columnas en la pantalla.

Manejo del mouse: Las rutinas para el manejo del mouse permiten la detección de los botones del mouse y el movimiento del cursor. Utiliza un mouse estándar *Microsof*.

Manejo de sonido: Se basa en la bocina de una IBM PC de forma limitada. El sonido puede ser activado a diferentes frecuencias y su duración depende del usuario.

Básicamente las funciones que se manejan para el ambiente gráfico son las mismas del lenguaje C estándar.

Referencias

Computer System Architects, 1990. Transputer Education Kit, User Guide, Theory of Operation, Installation, Schematics.

Computer System Architects, 1990. Logical System C for the Transputer: Version 89.1 User Manual.

Computer System Architects, 1991. Transputer Graphics Development Package for the Inmos transputer and logical System C. Version 1.0.

Gehani, N., Roome, W., 1989. The Concurrent C Programming Language. AT&T Bell Telephone Laboratories.

Apéndice D

Listados de Programas

D.1 Programa nodo_1 sistema homogéneo y heterogéneo

```
/*-----*/
/* Programa: APMASHO.C (Nodo_1) */
/* Descripción: Manejo del ambiente gráfico y mouse, lectura
/* del archivo de datos sintéticos en la unidad
/* de disco duro, modificación de los parámetros
/* del sistema, despliegue gráfico de la señal
/* Doppler y transferencia de mensajes (datos)
/* al Nodo_2 */
/*-----*/
```

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include <ctype.h>
#include <graph.h>
#include <mouse.h>
#include <con.h>
#include <text.h>
#include <time.h>
```

```
** Declaración de constantes **
#define NULL 0
#define TRUE 1
#define FALSE 0
#define MAXCOLORS 15
```

```
int menu(int nopciones,int c_barra1,int c_letrero1,int c_barra2,int c_letrero2,int marco);
void menu_principal();
int *salva_vent(int x1,int y1,int x2,int y2);
void limpia(int fondo, int marco,int x1,int y1,int x2,int y2);
void restaura_vent(int x, int y, int *iptr);
void lee_datos_arch();
void espectrograma();
void escala_tiempo();
void escala_frecuencia();
void muestra_colores();
void long_fft();
void frecuencia_de_muestreo();
void periodo_card();
void colores_grafica();
void parametros_eje_tiempo();
void byte_de_reconocimiento();
void comunicacion();
void despliega_ventana(int seg);
void obten_color();
float max(float valor_inicial, float p[512]);
void calcula_fft();
void letreros(int x1, int y1,char msg1[20],int color);
float getstring(int x, int y);
int getstringi(int x, int y);
void despliega_tiempos();

float periodo_cardiaco = 700; /* [ms] */
float frec_muestreo = 12.8; /* [KHz] */
int frame_size = 256;
int max_color = 7;
int num_color = 7;
float eje_tiempo = 0.20; /* s/Div */

float arch_datos[8960];
int dato_color[512];
int tecla_pres[1];
int tiempos[10];
int t_p[10];
int continua[1];
int opc_selec[1];
int fin_prog[1];
int memoria_in;
int cont_seg;
int running;
float *datos_resultado;

signed char colores[] = {64, 16, 2, 114, 22, 30, 23, 63,9, 54, 7, 4, 38, 69, 11, 36};
```

```

signed char colores_b[] = {0, 1, 2, 3, 4, 5, 20, 7, 56, 57, 58, 59, 60, 61, 62, 63};

int negro = 0, verde1 = 1, verde2 = 2, verde3 = 3, verde4 = 4,
verde5 = 5, verde6 = 6, verde7 = 7, d_blue = 8, amarillo = 9,
gris = 10, marron = 11, d_naranja = 12, violeta = 13,
l_azul = 14, l_naranja = 15;

int x_ventana[4096],
y_ventana[1024];
int xviewport = 500;
int yviewport = 300;

float dt, df;
float max_tiempo_eje_frec, max_frec;
int mitad_frame;
int segmentos, nsegmentos, segmenlong;

struct palettetype p, pd;

FILE *fpt;
char dato[30];

unsigned long t;
int count, x, y;
int status;
int result;

struct opos{
    int x1;
    int y1;
    int x2;
    int y2;
    char msg[20];
}posvent[20];

/*-----*/
/*                      Programa principal                      */
/*-----*/

void main() {
    int gdriver = DETECT, gmode;
    int opcion;
    char error[80];
    int errorcode;
    int ch;
    int bandera = 1;
    int j;

```

```

if( _node_number == 1 ) /* Nodo_1 */
{
    lee_datos_arch();
    initgraph(&gdriver, &gmode, "");
    errorcode = graphresult();
    if( errorcode != grOk ) {
        grapherrormsg(errorcode, error);
        printf("Graphics System Error : %s\n", error);
        exit(1);
    }
    result = MouseReset();
    if( !result ) {
        printf("No hay mouse instalado. Adios!\n");
        exit(2);
    }

    ChanOut(LINK1OUT, (char *)arch_datos, sizeof(arch_datos));

    do {
        setpalette(0, EGA_BLUE);

        posvent[0].x1 = 40;
        posvent[0].y1 = 10;
        posvent[0].x2 = 180;
        posvent[0].y2 = 30;
        strcpy(posvent[0].msg, "PARAMETROS");

        posvent[1].x1 = 220;
        posvent[1].y1 = 10;
        posvent[1].x2 = 360;
        posvent[1].y2 = 30;
        strcpy(posvent[1].msg, "ESPECTROGRAMA");

        posvent[2].x1 = 400;
        posvent[2].y1 = 10;
        posvent[2].x2 = 540;
        posvent[2].y2 = 30;
        strcpy(posvent[2].msg, "F I N");

        setfillstyle(SOLID_FILL, EGA_LIGHTGRAY);
        bar(4, 5, 634, 35);
        opcion = menu(3, EGA_LIGHTGRAY, EGA_YELLOW,
                    EGA_LIGHTGRAY, BLUE, EGA_WHITE);
        opc_selec[0] = opcion;
        ChanOut(LINK1OUT, (char *)opc_selec, sizeof(opc_selec));
        switch(opcion){
            case 1 : parametros();

```

```

        continua[0] = TRUE;
        ChanOut(LINK1OUT, (char *)continua,sizeof(continua));
        limpia(EGA_BLUE,BLUE,1,40,639,479);
        break;
    case 2 : HideMouseCursor();
             espectrograma();
             continua[0] = TRUE;
             ChanOut(LINK1OUT, (char *)continua,sizeof(continua));
             limpia(EGA_BLUE,BLUE,1,40,639,479);
             ShowMouseCursor();
             break;
    case 3 : bandera = 0;
             continua[0] = FALSE;
             ChanOut(LINK1OUT, (char *)continua,sizeof(continua));
             break;
    }
    }while(bandera);

    HideMouseCursor(0);
    closegraph();
}
ChanIn(LINK1IN, (char *)fin_prog,sizeof(fin_prog));
if( fin_prog[0] )
    printf("FINAL DEL PROGRAMA\n");
else
    printf("ERROR .....n");
}
/*-----*/
/*                               Menu principal                               */
/*-----*/

int menu(int nopciones,int c_barra1,int c_letrero1, int c_barra2,int c_letrero2,int marco) {
    int boton=0;
    int i;

    settxtjustify(CENTER_TEXT,CENTER_TEXT);
    settxtstyle(SMALL_FONT,HORIZ_DIR,10);
    setcolor(c_letrero1);

    for( i = 0; i < nopciones; i++ ){
        moveto(posvent[i].x1+((posvent[i].x2-posvent[i].x1) >> 1 ),
              posvent[i].y1+((posvent[i].y2-posvent[i].y1) >> 1 ));

        outtext(posvent[i].msg);
    }

    HideMouseCursor(); /* Oculta el cursor del mouse */
    ShowMouseCursor(); /* Muestra el cursor del mouse */
    i=0;

```

```

do {
    if( entra_ventana(posvent[i].x1,posvent[i].y1,
                     posvent[i].x2,posvent[i].y2) ){
        HideMouseCursor();
        setfillstyle(SOLID_FILL,c_barra2);
        bar(posvent[i].x1,posvent[i].y1,posvent[i].x2,posvent[i].y2);
        setcolor(marco);
        rectangle(posvent[i].x1,posvent[i].y1,posvent[i].x2,posvent[i].y2);
        setcolor(c_letrero2);
        moveto(posvent[i].x1+((posvent[i].x2-posvent[i].x1) >> 1 ),
              posvent[i].y1+((posvent[i].y2-posvent[i].y1) >> 1 ));

        outtext(posvent[i].msg);
        ShowMouseCursor();
        boton=sal_ventana(posvent[i].x1,posvent[i].y1,
                         posvent[i].x2,posvent[i].y2);

        if( boton ) boton=i+1;
        HideMouseCursor();
        setfillstyle(SOLID_FILL,c_barra1);
        bar(posvent[i].x1,posvent[i].y1,posvent[i].x2,posvent[i].y2);
        setcolor(c_letrero1);
        moveto(posvent[i].x1+((posvent[i].x2-posvent[i].x1) >> 1 ),
              posvent[i].y1+((posvent[i].y2-posvent[i].y1) >> 1 ));

        outtext(posvent[i].msg);
        ShowMouseCursor();
    };
    i++;
    if(i==nopciones)j=0;
} while(!boton);
return(boton);
}
/*-----*/
/*                               Función parámetros                               */
/*-----*/

void parametros() {
    int i;
    int ch,opcion;
    int bandera = 1;

    setfillstyle(SOLID_FILLEGA_DARKGRAY);
    bar(34,45,185,230);
    setcolor(EGA_GREEN);
    rectangle(34,45,185,230);

    do {
        posvent[0].x1 = 40;
        posvent[0].y1 = 50;
        posvent[0].x2 = 180;
        posvent[0].y2 = 70;

```



```

strcpy(posvent[0].msg,"LONG. FFT");

posvent[1].x1 = 40;
posvent[1].y1 = 80;
posvent[1].x2 = 180;
posvent[1].y2 = 100;
strcpy(posvent[1].msg,"FREC. MUESTREO");

posvent[2].x1 = 40;
posvent[2].y1 = 110;
posvent[2].x2 = 180;
posvent[2].y2 = 130;
strcpy(posvent[2].msg,"PERIODO CARDIACO");

posvent[3].x1 = 40;
posvent[3].y1 = 140;
posvent[3].x2 = 180;
posvent[3].y2 = 160;
strcpy(posvent[3].msg,"COLORES");

posvent[4].x1 = 40;
posvent[4].y1 = 170;
posvent[4].x2 = 180;
posvent[4].y2 = 190;
strcpy(posvent[4].msg,"EJE TIEMPO");

posvent[5].x1 = 40;
posvent[5].y1 = 200;
posvent[5].x2 = 180;
posvent[5].y2 = 220;
strcpy(posvent[5].msg,"SALIDA");

opcion=menu(6,EGA_DARKGRAY,EGA_YELLOW,EGA_LIGHTGRAY,BLUE,EGA_WHITE);

HideMouseCursor();
switch( opcion ) {
    case 1 : long_fft();
            break;
    case 2 : frecuencia_de_muestreo();
            break;
    case 3 : periodo_card();
            break;
    case 4 : colores_grafica();
            break;
    case 5 : parametros_eje_tiempo();
            break;
    case 6 : limpia(EGA_BLUE,BLUE,34.45,185,230);
            bandera = 0;
}

```

```

        break;
    }
}while(bandera);
}
/*-----*/
/*                Parámetro: longitud de FFT                */
/*-----*/
void long_fft() {
    int i;
    int ch,opcion;
    int bandera = 1;

    setfillstyle(SOLID_FILL,EGA_DARKGRAY);
    bar(400,50,550,230);
    setcolor(EGA_GREEN);
    rectangle(400,50,550,230);
    setttextjustify(CENTER_TEXT,CENTER_TEXT);
    setttextstyle(SMALL_FONT,HORIZ_DIR,10);
    setcolor(EGA_YELLOW);
    outtextxy(475,65,"SIZE FFT");
    do {
        posvent[0].x1 = 405;
        posvent[0].y1 = 85;
        posvent[0].x2 = 545;
        posvent[0].y2 = 105;
        strcpy(posvent[0].msg,"64");

        posvent[1].x1 = 405;
        posvent[1].y1 = 115;
        posvent[1].x2 = 545;
        posvent[1].y2 = 135;
        strcpy(posvent[1].msg,"128");

        posvent[2].x1 = 405;
        posvent[2].y1 = 145;
        posvent[2].x2 = 545;
        posvent[2].y2 = 165;
        strcpy(posvent[2].msg,"256");

        posvent[3].x1 = 405;
        posvent[3].y1 = 175;
        posvent[3].x2 = 545;
        posvent[3].y2 = 195;
        strcpy(posvent[3].msg,"512");

        posvent[4].x1 = 405;
        posvent[4].y1 = 205;

```

```

posvent[4].x2 = 545;
posvent[4].y2 = 225;
strcpy(posvent[4].msg,"1024");

opcion=menu(5,EGA_DARKGRAY,EGA_YELLOW,EGA_LIGHTGRAY,BLUE,EGA_WHITE);

HideMouseCursor();
switch( opcion ) {
case 1 : frame_size = 64;
        limpia(EGA_BLUE,BLUE,400,50,550,230);
        bandera = 0;
        break;
case 2 : frame_size = 128;
        limpia(EGA_BLUE,BLUE,400,50,550,230);
        bandera = 0;
        break;
case 3 : frame_size = 256;
        limpia(EGA_BLUE,BLUE,400,50,550,230);
        bandera = 0;
        break;
case 4 : frame_size = 512;
        limpia(EGA_BLUE,BLUE,400,50,550,230);
        bandera = 0;
        break;
case 5 : frame_size = 1024;
        limpia(EGA_BLUE,BLUE,400,50,550,230);
        bandera = 0;
        break;
}
}while(bandera);
}
}
/*-----*/
/*          Parámetro: Frecuencia de muestreo          */
/*-----*/
void frecuencia_de_muestreo() {
char ch;
float t_frecuencia_muestreo;
static char str1[30];
static char str2[30];

setfillstyle(SOLID_FILL,EGA_DARKGRAY);
bar(250,200,550,350);
setcolor(EGA_GREEN);
rectangle(250,200,550,350);
setcolor(EGA_YELLOW);
settextstyle(SMALL_FONT,HORIZ_DIR,10);
settextjustify(CENTER_TEXT,CENTER_TEXT);

```

```

outtextxy(400,220,"FRECUENCIA DE MUESTREO");
settextjustify(LEFT_TEXT, TOP_TEXT);
sprintf(str1,"Valor Actual: %4.1f KHz",frec_muestreo);
outtextxy(300,280,str1);
outtextxy(300,310,"Nuevo Valor:   KHz");
t_frecuencia_muestreo = getstringf(410,310);
if( t_frecuencia_muestreo != 0 )
    frec_muestreo = t_frecuencia_muestreo;
setfillstyle(SOLID_FILL,EGA_BLUE);
bar(250,200,550,350);
}
/*-----*/
/*          Parámetro: Periodo de muestreo          */
/*-----*/
void periodo_card() {
char ch;
float t_periodo_cardiaco;
static char str1[30];
static char str2[30];

setfillstyle(SOLID_FILL,EGA_DARKGRAY);
bar(250,200,550,350);
setcolor(EGA_GREEN);
rectangle(250,200,550,350);
setcolor(EGA_YELLOW);
settextstyle(SMALL_FONT,HORIZ_DIR,10);
settextjustify(CENTER_TEXT,CENTER_TEXT);
outtextxy(400,220,"PERIODO CARDIACO");
settextjustify(LEFT_TEXT, TOP_TEXT);
sprintf(str1,"Valor Actual: %4.1f ms",periodo_cardiaco);
outtextxy(300,280,str1);
outtextxy(300,310,"Nuevo Valor:   ms");
t_periodo_cardiaco = getstringf(410,310);
if( t_periodo_cardiaco != 0 )
    periodo_cardiaco = t_periodo_cardiaco;
setfillstyle(SOLID_FILL,EGA_BLUE);
bar(250,200,550,350);
}
/*-----*/
/*          Parámetro: Número de colores          */
/*-----*/
void colores_grafica() {
char ch;
int t_num_color;
static char str1[30];
static char str2[30];

setfillstyle(SOLID_FILL,EGA_DARKGRAY);

```

```

bar(250,200.550,350);
setcolor(EGA_GREEN);
rectangle(250,200,550,350);
setcolor(EGA_YELLOW);
settextstyle(SMALL_FONT,HORIZ_DIR,10);
setttextjustify(CENTER_TEXT,CENTER_TEXT);
outtextxy(400,220,"NUMERO DE COLORES");
setttextjustify(LEFT_TEXT, TOP_TEXT);
sprintf(str1,"Valor Actual: %4d ",num_color+1);
outtextxy(300,280,str1);
outtextxy(300,310,"Nuevo Valor: ");
t_num_color = getstringi(410,310);
if( t_num_color != 0 )
    num_color = t_num_color - 1;
setfillstyle(SOLID_FILL,EGA_BLUE);
bar(250,200.550,350);
}
/*-----*/
/*                               */
/*                               */
/*-----*/
void parametros_eje_tiempo() {
char ch;
float t_eje_tiempo;
static char str1[30];
static char str2[30];

setfillstyle(SOLID_FILL,EGA_DARKGRAY);
bar(250,200,550,350);
setcolor(EGA_GREEN);
rectangle(250,200,550,350);
setcolor(EGA_YELLOW);
settextstyle(SMALL_FONT,HORIZ_DIR,10);
setttextjustify(CENTER_TEXT,CENTER_TEXT);
outtextxy(400,220,"EJE TIEMPO");
setttextjustify(LEFT_TEXT, TOP_TEXT);
sprintf(str1,"Valor Actual: %5.3f s/D ",t_eje_tiempo);
outtextxy(300,280,str1);
outtextxy(300,310,"Nuevo Valor: s/D");
t_eje_tiempo = getstringf(410,310);
if( t_eje_tiempo != 0 )
    t_eje_tiempo = t_eje_tiempo;
setfillstyle(SOLID_FILL,EGA_BLUE);
bar(250,200.550,350);
}
/*-----*/
/*                               */
/*                               */
/*-----*/
float getstringf(int x, int y) {

```

```

char s2[30];
char ch;
int i;
char str[2];
int entero;

i=0;
str[1] = '\0';
while(1) {
    ch=getch();
    if( ch == 0x1B ) break;
    if( ch == '\n' || ch == '\r' ) break;
    if( ch == 8 ) {
        if( --i < 0 ) i=0;
        setfillstyle(SOLID_FILL,EGA_DARKGRAY);
        bar(x+10*i,y,x+10+10*i,y+10);
    }
    else {
        entero = (int)ch;
        if( isdigit(entero) ) {
            s2[i] = str[0] = ch;
            outtextxy(x+10*i,y,str);
            i++;
        }
        else {
            if( ch == '.' ){
                s2[i] = str[0] = ch;
                outtextxy(x+10*i,y,str);
                i++;
            }
            else
                continue;
        }
    } /* end del else */
} /* end del while */
s2[i] = '\0';
return(atof(s2));
}
/*-----*/
/*                               */
/*                               */
/*-----*/
Obtiene números enteros de la pantalla
/*-----*/
int getstringf(int x, int y) {
char s2[30];
char ch;
int i;
char str[2];
int entero;

```

```

i=0;
str[1] = '\0';
while(1){
    ch=getch();
    if( ch == 0x1B ) break;
    if( ch == '\n' || ch == '\r' ) break;
    if( ch == 8 ) {
        if( -i < 0 ) i=0;
        setfillstyle(SOLID_FILL,EGA_DARKGRAY);
        bar(x+10*i,y,x+10+10*i,y+10);
    }
    else {
        entero = (int)ch;
        if( isdigit(entero) ) {
            s2[i] = str[0] = ch;
            outtextxy(x+10*i,y,str);
            i++;
        }
        else
            continue;
    } /* end del else */
} /* end del while */
s2[i] = '\0';
return(atoi(s2));
}
/*-----*/
/*                               Espectrograma                               */
/*-----*/
void espectrograma() {
    int i,j;
    char ch;
    float datos_parametros[2];
    float datos_parametros2[2];
    int frame2;
    int frec2;
    char cadena5[4]="On ";
    char cadena6[4]="Off";
    char cadena7[4]=" ";

    char str1[30];
    char str2[30];
    char str3[30];
    char str4[30];
    char str5[30];
    char str6[30];
    char str7[30];
    char str8[30];
    char str13[4];

```

```

char str14[4];
char str15[4];

limpia(EGA_BLUE, BLUE,1,40,639,479);
for(j=0; j<MAXCOLORS+1; j++)
    p.colors[j] = colores[j];
p.size = 16;
setpalette(&p);
limpia(negro,negro,1,40,639,479);

dt = frame_size/frec_muestreo; /* [ms] */
df = (frec_muestreo*1000.0)/frame_size; /* [KHz] */
mitad_frame = frame_size/2;
max_tiempo = eje_tiempo * 10.00; /* [ms] */
eje_frec = frec_muestreo * 0.05; /* [KHz] */
max_frec = frec_muestreo/2.0; /* [KHz] */
segmentos = periodo_cardiaco/dt;
nsegmentos = max_tiempo*1000.0/dt;
segmentlong = frame_size/2;
datos_parametros[0] = frame_size + 0.0;
datos_parametros[1] = frec_muestreo;
continua[0] = TRUE;
datos_resultado = (float *) malloc(mitad_frame*sizeof(float));
memoria_in = mitad_frame*sizeof(float);

ventanas();
muestra_colores();
setcolor(d_blue);
rectangle(1,40,639,479);
escala_tiempo();
escala_frecuencia();
rectangle(119,59,621,361);
settextjustify(LEFT_TEXT, TOP_TEXT);
letreros(370,375,"T I E M P O",amarillo);
letreros(30,211,"FRECUENCIA",amarillo);
letreros(30,420,"STATUS",marron);
letreros(50,355,"0.0 KHz",amarillo);
settextstyle(SMALL_FONT,HORIZ_DIR,10);
setcolor(marron);
sprintf(str1,"Periodo card: %4.1f ms",periodo_cardiaco);
outtextxy(120,420,str1);
sprintf(str2,"Framesize: %4d",frame_size);
outtextxy(120,430,str2);
sprintf(str3,"Num. Colores: %2d",num_color+1);
outtextxy(120,440,str3);
sprintf(str4,"Frec. Muestreo: %4.1f KHz",frec_muestreo);
outtextxy(320,420,str4);
sprintf(str5,"Estima: FFT");

```

```

outtextxy(320,430,str5);
setcolor(amarillo);
sprintf(str6,"%4.1f KHz",max_freq);
outtextxy(30,80,str6);
sprintf(str7,"%4.2f s/D",eje_tiempo);
outtextxy(540,375,str7);
sprintf(str8,"%4.2f KHz/D",eje_freq);
outtextxy(30,300,str8);
sprintf(str13,"%3s",cadena5);
sprintf(str14,"%3s",cadena6);
sprintf(str15,"%3s",cadena7);

letreros(540,460,"ESC Salir",l_azul);
letreros(120,460,"[H] Hold",l_azul);
letreros(320,460,"HOLD: Off",marron);

ChanOut(LINK1OUT, (char *)datos_parametros,sizeof(datos_parametros));
tecla_pres[0] = FALSE;
cont_seg = 0;
running = TRUE;
while(running) {
    for( i=0; i<nsegmentos; i++ ) {
        if( cont_seg == segmentos ) cont_seg = 0;
        ChanOut(LINK1OUT, (char *) tecla_pres,sizeof(tecla_pres));
        if( tecla_pres[0] ) {
            ChanIn(LINK1IN, (char *) datos_resultado, memoria_in);
            for( j=0; j<10; j++ )
                tiempos[j] = datos_resultado[j];
            tiempos[6] = t_p[6];
            tiempos[7] = t_p[7];
            running = FALSE;
            break;
        }
        else {
            ChanIn(LINK1IN, (char *) datos_resultado, memoria_in);
            t_p[6] = Time();
            obten_color();
            despliega_ventana(i);
            t_p[7] = Time();
            cont_seg++;
            if( !kbhit() )
                tecla_pres[0] = FALSE;
            else {
                ch = toupper(getch());
                if( ch == 'H'){
                    setfillstyle(SOLID_FILL, negro);
                    bar(365,455,395,470);
                    setcolor(marron);

```

```

outtextxy(370,460,str13);
do{
    while( !kbhit() ) {
        }
        ch = toupper(getch());
    }while(ch != 'H');
    setfillstyle(SOLID_FILL, negro);
    bar(365,455,395,470);
    setcolor(marron);
    outtextxy(370,460,str14);
    tecla_pres[0] = FALSE;
}
else{
    if( ch == 0x1B )
        tecla_pres[0] = TRUE;
    else
        tecla_pres[0] = FALSE;
}
} /* end del else */
} /* end del else */
} /* end del for */
} /* while */

free(datos_resultado);
limpia(negro,negro,1,40,639,479);
for( j=0; j<MAXCOLORS+1; j++ )
    p.colors[j] = colores_b[j];
p.size = 16;
setallpalette(&p);
}
/*-----*/
/*                               Despliegue de los tiempos de ejecución                               */
/*-----*/
void despliega_tiempos() {
    int j;
    struct estruc_tiempos {
        char str[30];
    } tab_tiempos[10];

    setcolor(amarillo);
    for( j=0; j<10; j++ ) {
        sprintf(tab_tiempos[j].str,"tiempo[%d] = %d ",j,tiempos[j]);
        outtextxy(30,80+20*j,tab_tiempos[j].str);
    }
}
/*-----*/
/*                               Muestra escala de colores                               */
/*-----*/

```

```

void muestra_colores() {
    int j,i;

    for( j = 0, i = num_color, j < num_color+1; i--, j++ ) {
        setfillstyle(SOLID_FILL, i);
        bar(622,60+20*j,638,80+20*j);
    }
    setcolor(d_blue);
    rectangle(622,60,638,80+20*(j-1));
}
/*-----*/
/*                      Dibuja escala de tiempo                      */
/*-----*/
void escala_tiempo() {
    int i;

    for( i=0; i < 10; i++ )
        line(i*50+119,361,i*50+119,363);
}
/*-----*/
/*                      Dibuja escala en frecuencia                      */
/*-----*/
void escala_frecuencia() {
    int i;

    for( i=0; i < 10; i++ )
        line(117,361-i*30,119,361-i*30);
}
/*-----*/
/*                      Cálculo de la posición de ventanas                      */
/*-----*/
void ventanas() {
    int ij;

    for( i = 0; i < nsegmentos+1; i++ )
        x_ventana[i] = ((i*dt)*xviewport)/(max_tiempo*1000.0);
    for( j = 0; j < mitad_frame+1; j++ ) {
        y_ventana[j] = ((j*df)*yviewport)/(max_frec*1000.0);
        y_ventana[j] = yviewport - y_ventana[j];
    }
}
/*-----*/
/*                      Cuantización de los datos procesados en la escala de colores                      */
/*-----*/
void obten_color() {
    int j;
    float max_valor = 0.0;
    float max_potencia;

```

```

float cuantiza;

max_potencia = max(max_valor,datos_resultado);
cuantiza = max_potencia/num_color;
for( j = 0; j < mitad_frame; j++ )
    dato_color[j] = datos_resultado[j]/cuantiza;
}
/*-----*/
/*                      Encuentra el valor máximo de los datos procesados                      */
/*-----*/
float max(float valor_inicial, float p[512]) {
    int i;

    for( i = 0; i < mitad_frame; i++ ) {
        if( p[i] > valor_inicial )
            valor_inicial = p[i];
    }
    return(valor_inicial);
}
/*-----*/
/*                      Despliegue de la ventana de datos procesados                      */
/*-----*/
void despliega_ventana(int seg) {
    int ij;

    i = seg;
    setfillstyle(SOLID_FILL, negro);
    bar(120+x_ventana[i],60+300,120+x_ventana[i+1],60);
    for( j = 0; j < mitad_frame; j++ ) {
        if( dato_color[j] != 0 ) {
            setfillstyle(SOLID_FILL, dato_color[j]);
            bar(120+x_ventana[i],60+y_ventana[j],
                120+x_ventana[i+1],60+y_ventana[j+1]);
        }
    }
    setfillstyle(SOLID_FILL, d_blue);
    bar(121+x_ventana[i+1],60+300,122+x_ventana[i+1],60);
}
/*-----*/
/*                      Limpia ventana gráfica (x1,y1,x2,y2)                      */
/*-----*/
void limpia(int fondo, int marco, int x1, int y1, int x2, int y2) {
    setfillstyle(SOLID_FILL, fondo);
    bar(x1,y1,x2,y2);
    setcolor(marco);
    rectangle(x1,y1,x2,y2);
}

```

```

/*-----*/
/*                               */
/*                               */
/*-----*/
void letberos(int x1, int y1, char msg1[20],int color) {

    settextstyle(SMALL_FONT,HORIZ_DIR,10);
    setcolor(color);
    outtextxy(x1,y1,msg1);
}
/*-----*/
/*                               */
/*                               */
/*-----*/
void detecta_boton(void) {

    do {
        GetButtonPress(&status,&count,&x,&y);
    }while(status == 0);
}
/*-----*/
/*                               */
/*                               */
/*-----*/
void suelta_boton(void) {

    do {
        GetButtonRelease(&status,&count,&x,&y);
    }while(status != 0);
}
/*-----*/
/*                               */
/*                               */
/*-----*/
int entra_ventana(int x1,int y1,int x2, int y2) {

    GetMousePosButton(&status,&x,&y);
    if( !(x > x1) && (x < x2) && (y > y1) && (y < y2))
        return(0);
    else
        return(1);
}
/*-----*/
/*                               */
/*                               */
/*-----*/
int sal_ventana(int x1,int y1,int x2,int y2) {

    do {
        GetMousePosButton(&status,&x,&y);
        if( status ) return(status);
    }while( (x > x1) && (x < x2) && (y > y1) && (y < y2) );
    return(0);
}

```

```

}
/*-----*/
/*                               */
/*                               */
/*-----*/
void lee_datos_arch() {
    float total_muestras,total_cols;
    int i;

    if( (fpt = fopen("s1.dat","r")) == NULL )
        printf("\nERROR - No se puede abrir el archivo indicado \n");
    else {
        fgets(dato,80,fpt);
        total_muestras = atof(dato);
        fgets(dato,80,fpt);
        total_cols = atof(dato);
        for( i = 0; i < total_muestras; i++ ) {
            fgets(dato,80,fpt);
            arch_datos[i] = atof(dato);
        }
        fclose(fpt);
    }
}
/*----- Termina programa APMASHO.C -----*/

```

D.2 Programa nodo_2 sistema homogéneo

```

/*-----*/
/* Programa:          APWORHO.C (Nodo_2) Sis. Homogéneo */
/*-----*/
/* Descripción:      Realiza el muestreo y procesamiento de los */
/*                   datos transmitidos por el Nodo_1 y el resul- */
/*                   tado lo envia de nuevo al Nodo_1.          */
/*-----*/

```

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <stddef.h>
#include <string.h>
#include <ctype.h>
#include <conc.h>
#include <time.h>

```

```

/* Declaración de constantes */
#define          NULL          0

```

```
#define TRUE 1
#define FALSE 0
#define WS_SIZE 8192
```

```
void espectrograma();
```

```
int frame_size;
float frec_muestreo;
int mitad_frame;
float poten, modulo;
float dato_real, dato_imag;
int log_frame;
float datos_parametros[2];
float frame_log;
int tecla_pres[1];
int continua[1];
int opcion[1];
int fin[1];
int running;
int memoria_out;
int memoria_chan;
int bandera;
int j;
```

```
FILE *fpt;
char dato[30];
```

```
float arch_datos[8960];
```

```
COMPLEXF *datos_complex;
```

```
float *datos_muestreador;
float *datos_proc;
float *datos_resultado;
float *tiempos;
int t_p[10];
```

```
int cont_arch;
int tick_frame;
float tick;
float periodo_muestreo;
```

```
Process *muestreo_ptr, *procesamiento_ptr;
Channel *chan;
```

```
/*-----*/
/* Proceso de Muestreo */
/*-----*/
```

```
muestreo(Process *p, Channel *chan){
int j;
```

```
p = p;
t_p[0] = Time();
for(j = 0; j < frame_size; j++) {
datos_muestreador[j] = arch_datos[cont_arch];
cont_arch++;
if( cont_arch == 8960 ) cont_arch = 0;
}
periodo_muestreo = 1/(frec_muestreo * 1000.0);
tick = (periodo_muestreo/64.0e-06); /* Procesos de baja prioridad */
tick_frame = (tick * frame_size) + 0.5;
ProcWait(tick_frame);
t_p[1] = Time();
ChanOut(chan, (char *) datos_muestreador, memoria_chan);
```

```
/*-----*/
/* Proceso de cálculo de FFT y PSD */
/*-----*/
```

```
procesamiento(Process *p, Channel *chan) {
int j, i;
```

```
p = p;
ChanIn(chan, (char *) datos_proc, memoria_chan);
t_p[2] = Time();
t_p[3] = 0;
t_p[4] = Time();
for(j = 0; j < frame_size; j++) {
datos_complex[j].real = datos_proc[j];
datos_complex[j].imag = 0.0;
}
ffft(datos_complex, log_frame);
for(j = 0; j < mitad_frame; j++) {
poten = powf(datos_complex[j].real, 2.0) +
powf(datos_complex[j].imag, 2.0);
modulo = sqrtf(poten);
datos_resultado[j] = powf(modulo, 2.0);
}
t_p[5] = Time();
ChanOut(LINK1OUT, (char *) datos_resultado, memoria_out);
```

```
/*-----*/
/* Programa principal */
/*-----*/
```

```
int main() {
```

```
if( _node_number == 2 ) { /* Nodo_2 (TRAM) */
```



```

ChanIn(LINK1IN, (char *)arch_datos, sizeof(arch_datos));
do {
  ChanIn(LINK1IN, (char *)opcion, sizeof(opcion));
  switch( opcion[0] ) {
    case 1: /* parametros */
      ChanIn(LINK1IN, (char *)continua, sizeof(continua));
      bandera = continua[0];
      break;
    case 2: espectrograma();
      ChanIn(LINK1IN, (char *)continua, sizeof(continua));
      bandera = continua[0];
      break;
    case 3: ChanIn(LINK1IN, (char *)continua, sizeof(continua));
      bandera = continua[0];
      break;
  }
  while(bandera);
} /* end de if del _node */
fin[0] = TRUE;
ChanOut(LINK1OUT, (char *)fin, sizeof(fin));
} /* end del main */
/*-----*/
/*                               Función espectrograma                               */
/*-----*/
void espectrograma() {
/* localiza e inicializa memoria para el canal de comunicacion chan */
chan = ChanAlloc();

if( (muestreo_ptr = ProcAlloc(muestreo, WS_SIZE, 1, chan)) == NULL )
  printf("No memory for process 'muestreo'\n");
if( (procesamiento_ptr = ProcAlloc(procesamiento, WS_SIZE, 1, chan)) == NULL )
  printf("No memory for process 'procesamiento'\n");

ChanIn(LINK1IN, (char *)datos_parametros, sizeof(datos_parametros));
frame_size = datos_parametros[0];
frec_muestreo = datos_parametros[1];
mitad_frame = frame_size/2;
frame_log = frame_size;
log_frame = (log10f(frame_log)/log10f(2.0)) + 0.5;
datos_complex = (COMPLEX *) malloc(frame_size*sizeof(COMPLEX));
datos_muestreador = (float *) malloc(frame_size*sizeof(float));
datos_proc = (float *) malloc(frame_size*sizeof(float));
datos_resultado = (float *) malloc(mitad_frame*sizeof(float));
tiempos = (float *) malloc(mitad_frame*sizeof(float));
memoria_out = mitad_frame*sizeof(float);
memoria_chan = frame_size*sizeof(float);

```

```

for( j = 0; j < mitad_frame; j++ )
  tiempos[j] = 0;
for( j = 0; j < 10; j++ )
  t_p[j] = 0;

running = TRUE;
cont_arch = 0;
while(running){
  ChanIn(LINK1IN, (char *)tecla_pres, sizeof(tecla_pres));
  for( j = 0; j < 10; j++ )
    tiempos[j] = t_p[j];
  if( tecla_pres[0] ) {
    running = FALSE;
    ChanOut(LINK1OUT, (char *)tiempos, memoria_out);
  }
  else
    ProcPar(muestreo_ptr, procesamiento_ptr, NULL);

/* libera espacio en memoria */
ProcFree(muestreo_ptr);
ProcFree(procesamiento_ptr);
free(datos_muestreador);
free(datos_proc);
free(datos_resultado);
free(datos_complex);
free(tiempos);
}

/*----- Termina Programa APWORHO.C -----*/

```

D.3 Programa nodo_2 sistema heterogéneo

```

/*-----*/
/* Programa: APWORHE.C (Nodo_2) Sis. Heterogéneo */
/*
/* Descripción: Realiza el muestreo de los datos transmitidos
/* por el Nodo_1. Envía estos datos al Nodo_3
/* (DSP) y los recibe ya procesados, para trans
/* mitarlos al Nodo_1.
/*-----*/

```

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <stddef.h>

```

```

#include <string.h>
#include <ctype.h>
#include <conc.h>
#include <time.h>

/* declaración de constantes */
#define NULL 0
#define TRUE 1
#define FALSE 0
#define WS_SIZE 8192

void espectrograma();
void byte_de_reconocimiento();
void comunicacion();

int frame_size;
float frec_muestreo;
int mitad_frame;
float poten_modulo;
float dato_real, dato_imag;
int log_frame;
float datos_parametros[2];
float frame_log;
int tecla_pres[1];
int continua[1];
int opcion[1];
int fin[1];
int running;
int memoria_out;
int memoria_chan;
int bandera;
int j;

FILE *fpr;
char dato[30];
float arch_datos[8960];

float *datos_muestreador;
float *datos_proc;
float *datos_resultado;
float *tiempos;
int t_p[10];

int cont_arch;
int tick_frame;
float tick;
float periodo_muestreo;

```

```

Process *muestreo_ptr, *procesamiento_ptr;
Channel *chan;

/*-----*/
/*                               Proceso de muestreo                               */
/*-----*/
muestreo(Process *p, Channel *chan) {
    int j;

    p = p;
    t_p[0] = Time();
    for(j = 0; j < frame_size; j++) {
        datos_muestreador[j] = arch_datos[cont_arch];
        cont_arch++;
        if(cont_arch == 8960) cont_arch = 0;
    }
    periodo_muestreo = 1/(frec_muestreo * 1000.0);
    tick = (periodo_muestreo/64.0e-06); /* Procesos de baja prioridad */
    tick_frame = (tick * frame_size) + 0.5;
    ProcWait(tick_frame);
    t_p[1] = Time();
    ChanOut(chan, (char *) datos_muestreador, memoria_chan);
}
/*-----*/
/*                               Envío de los datos muestreados al DSP                               */
/*-----*/
procesamiento(Process *p, Channel *chan) {
    int j, i;

    p = p;
    ChanIn(chan, (char *) datos_proc, memoria_chan);
    t_p[2] = Time();
    t_p[3] = 0;
    t_p[4] = Time();

    comunicacion(): /* Envío y recepción de los datos al DSP */

    t_p[5] = Time();
    ChanOut(LINK1OUT, (char *) datos_resultado, memoria_out);
}
/*-----*/
/*                               Programa principal                               */
/*-----*/
int main() {

    if(_node_number == 2) { /* Nodo_2 o TRAM */
        byte_de_reconocimiento(); /* Byte de reconocimiento para */
        /* la sincronización con el DSP */
    }
}

```

```

ChanIn(LINK1IN, (char *)arch_datos, sizeof(arch_datos));
do {
  ChanIn(LINK1IN, (char *)opcion, sizeof(opcion));
  switch( opcion[0] ) {
    case 1: /* parametros */
      ChanIn(LINK1IN, (char *)continua, sizeof(continua));
      bandera = continua[0];
      break;
    case 2: espectrograma():
      ChanIn(LINK1IN, (char *)continua, sizeof(continua));
      bandera = continua[0];
      break;
    case 3: ChanIn(LINK1IN, (char *)continua, sizeof(continua));
      bandera = continua[0];
      break;
  }
  while(bandera);
} /* end de if del _node */
fin[0] = TRUE;
ChanOut(LINK1OUT, (char *)fin, sizeof(fin));
} /* end del main */
/*-----*/
/*                               */
/*                               */
/*-----*/
void espectrograma() {

/* localiza e inicializa memoria para el canal de comunicacion chan */
chan = ChanAlloc();

if( (muestreo_ptr = ProcAlloc(muestreo, WS_SIZE, 1, chan)) == NULL )
  printf("No memory for process 'muestreo'\n");
if( (procesamiento_ptr = ProcAlloc(procesamiento, WS_SIZE, 1, chan)) == NULL )
  printf("No memory for process 'procesamiento'\n");

ChanIn(LINK1IN, (char *)datos_parametros, sizeof(datos_parametros));
frame_size = datos_parametros[0];
frec_muestreo = datos_parametros[1];
mitad_frame = frame_size/2;
frame_log = frame_size;
log_frame = (log10f(frame_log)/log10f(2.0)) + 0.5;
datos_muestreador = (float *) malloc(frame_size*sizeof(float));
datos_proc = (float *) malloc(frame_size*sizeof(float));
datos_resultado = (float *) malloc(mitad_frame*sizeof(float));
tiempos = (float *) malloc(mitad_frame*sizeof(float));
memoria_out = mitad_frame*sizeof(float);
memoria_chan = frame_size*sizeof(float);

for( j = 0; j < mitad_frame; j++ )

```

```

  tiempos[j] = 0;
for( j = 0; j < 10; j++ )
  t_p[j] = 0;
running = TRUE;
cont_arch = 0;
while( running ) {
  ChanIn(LINK1IN, (char *)tecla_pres, sizeof(tecla_pres));
  for( j = 0; j < 10; j++ )
    tiempos[j] = t_p[j];
  if( tecla_pres[0] ) {
    running = FALSE;
    ChanOut(LINK1OUT, (char *)tiempos, memoria_out);
  }
  else
    ProcPar(muestreo_ptr, procesamiento_ptr, NULL);
}

/* libera espacio en memoria */
ProcFree(muestreo_ptr);
ProcFree(procesamiento_ptr);
free(datos_muestreador);
free(datos_proc);
free(datos_resultado);
free(tiempos);
}
/*-----*/
/* Envio de un byte de reconocimiento al DSP para iniciar la comunicacion con el */
/* Transputer */
/*-----*/
void byte_de_reconocimiento() {
  char byte_out;
  int going_r = 1;

  while(going_r) {
    byte_out = 0x67;
    if( ChanOutTimeFail(LINK0OUT, &byte_out, 1, 8000 + Time()));
    else
      going_r = 0;
  }
}
/*-----*/
/*                               */
/*                               */
/*-----*/
/*                               */
/*                               */
/*-----*/
void comunicacion() {

  int going_c = 1;

  ChanOutInt(LINK0OUT, frame_size); /* Link 0 de salida */

```

```

do {
  if( ChanOutTimeFail(LINK0OUT, (char *) datos_proc, memoria_chan,
    8000 - Time());
  else
    going_c = 0;
}while(going_c);
going_c = 1;
do {
  if( ChanInTimeFail(LINK0IN, (char *) datos_resultado, memoria_out,
    8000 - Time());
  else
    going_c = 0;
}while(going_c);
}
----- Termina programa APWORHE.C -----*

```

D.4 Programa nodo_3 sistema heterogéneo

```

: Programa:      TMS_FFT.ASM
: Descripción:   Realiza la transmisión y recepción de datos a la interfaz del
:               nodo de procesamiento heterogéneo, convierte los datos de
:               un formato IEEE a TMS320C30 y TMS320C30 a IEEE, y
:               efectúa el llamado de los programas que realizan el cálculo
:               de la FFT (TMS_RAD2.ASM) y PSD (TMS_DPS.ASM).
:
:
:

```

```

:               .title "Programa Principal"
:               .global begin
:               .global FFTSIZ
:               .global LOGFFT
:               .global FFT
:               .global COEF_SINE
:               .global COEF_NEG
:               .global DPS
:

```

```

: Tabla con constantes para conversión IEEE a TMS320C30 y TMS320C30 a IEEE
:

```

```

:               .data
ctab .word    0FF800000h
      .word    0FF000000h
      .word    07F000000h
      .word    080000000h

```

```

      .word    081000000h
:
: taba .word    ctab
:
: .sect      "vectors"
:
: reset .word    begin
:
: .text
:
: Dirección base del puerto serial 1 y del timer 1
:
: serial_1 .word    808050h
: timer_1 .word    808030h
:
: Desplazamiento de la dirección base de los registros del puerto serie
:
: gcr .set      00h             ;Registro de control global
: tx_cr .set    02h           ;Registro de control de las funciones del
:                                   ;Puerto serial (FSX/DX/CLKX)
: rx_cr .set    03h           ;Registro de control de las funciones del
:                                   ;Puerto serial (FSR/DR/CLKR)
: timer_cr .set 04h           ;Registro de control del timer R/X
: period .set   06h           ;Registro del periodo del timer R/X
: dxr .set     08h           ;Registro de transmisión de datos
: drr .set     0Ch           ;Registro de recepción de datos
: timer_gc .set 00h           ;Registro de Control global del timer
:
: Asignación de valores para ser puestos en los registros del puerto serial
:
: gcr_word .word    0E800044h;0000 1110 1000 0000 0000 0000 0100 0100
: serial_reset .word 00800044h;0000 0000 1000 0000 0000 0000 0100 0100
:
: tx_cr_word .word    0111h ;0000 0000 0000 0000 0000 0001 0001 0001
: rx_cr_word .word    0111h ;0000 0000 0000 0000 0000 0001 0001 0001
:
: timer_ctl_word .word 000Fh ;0000 0000 0000 0000 0000 0000 0000 1111
:
: timer_prd_word .word 0001h ;0000 0000 0000 0000 0000 0000 0000 0010
:
: timer_gc_word .word 0002h ;0000 0000 0000 0000 0000 0000 0000 0010
:
: timer_gc_word2 .word 0006h ;0000 0000 0000 0000 0000 0000 0000 0110
:
: .word    FFT
: .word    DPS
:
: data_input .usect "data",1024

```

```

FFTSIZ      .usect "data",1
LOGFFT      .usect "data",1
COEF_SINE   .usect "data",1
COEF_NEG    .usect "data",1
:
: Programa principal:
:
begin:      ldi      80h, DP           ;apuntador a los registros internos
            ldi      0, R0
            sti      R0, @8064h      ;cero estados de espera en el bus primario
            or       0800h, ST       ;habilitación de la memoria caché,
            ldi      0, DP           ;carga 0 al reg. apuntador de página
:
: Carga la dirección de inicio del mapa de memoria del puerto serial 1 :
            ld       @serial_1, AR4
:
: Carga la dirección de inicio del mapa de memoria del timer 1:
            ldi      @timer_1, AR2
            ldi      @timer_gc_word2, R0
            sti      R0, *+AR2(timer_gc) ;habilita la señal TCLK1
:
: Carga el valor del registro de control del puerto serie para TX:
            ldi      @tx_cr_word, R0
            sti      R0, *+AR4(tx_cr)
:
: Carga el valor del registro de control del puerto serie para RX:
            ldi      @rx_cr_word, R0
            sti      R0, *+AR4(rx_cr)
:
: Carga el valor del registro de control del timer:
            ldi      @timer_ctl_word, R0
            sti      R0, *+AR4(timer_cr)
:
: Carga el valor del registro del periodo del timer:
            ldi      @timer_prd_word, R0
            sti      R0, *+AR4(period)
:
: Inicializa el puerto serial:
            ldi      @serial_reset, R0
            sti      R0, *+AR4(gcr)

```

```

            or       00h, IOF        ;señal XFI configurada como entrada
:
            nop
            nop
hang:       nop
:
: Desactiva inicialización del puerto serial:
            ldi      @gcr_word, R0
            sti      R0, *+AR4(gcr)
:
            or       0C0h, IE        ;habilitación de los bits de
            ;interrupción EXINT1 y ERINT1
            or       40h, IF        ;bandera de TX lista
:
            ldi      00h, R1        ;valor inicial de R1 para transmisión
            ldi      @data_input, AR0 ;carga la primer palabra
            ;reservada a AR0
:
rx_i_init: tstb   80h, IF          ;prueba el bit 7 de IF
            bz      rx_i_init      ;espera a que el buffer para RX
            ;este lleno
            ldi      0FF7Fh, R2
            and     R2, IF          ;borra bandera de interrupción
:
            ldi      *+AR4(drr), R1 ;carga el contenido del buffer
            ;RX a R1
            and     00FFh, R1
            ldi      0067h, R7
            cmpi   R1, R7          ;compara RX con 0067h
            bnz    rx_i_init      ;espera si RX no es igual a 0067h
:
            ldi      @timer_gc_word, R0
            sti      R0, *+AR2(timer_gc) ;deshabilita la señal TCLK1
:
: Recibe la longitud de FFT
:
long_fft:   ldi      @timer_gc_word2, R0
            sti      R0, *+AR2(timer_gc) ;habilita la señal TCLK1
:
            ldi      00h, R3
:
rx_i_long: tstb   80h, IF          ;prueba bit 7 de IF
            bz      rx_i_long      ;espera a que el buffer de RX
            ;este lleno

```

Apéndice D Listados de Programas D-16

```

ldi 0FF7Fh, R2
and R2, IF
;borra la bandera de interrupción

ldi *+AR4(drr), R1
and 00FFh, R1
ldi R1, R3
; carga el valor del buffer RX a R1
; carga el valor de R1 a R3

rx_2_long:
tstb 80h, IF
bz rx_2_long

ldi 0FF7Fh, R2
and R2, IF

ldi *+AR4(drr), R1
and 00FFh, R1
ash 8, R1
;corrimento de 8 bits a la izq.
;del valor de R1

or R1, R3

rx_3_long:
tstb 80h, IF
bz rx_3_long

ldi 0FF7Fh, R2
and R2, IF

ldi *+AR4(drr), R1
and 00FFh, R1
ash 16, R1
;corrimento de 16 bits a la izq.
;del valor de R1

or R1, R3

rx_4_long:
tstb 80h, IF
bz rx_4_long

ldi 0FF7Fh, R2
and R2, IF

ldi *+AR4(drr), R1
and 00FFh, R1
ash 24, R1
;corrimento de 24 bits a la izq.
;del valor de R1

or R1, R3

sti R3, @FFTSIZ
; carga longitud de FFT en @FFTSIZ

ldi 40h, R6

cmpi R6, R3
bnz long_128
;salta si longitud <> 64

ldi 6h, R0
;longitud = 64

```

```

sti R0, @LOGFFT
;log2 de FFT = 6

ldi 4, R0
; carga desplazamiento (+) a R0

sti R0, @COEF_SINE
;almacena desplazamiento (+) para
;accesar la tabla de SIN y COS

ldi -4, R0
; carga desplazamiento (-) a R0

sti R0, @COEF_NEG
;almacena desplazamiento (-) para
;accesar la tabla de SIN y COS

b term_long
;termina de recibir longitud de FFT

long_128:
ldi 80h, R6
cmpi R6, R3
bnz long_256
;salta si longitud <> 128

ldi 7h, R0
;longitud = 128

sti R0, @LOGFFT
;log2 de FFT = 7

ldi 3, R0
sti R0, @COEF_SINE
ldi -3, R0
sti R0, @COEF_NEG
b term_long

long_256:
ldi 0100h, R6
cmpi R6, R3
bnz long_512
;salta si longitud <> 256

ldi 8h, R0
;longitud = 256

sti R0, @LOGFFT
;log2 de FFT = 8

ldi 2, R0
sti R0, @COEF_SINE
ldi -2, R0
sti R0, @COEF_NEG
b term_long

long_512:
ldi 0200h, R6
cmpi R6, R3
bnz long_1024
;salta si longitud <> 512

ldi 9h, R0
;longitud = 512

sti R0, @LOGFFT
;log2 de FFT = 9

ldi 1, R0
sti R0, @COEF_SINE
ldi -1, R0
sti R0, @COEF_NEG
b term_long

long_1024:
ldi 0400h, R6
cmpi R6, R3
bnz term_long
;longitud = 1024

ldi 0Ah, R0
;log2 de FFT = 10

sti R0, @LOGFFT
ldi 0, R0

```

```

sti R0, @COEF_SINE
ldi 0, R0
sti R0, @COEF_NEG

term_long:
ldi @timer_gc_word, R0
sti R0, *+AR2(timer_gc) ;deshabilita TCLK1 y termina
;de recibir longitud

; recibe datos
main_loop:
nop
ldi 0, IR0

ldi @timer_gc_word2, R0
sti R0, *+AR2(timer_gc) ;activa la señal TCLK1

inicia_rx:
ldi 00h, R3

rx_1_wait:
tstb 80h, IF ;prueba bit 7 de IF
bz rx_1_wait ;espera a que el buffer RX
;este lleno

ldi 0FF7Fh, R2
and R2, IF ;borra la bandera de interrupción

ldi *+AR4(drr), R1 ;carga el valor del buffer RX en R1
and 00FFh, R1
ldi R1, R3 ;carga R1 en R3

rx_2_wait:
tstb 80h, IF
bz rx_2_wait

ldi 0FF7Fh, R2
and R2, IF

ldi *+AR4(drr), R1
and 00FFh, R1
ash 8, R1 ;corrimiento de 8 bits a la izq. de R1
or R1, R3 ;carga R1 en R3

rx_3_wait:
tstb 80h, IF
bz rx_3_wait

ldi 0FF7Fh, R2
and R2, IF

ldi *+AR4(drr), R1
and 00FFh, R1

ash 16, R1 ;corrimiento de 16 bits a la izq. de R1
or R1, R3

rx_4_wait:
tstb 80h, IF
bz rx_4_wait

ldi 0FF7Fh, R2
and R2, IF

ldi *+AR4(drr), R1
and 00FFh, R1
ash 24, R1 ;corrimiento de 24 bits a la izq. de R1
or R1, R3
sti R3, *+AR0(IR0) ;carga R3 en AR0
cmpi @FFTSIZ, IR0 ;compara FFTSIZ con IR0
;si no es el total de datos recibe
;más datos

bnz inicia_rx

ldi @timer_gc_word, R0
sti R0, *+AR2(timer_gc) ;deshabilita TCLK1 y termina
;de recibir los datos

;
; Conversión del formato en punto flotante de IEEE a TMS320C30
;
ldi @data_input, AR0 ;carga la primer palabra reservada
ldi @FFTSIZ, RC ;RC <= N
subi 1, RC ;RC <= N-1
ldi @taba, AR3 ;AR3 -> tabla de constantes

;
; lazo de repetición para la conversión ieee -> 'C30
;
rptb loop4 ;repite N veces
and *AR0, *AR3, R0 ;reemplaza la parte fraccional con 0
addi *AR0, R0 ;corrimiento del signo y del
;exponente insertando 0's
ldiz *+AR3(1), R0 ;si todo es cero, carga 0.0
ldi *AR0, R1 ;prueba del número original
bged loop4 ;si >= 0, almacena número
subi *+AR3(2), R0 ;elimina exponente via (127)
push R0 ;guarda R0 como un entero
popf R0 ;recupera R0 como punto flotante

negf R0 ;número negativo
stf R0, *AR0+@ ;guarda número con formato 'C30

loop4:

```

```

; incrementa ARO
;
; Cálculo de la FFT Real Radix-2
;
; carga la primer palabra reservada
; llama al programa que calcula FFT
; los datos transformados son
; cargados a la dirección apuntada
; por @data_input
ldi @data_input, ARO
call FFT
sti ARO, @data_input
;
; Calcula la PSD
;
; carga la primer palabra reservada
; llama al programa que realiza la PSD
; después de realizar la PSD los
; datos son cargados a la dirección
; apuntada por @data_input
ldi @data_input, ARO
call DPS
sti ARO, @data_input
;
; Conversión del formato en punto flotante del TMS320C30 a IEEE
;
; carga la primer palabra reservada
; RC <= N
; RC = N/2
; RC <= N-1
; AR3 -> tabla de constantes
ldi @data_input, ARO
ldi @FFTSIZ, RC
lsh -1, RC
subi 1, RC
ldi @tabla, AR3
;
; Lazo de repetición para la conversión C30 -> IEEE
; repite el lazo N veces
; prueba valor abs(número)
; si == cero, carga como si fuera 0.0
; corrimiento del bit (eliminación)
; guarda como punto flotante
; prueba del número original
; si >= 0, almacena el número
; recupera como un entero
; suma el exponente via (127)
; corrige para el bit de signo
rptb loop5
absf *.ARO, R0
ldfz *+AR3(4), R0
lsh 1, R0
pushf R0
ldf *.ARO, R1
bged loop5
pop R0
addi *+AR3(2), R0
lsh -1, R0
or *+AR3(3), R0
; número ieee negativo
loop5: sti R0, *AR0++
; guarda número ieee, incr. ARO

```

```

; Transmisión del dato
;
; carga la primer palabra reservada
;
; carga la primer palabra reservada
; envía R1 a través de TX1
ldi @data_input, ARO
ldi 0, IR0
inicia_tx: nop
tx_1_loop: ldi *+AR0(IR0), R1
sti R1, *+AR4(dxr)
; espera a que el buffer TX este vacio
tx_1_wait: tstb 40h, IF
bz tx_1_wait
; limpia la bandera de interrupción
ldi 0FFBFh, R2
and R2, IF
; espera hasta que IACK = 1
iack_11_wait: tstb 80h, IOF
bz iack_11_wait
; habilita la señal TCLK1
ldi @timer_gc_word2, R0
sti R0, *+AR2(timer_gc)
; espera hasta que IACK = 0
iack_12_wait: tstb 80h, IOF
bnz iack_12_wait
; deshabilita la señal TCLK1
ldi @timer_gc_word, R0
sti R0, *+AR2(timer_gc)
; corrimiento 8 bits a la derecha de R1
; envía R1 a través de TX1
ldi *+AR0(IR0), R1
ash -8, R1
sti R1, *+AR4(dxr)
; espera a que el buffer TX este vacio
tx_2_wait: tstb 40h, IF
bz tx_2_wait
;
ldi 0FFBFh, R2
and R2, IF
;
iack_21_wait: tstb 80h, IOF
bz iack_21_wait
;
ldi @timer_gc_word2, R0
sti R0, *+AR2(timer_gc)
;
iack_22_wait: tstb 80h, IOF

```



```

.WORD FFT ;dirección de inicio del programa FFT
.SPACE 100 ;reserva 100 palabras para vectores,etc.

.word FFTSIZ
.word LOGFFT
.word COEF_SINE
.word COEF_NEG

SINTAB .WORD SINE1024
INPUT .WORD INP
OUTPUT WORD OUTP

FFT: push AR0 ;salva el valor de los registros
      push AR1 ;auxiliares en el stack
      push AR2
      push AR3
      push AR4
      push AR5
      push AR6
      push AR7

      sti AR0, @INPUT ;los datos de entrada son almacenados
                        ;en la dirección apuntada por @INPUT

      LDP FFTSIZ ;carga del apuntador de página de datos

      ; Realiza el bit-reserved al inicio

      LDI @FFTSIZ,RC ;RC=N
      SUBI 1,RC ;Decremento de RC una unidad
      LDI @FFTSIZ,IR0
      LSH -1,IR0 ;IR0=N/2
      LDI @INPUT,AR0
      LDI @INPUT,ARI

      RPTB BITRV
      CMPI ARI,AR0 ;intercambio de localidades
      BGE CONT ;Si AR0<ARI
      LDF *AR0,R0
      LDF *ARI,R1
      STF R0,*ARI
      STF R1,*AR0
      CONT NOP *AR0++
      BITRV NOP *ARI++(IR0)B

```

```

; Mariposas logitud=2
;
      LDI @INPUT,AR0 ;AR0 apunta a X(I)
      LDI IR0,RC ;repite N/2 veces
      SUBI 1,RC ;decremento de RC una unidad

      RPTB BLK1
      ADDF *+AR0,*AR0++,R0 ;R0=X(I)+X(I+1)
      SUBF *AR0,*-AR0,R1 ;R1=X(I)-X(I+1)
      BLK1 STF R0,*-AR0 ;X(I)=X(I)+X(I+1)
      || STF R1,*AR0++ ;X(I+1)=X(I)-X(I+1)

; Cálculo de otras mariposas

      LDI @INPUT,AR0 ;AR0 apunta a X(I)
      LDI 2,IR0 ;IR0=2=N2
      LDI @FFTSIZ,RC
      LSH -2,RC ;repite N/4 veces
      SUBI 1,RC ;decremento de RC una unidad

      RPTB BLK2
      ADDF *+AR0(IR0),*AR0++(IR0),R0 ;R0=X(I)+X(I+2)
      SUBF *AR0,*-AR0(IR0),R1 ;R1=X(I)-X(I+2)
      NEGF *+AR0,R0 ;R0=-X(I+3)
      || STF R0,*-AR0(IR0) ;X(I)=X(I)+X(I+2)
      BLK2 STF R1,*AR0++(IR0) ;X(I+2)=X(I)-X(I+2)
      || STF R0,*+AR0 ;X(I+3)=-X(I+3)

; Lazo de repetición principal (etapas de FFT)

      LDI @FFTSIZ,IR0
      LSH -2,IR0 ;IR0=índice para E
      LDI 3,R5 ;R5 mantiene el número de la etapa
                        ;actual
                        ;R4=N4
                        ;R3=N2
                        ;E=E/2
                        ;N4=2*N4
                        ;N2=2*N2

      LDI 1,R4
      LDI 2,R3
      LOOP LSH -1,IR0
           LSH 1,R4
           LSH 1,R3

      ldi @COEF_SINE, R6
      lsh R6, IR0

; Lazo repetición interno
;

```

```

INLOP    LDI      @INPUT,AR5          ;AR5 apunta a X(I)
          LDI      IR0,AR0
          ADDI     @SINTAB,AR0       ;AR0 apunta a la tabla SIN/COS
          LDI      R4,IR1           ;IR1=N4

          LDI      AR5,AR1
          ADDI     1,AR1             ;AR1 apunta a X(I1)=X(I+J)
          LDI      AR1,AR3
          ADDI     R3,AR3           ;AR3 apunta a X(I3)=X(I+J+N2)
          LDI      AR3,AR2
          SUBI     2,AR2            ;AR2 apunta a X(I2)=X(I-J-N2)
          ADDI     R3,AR2,AR4       ;AR4 apunta a X(I4)=X(I-J+N1)

          LDF      *AR5++(IR1),R0   ;R0=X(I)
          ADDF     *+AR5(IR1),R0,R1 ;R1=X(I)+X(I+N2)
          SUBF     R0,*+AR5(IR1),R0 ;R0=-X(I)+X(I+N2)
          STF      R1,*-AR5(IR1)    ;X(I)-X(I)+X(I+N2)
          NEGF     R0
          NEGF     *+AR5(IR1),R1    ;R0=X(I)-X(I+N2)
          STF      R0,*AR5         ;R1=-X(I+N4+N2)
          STF      R1,*AR5         ;X(I+N2)=X(I)-X(I+N2)
          ;X(I+N4+N2)=-X(I+N4+N2)

          ; Lazo de repetición más interno

          LDI      @FFTSIZ,IR1
          LSH      -2,IR1           ;IR1=separación de las tablas SIN/COS
          ;TBL5

          LDI      R4,RC
          SUBI     2,RC             ;repite N4-1 veces

          ldi      @COEF_SINE, R6
          lsh      R6, IR1

          RPTB    BLK3
          MPYF     *AR3,*+AR0(IR1),R0 ;R0=X(I3)*COS
          MPYF     *AR4,*AR0,R1       ;R1=X(I4)*SIN
          MPYF     *AR4,*+AR0(IR1),R1 ;R1=X(I4)*COS
          ADDF     R0,R1,R2           ;R2=X(I3)*COS+X(I4)*SIN
          MPYF     *AR3,*AR0+-(IR0),R0 ;R0=X(I3)*SIN
          SUBF     R0,R1,R0           ;R0=-X(I3)*SIN+X(I4)*COS !!!
          SUBF     *AR2,R0,R1        ;R1=-X(I2)+R0 !!!
          ADDF     *AR2,R0,R1        ;R1=X(I2)+R0 !!!
          STF      R1,*AR3++        ;X(I3)=X(I2)+R0 !!!
          ADDF     *AR1,R2,R1        ;R1=X(I1)+R2

          STF      R1,*AR4-
          SUBF     R2,*AR1,R1
          STF      R1,*AR1++
          BLK3   STF      R1,*AR2-

          ldi      @COEF_NEG, R6
          lsh      R6, IR1

          SUBI     @INPUT,AR5
          ADDI     R4,AR5             ;AR5=I+NI
          CMPI     @FFTSIZ,AR5
          BLTD
          ADDI     @INPUT,AR5
          NOP
          NOP

          ldi      @COEF_NEG, R6
          lsh      R6, IR0

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

          pop      AR7
          pop      AR6               ;recupera los registros auxiliares
          pop      AR5               ;del stack
          pop      AR4
          pop      AR3
          pop      AR2
          pop      AR1
          pop      AR0

          ldi      @OUTPUT, AR0     ;AR0 tiene los datos de salida

          rets                    ;regresa al programa que donde fue
                                   ;llamado
          end                      ;termina programa de cálculo de FFT

```

D.4.2 Cálculo de la Densidad de Potencia Espectral

```

Programa:      TMS_DPS.ASM
Descripción:   Realiza el cálculo de la Densidad de Potencia Espectral (PSD)

                .GLOBAL DPS                :punto de entrada para la ejecución
                .GLOBAL FFTSIZ             :longitud de FFT
                .GLOBAL SQR              :rutina para el cálculo de raíz cuadrada

INP_DPS0       USECT  "DPS".1024           :memoria con datos de entrada
INP_DPS1       USECT  "DPS"               :memoria con datos de entrada

                .TEXT

: Inicialización
                .WORD  DPS                 :dirección de inicio del programa DPS
                .WORD  SQR                 :dirección de inicio de la rutina SQR

                .SPACE 100                 :reserva 100 palabras para vectores.
                :ETC.

                .WORD  FFTSIZ              :dirección de N

INPUT0         WORD  INP_DPS0              :localización de los datos de entrada
INPUT1         WORD  INP_DPS1              :localización de los datos de entrada

DPS:           STI    AR0, @INPUT0          :carga los datos de entrada (AR0)
                :en la dirección apuntada por @INPUT0
                LD    @INPUT1, AR1         :la dirección @INPUT1 es cargada en
                :AR1

                LDP   FFTSIZ               :carga la dirección de FFTSIZ a la
                :página de datos
                LDI   @FFTSIZ, R6          :R6=N
                LSH   -1, R6               :R6=N/2

                LDI   0, IR0               :IR0=0
                LDI   0, IR1               :IR1=0
    
```

```

: Cálculo de la potencia de 2 de la parte real
:
MUL_REAL:
    LDF    *+AR0(IR0), R0                 :carga el dato a R0
    MPYF   R0, R0                          :eleva al cuadrado el valor de R0
    STF    R0, *+AR0(IR0)                 :almacena R0 en AR0
    ADDI   1, IR0                           :incrementa IR0
    CMPI   R6, IR0                          :
    BNZ   MUL_REAL                          :if IR0 < N/2 salta a la etiqueta
                                           :MUL_REAL
                                           :R0=0
    LDF    0, R0                            :guarda R0 en AR1
    STF    R0, *+AR1(IR1)                 :
    ADDI   1, IR1                           :IR1=IR1+1
    LDI   @FFTSIZ, IR0                      :IR0=N
    SUBI   1, IR0                           :IR0=N-1

: Cálculo de la potencia de 2 de la parte imaginaria
:
MUL_IMAG:
    LDF    *+AR0(IR0), R0                 :carga AR0 a R0
    MPYF   R0, R0                          :eleva al cuadrado el valor de R0
    STF    R0, *+AR1(IR1)                 :almacena R0 en AR0
    ADDI   1, IR1                           :incrementa IR1
    SUBI   1, IR0                           :decrementa IR0
    CMPI   R6, IR1                          :
    BNZ   MUL_IMAG                          :si IR1 < N/2 salta a la etiqueta
                                           :MUL_IMAG

: Suma de la parte real y la parte imaginaria, y cálculo de la raíz cuadrada
:
    LDI   @INPUT0, AR0                      :recupera valores en AR0
    LDI   @INPUT1, AR1                      :recupera valores en AR1
    LDI   0, IR0

OBTEN_MOD:
    ADDF   *+AR0(IR0), *+AR1(IR0), R0      :
    CALL   SQR                               :llamado del programa que realiza
                                           :el cálculo de la SQRT
    MPYF   R0, R0                            :calcula el cuadrado de R0
    STF    R0, *+AR0(IR0)                 :almacena dato en AR0
    ADDI   1, IR0                           :incrementa IR0
    CMPI   R6, IR0                          :
    BNZ   OBTEN_MOD                          :si IR0 < N/2 obtén el siguiente
                                           :módulo

    LDI   @INPUT0, AR0                      :datos de salida en AR0
    
```

```
rets          ;regresa al programa que llamo el
              ;cálculo de la PSD
.end          ;fin del programa de cálculo de PSD
```

D.4.3 Cálculo de la SQRT (raíz cuadrada)

```
; Programa:      TMS SQRT.ASM
; Descripción:   Cálculo de la raíz cuadrada SQRT, R0 <= SQRT(R0).
;               Aproximación: 8 dígitos decimales
;               Restricción de entrada: R0 >= 0.0
;               Registros para entrada: R0
;               Registros para salida: R0
;               Registros alterados: R0-4
;               Registros usados y restaurados: DP, SP
;
; Nombre de programas externos
;
; .GLOBL SQRT          ;punto de entrada para ejecución
;
; Constantes internas
;
; .DATA
CNST1 .SET 0.5
CNST2 .SET 1.5
CNST3 .FLOAT 1.103553391 ;ajuste a 1.0
CNST4 .FLOAT 0.780339086 ;ajuste de SQRT(1.2)
SMSK .WORD 0FF7FFFFFH
;
; .TEXT
; Inicio del programa SQRT
SQRT
;
; LDF R0,R3          ;prueba y salva V
; RETSLE            ;retorna si V = 0
;
; Obtiene aproximación para 1/V. Para V = (1-M)*2**E
; y 0 <= M < 1, para E par: X[0] = (1-M/2)*2**E/2
```

```
; y para E impar: X[0] = SQRT(1/2)*(1-M/2)*2**E/2
```

```
PUSH DP          ;salva DP
LDP @SMSK        ;carga apuntador de página de datos
PUSHF R0         ;salva V como punto flotante
                ;V = (1+M)*2**E
                ;R2 <= V como entero
                ;R2 <= complementa todo pero con signo
                ;R1 <= (1-M/2)*2**E
                ;R4 <= R1
                ;R1 <= R1 exponente eliminado
                ;R2 <= R2 con exponente -E/2
                ;salva R2 como entero
                ;R2 <= punto flotante
                ;R1 <= (1-M/2)*2**E/2
                ;R2 <= 1.1... para E impar
                ;prueba bit menos significativo de E
                ;(como signo)
                ;si E es par R2 <= 0.78...
                ;R1 <= cálculo corregido
                ;recupera DP
```

```
; Genera V/2 (usa MPYF).
```

```
MPYF CNST1,R0    ;R0 <= V/2 truncado
RND R0           ;R0 <= redondeo de V/2
```

```
; Iteración de NEWTON para Y(X) = X - V**2 = 0 ...
```

```
MPYF R1,R1,R2    ;R2 <= X[0]**2
MPYF R0,R2       ;R2 <= (V/2) * X[0]**2
SUBRF CNST2,R2   ;R2 <= 1.5 - (V/2) * X[0]**2
MPYF R2,R1       ;R1 <= X[1] = X[0] * (1.5 -
                ;(V/2)*X[0]**2)
;
MPYF R1,R1,R2    ;R2 <= X[1]**2
MPYF R0,R2       ;R2 <= (V/2) * X[1]**2
SUBRF CNST2,R2   ;R2 <= 1.5 - (V/2) * X[1]**2
MPYF R2,R1       ;R1 <= X[2] = X[1] * (1.5 -
                ;(V/2)*X[1]**2)
;
MPYF R1,R1,R2    ;R2 <= X[2]**2
MPYF R0,R2       ;R2 <= (V/2) * X[2]**2
SUBRF CNST2,R2   ;R2 <= 1.5 - (V/2) * X[2]**2
MPYF R2,R1       ;R1 <= X[3] = X[2] * (1.5 -
                ;(V/2)*X[2]**2)
;
RND R1           ;redondeo antes de *
```

MPYF R1,R1,R2
RND R2
MPYF R0,R2
SUBRF CNST2,R2
RND R2
MPYF R2,R1

: Invierte resultado final y regresa

POP R2
BUD R2
RND R3
RND R1
MPYF R1,R3,R0

:R2 <= X[3]**2
:redondeo antes de *
:R2 <= (V/2) * X[3]**2
:R2 <= 1.5 - (V/2) * X[3]**2
:redondeo antes de *
:R1 <= X[4] = X[3] * (1.5 -
:(V/2)*X[3]**2)

:R2 = recupera dirección de regreso
:regreso (retrasado)
:redondea antes de *
:redondeo antes de *
:R0 = SQRT(V) = V*SQRT(1/V)

D.4.4 Tabla de valores para el cálculo de FFT

: Archivo: SINE1024.ASM

: Descripción: Este archivo es encadenado con el código fuente del
: programa que realiza la FFT para números reales con
: el método Radix-2. La tabla permite calcular una
: transformada rápida de Fourier de 1024 puntos.
:

: global SINE1024

.data

SINE1024	.float 0.183040	.float 0.377007	.float 0.555570
.float 0.000000	.float 0.189069	.float 0.382683	.float 0.560662
.float 0.006136	.float 0.195090	.float 0.388345	.float 0.565732
.float 0.012272	.float 0.201105	.float 0.393992	.float 0.570781
.float 0.018407	.float 0.207111	.float 0.399624	.float 0.575808
.float 0.024541	.float 0.213110	.float 0.405241	.float 0.580814
.float 0.030675	.float 0.219101	.float 0.410843	.float 0.585798
.float 0.036807	.float 0.225084	.float 0.416430	.float 0.590760
.float 0.042938	.float 0.231058	.float 0.422000	.float 0.595699
.float 0.049068	.float 0.237024	.float 0.427555	.float 0.600616
.float 0.055195	.float 0.242980	.float 0.433094	.float 0.605511
.float 0.061321	.float 0.248928	.float 0.438616	.float 0.610383
.float 0.067444	.float 0.254866	.float 0.444122	.float 0.615232
.float 0.073565	.float 0.260794	.float 0.449611	.float 0.620057
.float 0.079682	.float 0.266713	.float 0.455084	.float 0.624860
.float 0.085797	.float 0.272621	.float 0.460539	.float 0.629638
.float 0.091909	.float 0.278520	.float 0.465977	.float 0.634393
.float 0.098017	.float 0.284408	.float 0.471397	.float 0.639124
.float 0.104122	.float 0.290285	.float 0.476799	.float 0.643832
.float 0.110222	.float 0.296151	.float 0.482184	.float 0.648514
.float 0.116319	.float 0.302006	.float 0.487550	.float 0.653173
.float 0.122411	.float 0.307850	.float 0.492898	.float 0.657807
.float 0.128498	.float 0.313682	.float 0.498228	.float 0.662416
.float 0.134581	.float 0.319502	.float 0.503538	.float 0.667000
.float 0.140658	.float 0.325310	.float 0.508830	.float 0.671559
.float 0.146730	.float 0.331106	.float 0.514103	.float 0.676093
.float 0.152797	.float 0.336890	.float 0.519356	.float 0.680601
.float 0.158858	.float 0.342661	.float 0.524590	.float 0.685084
.float 0.164913	.float 0.348419	.float 0.529804	.float 0.689541
.float 0.170962	.float 0.354164	.float 0.534998	.float 0.693971
.float 0.177004	.float 0.359895	.float 0.540171	.float 0.698376
	.float 0.365613	.float 0.545325	.float 0.702755
	.float 0.371317	.float 0.550458	.float 0.707107

.float 0.711432
.float 0.715731
.float 0.720003
.float 0.724247
.float 0.728464
.float 0.732654
.float 0.736817
.float 0.740951
.float 0.745058
.float 0.749136
.float 0.753187
.float 0.757209
.float 0.761202
.float 0.765167
.float 0.769103
.float 0.773010
.float 0.776888
.float 0.780737
.float 0.784557
.float 0.788346
.float 0.792107
.float 0.795837
.float 0.799537
.float 0.803208
.float 0.806848
.float 0.810457
.float 0.814036
.float 0.817585
.float 0.821102
.float 0.824589
.float 0.828045
.float 0.831470
.float 0.834863
.float 0.838225
.float 0.841555
.float 0.844854
.float 0.848120
.float 0.851355
.float 0.854558
.float 0.857729
.float 0.860867
.float 0.863973
.float 0.867046
.float 0.870087
.float 0.873095
.float 0.876070
.float 0.879012
.float 0.881921

.float 0.884797
.float 0.887640
.float 0.890449
.float 0.893224
.float 0.895966
.float 0.898674
.float 0.901349
.float 0.903989
.float 0.906596
.float 0.909168
.float 0.911706
.float 0.914210
.float 0.916679
.float 0.919114
.float 0.921514
.float 0.923880
.float 0.926210
.float 0.928506
.float 0.930767
.float 0.932993
.float 0.935184
.float 0.937339
.float 0.939459
.float 0.941544
.float 0.943593
.float 0.945607
.float 0.947586
.float 0.949528
.float 0.951435
.float 0.953306
.float 0.955141
.float 0.956940
.float 0.958703
.float 0.960431
.float 0.962121
.float 0.963776
.float 0.965394
.float 0.966976
.float 0.968522
.float 0.970031
.float 0.971504
.float 0.972940
.float 0.974339
.float 0.975702
.float 0.977028
.float 0.978317
.float 0.979570
.float 0.980785

.float 0.981964
.float 0.983105
.float 0.984210
.float 0.985278
.float 0.986308
.float 0.987301
.float 0.988258
.float 0.989177
.float 0.990058
.float 0.990903
.float 0.991710
.float 0.992480
.float 0.993212
.float 0.993907
.float 0.994565
.float 0.995185
.float 0.995767
.float 0.996313
.float 0.996820
.float 0.997290
.float 0.997723
.float 0.998118
.float 0.998476
.float 0.998795
.float 0.999078
.float 0.999322
.float 0.999529
.float 0.999699
.float 0.999831
.float 0.999925
.float 0.999981
COSINE
.float 1.000000
.float 0.999981
.float 0.999925
.float 0.999831
.float 0.999699
.float 0.999529
.float 0.999322
.float 0.999078
.float 0.998795
.float 0.998476
.float 0.998118
.float 0.997723
.float 0.997290
.float 0.996820
.float 0.996313
.float 0.995767

.float 0.995185
.float 0.994565
.float 0.993907
.float 0.993212
.float 0.992480
.float 0.991710
.float 0.990903
.float 0.990058
.float 0.989177
.float 0.988258
.float 0.987301
.float 0.986308
.float 0.985278
.float 0.984210
.float 0.983105
.float 0.981964
.float 0.980785
.float 0.979570
.float 0.978317
.float 0.977028
.float 0.975702
.float 0.974339
.float 0.972940
.float 0.971504
.float 0.970031
.float 0.968522
.float 0.966976
.float 0.965394
.float 0.963776
.float 0.962121
.float 0.960431
.float 0.958703
.float 0.956940
.float 0.955141
.float 0.953306
.float 0.951435
.float 0.949528
.float 0.947586
.float 0.945607
.float 0.943593
.float 0.941544
.float 0.939459
.float 0.937339
.float 0.935184
.float 0.932993
.float 0.930767
.float 0.928506
.float 0.926210

.float 0.923880
.float 0.921514
.float 0.919114
.float 0.916679
.float 0.914210
.float 0.911706
.float 0.909168
.float 0.906596
.float 0.903989
.float 0.901349
.float 0.898674
.float 0.895966
.float 0.893224
.float 0.890449
.float 0.887640
.float 0.884797
.float 0.881921
.float 0.879012
.float 0.876070
.float 0.873095
.float 0.870087
.float 0.867046
.float 0.863973
.float 0.860867
.float 0.857729
.float 0.854558
.float 0.851355
.float 0.848120
.float 0.844854
.float 0.841555
.float 0.838225
.float 0.834863
.float 0.831470
.float 0.828045
.float 0.824589
.float 0.821102
.float 0.817585
.float 0.814036
.float 0.810457
.float 0.806848
.float 0.803208
.float 0.799537
.float 0.795837
.float 0.792107
.float 0.788346
.float 0.784557
.float 0.780737
.float 0.776888

.float 0.773010
.float 0.769103
.float 0.765167
.float 0.761202
.float 0.757209
.float 0.753187
.float 0.749136
.float 0.745058
.float 0.740951
.float 0.736817
.float 0.732654
.float 0.728464
.float 0.724247
.float 0.720003
.float 0.715731
.float 0.711432
.float 0.707107
.float 0.702755
.float 0.698376
.float 0.693971
.float 0.689541
.float 0.685084
.float 0.680601
.float 0.676093
.float 0.671559
.float 0.667000
.float 0.662416
.float 0.657807
.float 0.653173
.float 0.648514
.float 0.643832
.float 0.639124
.float 0.634393
.float 0.629638
.float 0.624860
.float 0.620057
.float 0.615232
.float 0.610383
.float 0.605511
.float 0.600616
.float 0.595699
.float 0.590760
.float 0.585798
.float 0.580814
.float 0.575808
.float 0.570781
.float 0.565732
.float 0.560662

.float 0.555570
.float 0.550458
.float 0.545325
.float 0.540171
.float 0.534998
.float 0.529804
.float 0.524590
.float 0.519356
.float 0.514103
.float 0.508830
.float 0.503538
.float 0.498228
.float 0.492898
.float 0.487550
.float 0.482184
.float 0.476799
.float 0.471397
.float 0.465977
.float 0.460539
.float 0.455084
.float 0.449611
.float 0.444122
.float 0.438616
.float 0.433094
.float 0.427555
.float 0.422000
.float 0.416430
.float 0.410843
.float 0.405241
.float 0.399624
.float 0.393992
.float 0.388345
.float 0.382683
.float 0.377007
.float 0.371317
.float 0.365613
.float 0.359895
.float 0.354164
.float 0.348419
.float 0.342661
.float 0.336890
.float 0.331106
.float 0.325310
.float 0.319502
.float 0.313682
.float 0.307850
.float 0.302006
.float 0.296151

.float 0.290285
.float 0.284408
.float 0.278520
.float 0.272621
.float 0.266713
.float 0.260794
.float 0.254866
.float 0.248928
.float 0.242980
.float 0.237024
.float 0.231058
.float 0.225084
.float 0.219101
.float 0.213110
.float 0.207111
.float 0.201105
.float 0.195090
.float 0.189069
.float 0.183040
.float 0.177004
.float 0.170962
.float 0.164913
.float 0.158858
.float 0.152797
.float 0.146730
.float 0.140658
.float 0.134581
.float 0.128498
.float 0.122411
.float 0.116319
.float 0.110222
.float 0.104122
.float 0.098017
.float 0.091909
.float 0.085797
.float 0.079682
.float 0.073565
.float 0.067444
.float 0.061321
.float 0.055195
.float 0.049068
.float 0.042938
.float 0.036807
.float 0.030675
.float 0.024541
.float 0.018407
.float 0.012272
.float 0.006136

.float 0.000000
.float -0.006136
.float -0.012272
.float -0.018407
.float -0.024541
.float -0.030675
.float -0.036807
.float -0.042938
.float -0.049068
.float -0.055195
.float -0.061321
.float -0.067444
.float -0.073565
.float -0.079682
.float -0.085797
.float -0.091909
.float -0.098017
.float -0.104122
.float -0.110222
.float -0.116319
.float -0.122411
.float -0.128498
.float -0.134581
.float -0.140658
.float -0.146730
.float -0.152797
.float -0.158858
.float -0.164913
.float -0.170962
.float -0.177004
.float -0.183040
.float -0.189069
.float -0.195090
.float -0.201105
.float -0.207111
.float -0.213110
.float -0.219101
.float -0.225084
.float -0.231058
.float -0.237024
.float -0.242980
.float -0.248928
.float -0.254866
.float -0.260794
.float -0.266713
.float -0.272621
.float -0.278520
.float -0.284408

.float -0.290285
.float -0.296151
.float -0.302006
.float -0.307850
.float -0.313682
.float -0.319502
.float -0.325310
.float -0.331106
.float -0.336890
.float -0.342661
.float -0.348419
.float -0.354164
.float -0.359895
.float -0.365613
.float -0.371317
.float -0.377007
.float -0.382683
.float -0.388345
.float -0.393992
.float -0.399624
.float -0.405241
.float -0.410843
.float -0.416430
.float -0.422000
.float -0.427555
.float -0.433094
.float -0.438616
.float -0.444122
.float -0.449611
.float -0.455084
.float -0.460539
.float -0.465977
.float -0.471397
.float -0.476799
.float -0.482184
.float -0.487550
.float -0.492898
.float -0.498228
.float -0.503538
.float -0.508830
.float -0.514103
.float -0.519356
.float -0.524590
.float -0.529804
.float -0.534998
.float -0.540171
.float -0.545325
.float -0.550458

.float -0.555570
.float -0.560662
.float -0.565732
.float -0.570781
.float -0.575808
.float -0.580814
.float -0.585798
.float -0.590760
.float -0.595699
.float -0.600616
.float -0.605511
.float -0.610383
.float -0.615232
.float -0.620057
.float -0.624860
.float -0.629638
.float -0.634393
.float -0.639124
.float -0.643832
.float -0.648514
.float -0.653173
.float -0.657807
.float -0.662416
.float -0.667000
.float -0.671559
.float -0.676093
.float -0.680601
.float -0.685084
.float -0.689541
.float -0.693971
.float -0.698376
.float -0.702755
.float -0.707107
.float -0.711432
.float -0.715731
.float -0.720003
.float -0.724247
.float -0.728464
.float -0.732654
.float -0.736817
.float -0.740951
.float -0.745058
.float -0.749136
.float -0.753187
.float -0.757209
.float -0.761202
.float -0.765167
.float -0.769103

.float -0.773010
.float -0.776888
.float -0.780737
.float -0.784557
.float -0.788346
.float -0.792107
.float -0.795837
.float -0.799537
.float -0.803208
.float -0.806848
.float -0.810457
.float -0.814036
.float -0.817585
.float -0.821102
.float -0.824589
.float -0.828045
.float -0.831470
.float -0.834863
.float -0.838225
.float -0.841555
.float -0.844854
.float -0.848120
.float -0.851355
.float -0.854558
.float -0.857729
.float -0.860867
.float -0.863973
.float -0.867046
.float -0.870087
.float -0.873095
.float -0.876070
.float -0.879012
.float -0.881921
.float -0.884797
.float -0.887640
.float -0.890449
.float -0.893224
.float -0.895966
.float -0.898674
.float -0.901349
.float -0.903989
.float -0.906596
.float -0.909168
.float -0.911706
.float -0.914210
.float -0.916679
.float -0.919114
.float -0.921514

.float -0.923880
.float -0.926210
.float -0.928506
.float -0.930767
.float -0.932993
.float -0.935184
.float -0.937339
.float -0.939459
.float -0.941544
.float -0.943593
.float -0.945607
.float -0.947586
.float -0.949528
.float -0.951435
.float -0.953306
.float -0.955141
.float -0.956940
.float -0.958703
.float -0.960431
.float -0.962121
.float -0.963776
.float -0.965394
.float -0.966976
.float -0.968522
.float -0.970031
.float -0.971504
.float -0.972940
.float -0.974339
.float -0.975702
.float -0.977028
.float -0.978317
.float -0.979570
.float -0.980785
.float -0.981964
.float -0.983105
.float -0.984210
.float -0.985278
.float -0.986308
.float -0.987301
.float -0.988258
.float -0.989177
.float -0.990058
.float -0.990903
.float -0.991710
.float -0.992480
.float -0.993212
.float -0.993907
.float -0.994565

.float -0.995185
.float -0.995767
.float -0.996313
.float -0.996820
.float -0.997290
.float -0.997723
.float -0.998118
.float -0.998476
.float -0.998795
.float -0.999078
.float -0.999322
.float -0.999529
.float -0.999699
.float -0.999831
.float -0.999925
.float -0.999981
.float -1.000000
.float -0.999981
.float -0.999925
.float -0.999831
.float -0.999699
.float -0.999529
.float -0.999322
.float -0.999078
.float -0.998795
.float -0.998476
.float -0.998118
.float -0.997723
.float -0.997290
.float -0.996820
.float -0.996313
.float -0.995767
.float -0.995185
.float -0.994565
.float -0.993907
.float -0.993212
.float -0.992480
.float -0.991710
.float -0.990903
.float -0.990058
.float -0.989177
.float -0.988258
.float -0.987301
.float -0.986308
.float -0.985278
.float -0.984210
.float -0.983105
.float -0.981964

.float -0.980785
.float -0.979570
.float -0.978317
.float -0.977028
.float -0.975702
.float -0.974339
.float -0.972940
.float -0.971504
.float -0.970031
.float -0.968522
.float -0.966976
.float -0.965394
.float -0.963776
.float -0.962121
.float -0.960431
.float -0.958703
.float -0.956940
.float -0.955141
.float -0.953306
.float -0.951435
.float -0.949528
.float -0.947586
.float -0.945607
.float -0.943593
.float -0.941544
.float -0.939459
.float -0.937339
.float -0.935184
.float -0.932993
.float -0.930767
.float -0.928506
.float -0.926210
.float -0.923880
.float -0.921514
.float -0.919114
.float -0.916679
.float -0.914210
.float -0.911706
.float -0.909168
.float -0.906596
.float -0.903989
.float -0.901349
.float -0.898674
.float -0.895966
.float -0.893224
.float -0.890449
.float -0.887640
.float -0.884797

.float -0.881921
.float -0.879012
.float -0.876070
.float -0.873095
.float -0.870087
.float -0.867046
.float -0.863973
.float -0.860867
.float -0.857729
.float -0.854558
.float -0.851355
.float -0.848120
.float -0.844854
.float -0.841555
.float -0.838225
.float -0.834863
.float -0.831470
.float -0.828045
.float -0.824589
.float -0.821102
.float -0.817585
.float -0.814036
.float -0.810457
.float -0.806848
.float -0.803208
.float -0.799537
.float -0.795837
.float -0.792107
.float -0.788346
.float -0.784557
.float -0.780737
.float -0.776888
.float -0.773010
.float -0.769103
.float -0.765167
.float -0.761202
.float -0.757209
.float -0.753187
.float -0.749136
.float -0.745058
.float -0.740951
.float -0.736817
.float -0.732654
.float -0.728464
.float -0.724247
.float -0.720003
.float -0.715731
.float -0.711432

.float -0.707107
.float -0.702755
.float -0.698376
.float -0.693971
.float -0.689541
.float -0.685084
.float -0.680601
.float -0.676093
.float -0.671559
.float -0.667000
.float -0.662416
.float -0.657807
.float -0.653173
.float -0.648514
.float -0.643832
.float -0.639124
.float -0.634393
.float -0.629638
.float -0.624860
.float -0.620057
.float -0.615232
.float -0.610383
.float -0.605511
.float -0.600616
.float -0.595699
.float -0.590760
.float -0.585798
.float -0.580814
.float -0.575808
.float -0.570781
.float -0.565732
.float -0.560662
.float -0.555570
.float -0.550458
.float -0.545325
.float -0.540171
.float -0.534998
.float -0.529804
.float -0.524590
.float -0.519356
.float -0.514103
.float -0.508830
.float -0.503538
.float -0.498228
.float -0.492898
.float -0.487550
.float -0.482184
.float -0.476799

.float -0.471397
.float -0.465977
.float -0.460539
.float -0.455084
.float -0.449611
.float -0.444122
.float -0.438616
.float -0.433094
.float -0.427555
.float -0.422000
.float -0.416430
.float -0.410843
.float -0.405241
.float -0.399624
.float -0.393992
.float -0.388345
.float -0.382683
.float -0.377007
.float -0.371317
.float -0.365613
.float -0.359895
.float -0.354164
.float -0.348419
.float -0.342661
.float -0.336890
.float -0.331106
.float -0.325310
.float -0.319502
.float -0.313682
.float -0.307850
.float -0.302006
.float -0.296151
.float -0.290285
.float -0.284408
.float -0.278520
.float -0.272621
.float -0.266713
.float -0.260794
.float -0.254866
.float -0.248928
.float -0.242980
.float -0.237024
.float -0.231058
.float -0.225084
.float -0.219101
.float -0.213110
.float -0.207111
.float -0.201105

.float -0.195090
.float -0.189069
.float -0.183040
.float -0.177004
.float -0.170962
.float -0.164913
.float -0.158858
.float -0.152797
.float -0.146730
.float -0.140658
.float -0.134581
.float -0.128498
.float -0.122411
.float -0.116319
.float -0.110222
.float -0.104122
.float -0.098017
.float -0.091909
.float -0.085797
.float -0.079682
.float -0.073565
.float -0.067444
.float -0.061321
.float -0.055195
.float -0.049068
.float -0.042938
.float -0.036807
.float -0.030675
.float -0.024541
.float -0.018407
.float -0.012272
.float -0.006136
.float 0.000000
.float 0.006136
.float 0.012272
.float 0.018407
.float 0.024541
.float 0.030675
.float 0.036807
.float 0.042938
.float 0.049068
.float 0.055195
.float 0.061321
.float 0.067444
.float 0.073565
.float 0.079682
.float 0.085797
.float 0.091909

.float 0.098017
.float 0.104122
.float 0.110222
.float 0.116319
.float 0.122411
.float 0.128498
.float 0.134581
.float 0.140658
.float 0.146730
.float 0.152797
.float 0.158858
.float 0.164913
.float 0.170962
.float 0.177004
.float 0.183040
.float 0.189069
.float 0.195090
.float 0.201105
.float 0.207111
.float 0.213110
.float 0.219101
.float 0.225084
.float 0.231058
.float 0.237024
.float 0.242980
.float 0.248928
.float 0.254866
.float 0.260794
.float 0.266713
.float 0.272621
.float 0.278520
.float 0.284408
.float 0.290285
.float 0.296151
.float 0.302006
.float 0.307850
.float 0.313682
.float 0.319502
.float 0.325310
.float 0.331106
.float 0.336890
.float 0.342661
.float 0.348419
.float 0.354164
.float 0.359895
.float 0.365613
.float 0.371317
.float 0.377007

.float 0.382683
.float 0.388345
.float 0.393992
.float 0.399624
.float 0.405241
.float 0.410843
.float 0.416430
.float 0.422000
.float 0.427555
.float 0.433094
.float 0.438616
.float 0.444122
.float 0.449611
.float 0.455084
.float 0.460539
.float 0.465977
.float 0.471397
.float 0.476799
.float 0.482184
.float 0.487550
.float 0.492898
.float 0.498228
.float 0.503538
.float 0.508830
.float 0.514103
.float 0.519356
.float 0.524590
.float 0.529804
.float 0.534998
.float 0.540171
.float 0.545325
.float 0.550458
.float 0.555570
.float 0.560662
.float 0.565732
.float 0.570781
.float 0.575808
.float 0.580814
.float 0.585798
.float 0.590760
.float 0.595699
.float 0.600616
.float 0.605511
.float 0.610383
.float 0.615232
.float 0.620057
.float 0.624860
.float 0.629638

.float 0.634393
.float 0.639124
.float 0.643832
.float 0.648514
.float 0.653173
.float 0.657807
.float 0.662416
.float 0.667000
.float 0.671559
.float 0.676093
.float 0.680601
.float 0.685084
.float 0.689541
.float 0.693971
.float 0.698376
.float 0.702755
.float 0.707107
.float 0.711432
.float 0.715731
.float 0.720003
.float 0.724247
.float 0.728464
.float 0.732654
.float 0.736817
.float 0.740951
.float 0.745058
.float 0.749136
.float 0.753187
.float 0.757209
.float 0.761202
.float 0.765167
.float 0.769103
.float 0.773010
.float 0.776888
.float 0.780737
.float 0.784557
.float 0.788346
.float 0.792107
.float 0.795837
.float 0.799537
.float 0.803208
.float 0.806848
.float 0.810457
.float 0.814036
.float 0.817585
.float 0.821102
.float 0.824589
.float 0.828045

.float 0.831470
.float 0.834863
.float 0.838225
.float 0.841555
.float 0.844854
.float 0.848120
.float 0.851355
.float 0.854558
.float 0.857729
.float 0.860867
.float 0.863973
.float 0.867046
.float 0.870087
.float 0.873095
.float 0.876070
.float 0.879012
.float 0.881921
.float 0.884797
.float 0.887640
.float 0.890449
.float 0.893224
.float 0.895966
.float 0.898674
.float 0.901349
.float 0.903989
.float 0.906596
.float 0.909168
.float 0.911706
.float 0.914210
.float 0.916679
.float 0.919114
.float 0.921514
.float 0.923880
.float 0.926210
.float 0.928506
.float 0.930767
.float 0.932993
.float 0.935184
.float 0.937339
.float 0.939459
.float 0.941544
.float 0.943593
.float 0.945607
.float 0.947586
.float 0.949528
.float 0.951435
.float 0.953306
.float 0.955141

.float 0.956940
.float 0.958703
.float 0.960431
.float 0.962121
.float 0.963776
.float 0.965394
.float 0.966976
.float 0.968522
.float 0.970031
.float 0.971504
.float 0.972940
.float 0.974339
.float 0.975702
.float 0.977028
.float 0.978317
.float 0.979570
.float 0.980785
.float 0.981964
.float 0.983105
.float 0.984210
.float 0.985278
.float 0.986308
.float 0.987301
.float 0.988258
.float 0.989177
.float 0.990058
.float 0.990903
.float 0.991710
.float 0.992480
.float 0.993212
.float 0.993907
.float 0.994565
.float 0.995185
.float 0.995767
.float 0.996313
.float 0.996820
.float 0.997290
.float 0.997723
.float 0.998118
.float 0.998476
.float 0.998795
.float 0.999078
.float 0.999322
.float 0.999529
.float 0.999699
.float 0.999831
.float 0.999925
.float 0.999981

D.5 Archivo de configuración sistema homogéneo

```
:  
: "cfgapmas.nif" Archivo de configuración para el sistema  
: Homogéneo  
:  
:  
:
```

```
buffer_size 200;  
host_server tgio.exe.  
level_timeout 400: milisegundos  
decode_timeout 2000: milisegundos
```

```
: Descripción de la topología de la red y los programas  
: a ser cargados en cada nodo.  
:
```

```
1, apmasho, R0, 0, 2, . :  
2, apworho, R1, ., 1, . :
```

D.5 Archivo de configuración sistema heterogéneo

```
:  
: "cfgapmhe.nif" Archivo de configuración del sistema  
: Heterogéneo  
:  
:  
:  
:
```

```
buffer_size 200;  
host_server tgio.exe.  
level_timeout 400: milisegundos  
decode_timeout 2000: milisegundos
```

```
: Descripción de la topología de la red y los programas a ser  
: cargados en cada nodo.  
:
```

```
1, apmasho, R0, 0, 2, . :  
2, apworhe, R1, ., 1, . :
```