UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD INGENIERÍA

PRÁCTICAS DE LABORATORIO CON CON MICROPROCESADORES TMS320C6711

Dr. BOHUMIL PŠENIČKA
Ing. OMAR NIETO CRISÓSTOMO
Ing. VICTOR LÓPEZ MIRANDA

DIVISIÓN DE INGENIERÍA ELÉCTRICA
DEPARTAMENTO DE INGENIERÍA EN TELECOMUNICACIONES

Prólogo

La gran cantidad de aplicaciones que tiene el procesamiento digital de señales incluyen áreas, tales como: radar, sonar, voz, comunicaciones, telefonía, medicina, control, sismología, imágenes, etc. Las herramientas que han permitido obtener soluciones reales y eficientes son el desarrollo de las tecnologías de programación y los microprocesadores de procesamiento digital de señales.

El amplio crecimiento que ha tenido la industria digital se debe en gran medida al desarrollo de algoritmos de procesamiento digital de señales. Las áreas antes mencionadas requieren aplicaciones, tales como filtrado, compresión, análisis en frecuencia, entre otras. El uso de sistemas digitales permite realizar estas tareas con una computadora digital o un microprocesador.

Dentro de las herramientas de cómputo para el procesamiento digital de señales se encuentran los programas de simulación donde los datos pueden analizarse, aplicando filtros digitales o transformada de Fourier, y graficarse en una computadora. El paquete de cómputo MATLAB es un ejemplo de una poderosa herramienta para este tipo de simulaciones. Las tarjetas de microprocesadores *starter-kit* son otra ayuda para la ejecución de programas para procesamiento de señales.

El propósito de este libro es presentar, a los alumnos de las carreras de Ingeniería en Telecomunicaciones, Electrónica y Computación de la Facultad de Ingeniería de la UNAM, aplicaciones del procesamiento digital de señales.

En la primera parte se hace una presentación de la estructura básica del microcontrolador DSP TMS320C6711, se explican características generales del *Starter-kit* de TMS320C6711 y se realiza una breve descripción del conjunto de instrucciones del DSP TMS320C6711.

En la ultima parte, capítulo 6, se presentan programas elementales para que el alumno los ensamble y los corra en el *Starter-Kit* de TMS320C6711, observando en el analizador de espectro los resultados. El alumno, mediante los programas escritos para el paquete MATLAB y el simulador, puede verificar si dichos programas están bien hechos.

Este libro representa un gran esfuerzo de los autores para contribuir a que los alumnos tengan un material para desarrollar aplicaciones del procesamiento digital de señales a algunos problemas reales. Es nuestro deseo que los alumnos encuentren en este material una motivación para el estudio del procesamiento digital de señales. Se agradece el apoyo a la Facultad de Ingeniería de la UNAM y a las personas que colaboraron en el proceso para la publicación de esta obra.

Dr. Bohumil Pšenička
Ing. Omar Nieto Crisóstomo
Ing. Victor López Miranda

PRESENTACIÓN

La Facultad de Ingeniería ha decidido realizar una serie de ediciones provisionales de obras recientemente elaboradas por académicos de la institución, como material de apoyo para sus clases, de manera que puedan ser aprovechadas de inmediato por alumnos y profesores. Tal es el caso de las *Prácticas de laboratorio con microprocesadores TMS320C6711*, elaborados por Bohumil Psenicka, Omar Nieto Crisóstomo, Víctor López Miranda.

Se invita a los estudiantes y profesores a que comuniquen a los autores las observaciones y sugerencias que mejoren el contenido de la obra, con el fin de que se incorporen en una futura edición definitiva.

Índice General

Capítulo 1

Características y opciones del TMS320C62x/C67x

Las familias de dispositivos TMS320C62x / C67x, ejecutan un máximo de 8 instrucciones, de 32 bits, por ciclo. El CPU contiene 32 registros de propósito general, de 32 bits y 8 unidades funcionales. Estos dispositivos tienen un conjunto completo de herramientas de desarrollo y optimización, que incluyen un compilador C eficiente, un optimizador de ensamble para simplificar la planificación y programación del lenguaje ensamblador, y un depurador con interfase gráfica, basada en Windows, para visualizar las características de ejecución en el código fuente. Además, contiene una tarjeta de emulación de hardware compatible con la interfase del emulador TI XDS510. Estas herramientas cumplen con los estándares 1149.1-1990, revisión de acceso a puerto y arquitectura de verificación de límites, de la IEEE.

Las características de los dispositivos C62x / C67x, incluyen

1. Un CPU avanzado VLIW (very long instruction word) con 8 unidades funcionales, que incluyen 2 multiplicadores y 6 ALU's (unidades lógico aritméticas).
   (a) Ejecuta un máximo de 8 instrucciones por ciclo, 10 veces más que los DSP's típicos.
   (b) Permite rápido tiempo de desarrollo en diseños con código RISC altamente efectivos.
2. Empaquetado de instrucción
   (a) Obtiene el tamaño del código equivalente a las 8 instrucciones ejecutadas serialmente o en paralelo.
   (b) Reduce el tamaño del código, el consumo de energía y el *fetch* del programa.
3. Ejecución condicional de todas las instrucciones.
   (a) Reduce los saltos costosos.
   (b) Incrementa el paralelismo para mantener un alto desempeño.
4. Ejecuta el código programado, en unidades funcionales independientes.
5. Proporciona soporte eficiente de memoria para una variedad de aplicaciones de 8, 16 y 32 bits de datos.

6. Maneja operaciones aritméticas de 40 bits, adicionando precisión extra a vocoders y otras aplicaciones computacionalmente intensivas.

7. Proporciona soporte de normalización y saturación, en operaciones aritméticas claves.

8. Soporta operaciones comunes, halladas en aplicaciones de control y manipulación de datos, como: manipulación de campos, extracción de instrucción, activación, desactivación y conteo de bits

Además, el C67x tiene las siguientes características:

- Un máximo de 1336 MIPS (millones de instrucciones por segundo), a 167 MHz.
- Un máximo de 1 G FLOPS (operaciones en punto flotante por segundo), a 167 MHz, para operaciones de precisión simple.
- Un máximo de 250 M FLOPS a 167 MHz, para operaciones de doble precisión.
- Un máximo de 688 M FLOPS a 167 MHz, para operaciones de multiplicación y acumulación.
- Soporte de hardware para operaciones de punto flotante, de simple y doble precisión (con formato IEEE).
- Multiplicación entera de 32 x 32 bits, con resultado de 32 o 64 bits.

Los dispositivos C62x / C67x tienen la siguiente variedad de opciones en memoria y periféricos:

- Amplia memoria RAM para ejecución rápida de algoritmos.
- Soporta interfases para memoria externa de 32 bits (SDRAM, SBSRAM, SRAM y otras memorias asíncronas), para aumentar el rango de memoria externa y maximizar el desempeño del sistema.
- Acceso a la memoria y periféricos de los dispositivos C62x / C67x a través del puerto *host*.
- Controlador del multicanal DMA
- Puerto (s) serie multicanal.
- Timer (s) de 32 bits.

Capítulo 2

Arquitectura de los dispositivos TMS320C62x/c67x

Los dispositivos 'C6211, 'C6711, 'C6701, 'C6201 y 'C6202 operan a 150, 150, 167, 200 y 250 Mhz respectivamente. Todos estos DSP's ejecutan un máximo de 8 instrucciones por ciclo. El DSP 'C6211 es de punto fijo mientras que el 'C6711 es de punto flotante.

Los procesadores 'C62x/C67x consisten de tres partes: el CPU, los periféricos y la memoria. Ocho unidades funcionales operan en paralelo (seis ALU's y dos multiplicadores), con dos conjuntos similares de cuatro unidades funcionales básicas. Las unidades se comunican usando un camino cruzado entre dos clasificaciones de registros, cada una de los cuales contiene 16 registros de 32bits. La figura 6.10 muestra el diagrama de bloque de los dispositivos TMS320C62x/C67x.

2.1 Unidad de procesamiento Central (CPU)

El CPU contiene:

- Unidad *fetch* de programa
- Unidad de despacho de instrucción
- Unidad de decodificación de instrucción
- 32 registros de 32 bits
- Dos caminos de datos (*path*), cada uno con cuatro unidades funcionales
- Registros de control
- Lógica de control
- Lógica de interrupción, emulación y prueba

El CPU tiene dos caminos de datos (A y B), cada camino tiene cuatro unidades funcionales (.L, .S, .M y .D) y un archivo de registros que contiene 16 registros de 32 bits *(register file)*. Las unidades funcionales ejecutan operaciones de lógica, corrimiento, multiplicación y

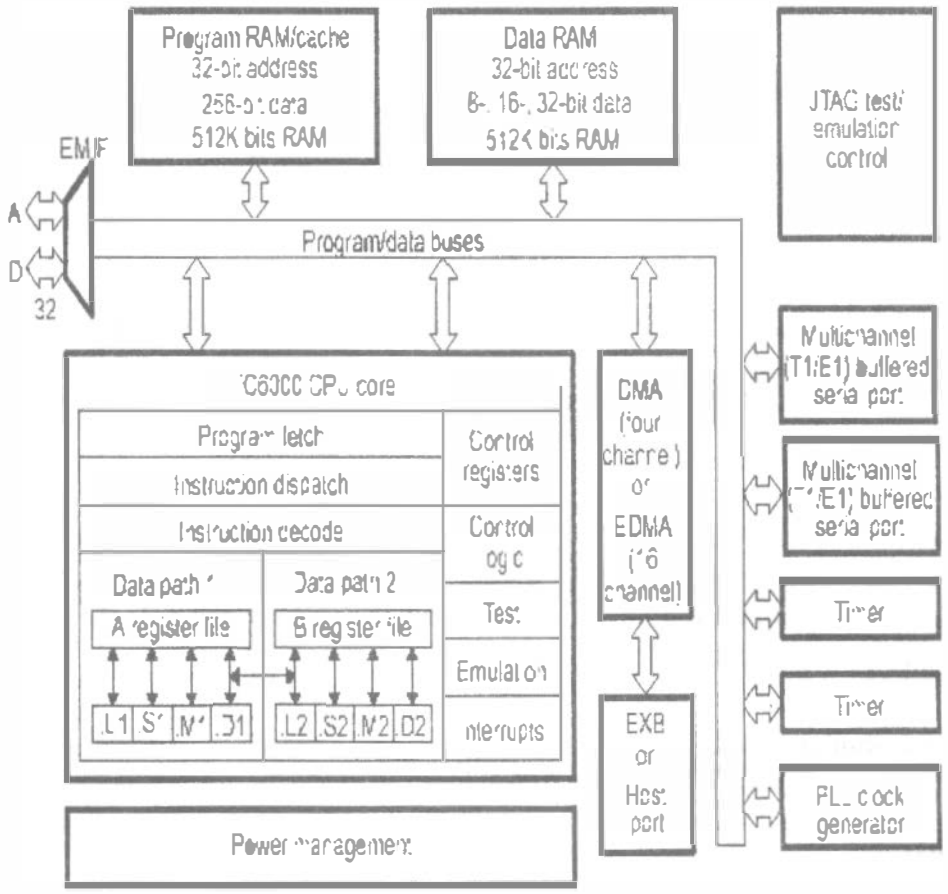

Figura 2.1: Diagrama de bloques de TMS320C62x/C67x.

direccionamiento de datos. Todas las instrucciones aceptan operaciones de carga y almacenamiento sobre los registros. Las dos unidades de direccionamiento de datos (.D1 y .D2) son exclusivamente responsables de toda la transferencia de datos entre los archivos de registros y la memoria.

2.2 Caminos de datos del CPU

Los caminos de datos del CPU consisten de: dos archivos de registros de propósito general (A y B), ocho unidades funcionales (.L1, .L2, .S1, .S2, .M1, .M2, .D1 y .D2), dos caminos de lectura de memoria (LD1 y LD2), dos caminos de almacenamiento en memoria (ST1 y ST2), dos caminos cruzados entre los archivos de registros (1X y 2X) y dos caminos de direccionamiento de datos (DA1 y DA2). La figura 2.2 muestra el camino de los datos del CPU 'C67x.

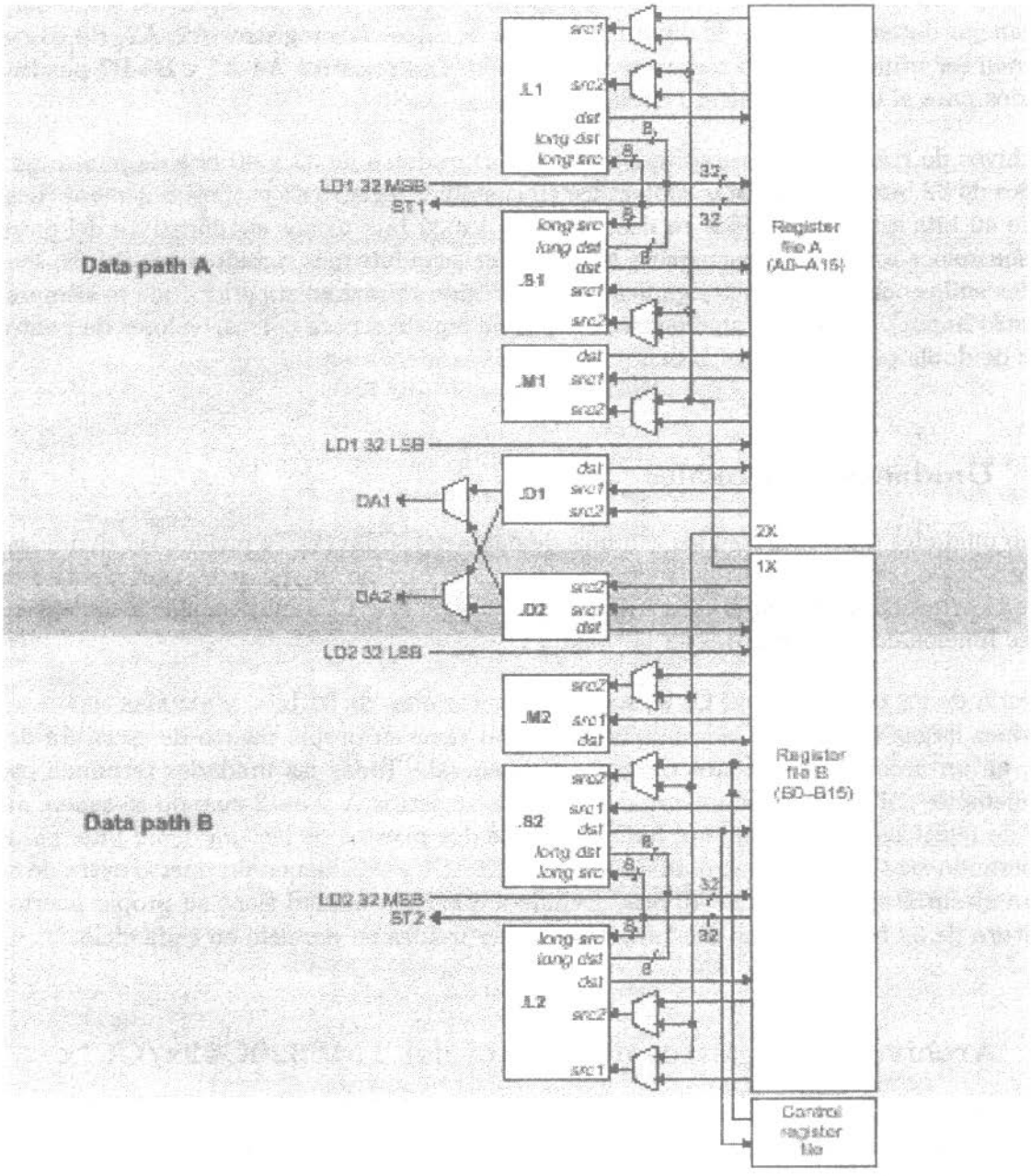

Figura 2.2: Camino de datos del TMS320C67x

2.2.1 Archivos de registros de propósito general *(register files)*

Hay dos archivos de registros de propósito general (A y B) en los caminos de datos del 'C62x/'C67x. Cada uno de esos archivos contiene 16 registros de 32 bits (A0-A15) para el archivo A y (B0-B15) para el archivo B. Los registros de propósito general pueden ser usados para manejar datos o punteros de direccionamiento de estos. Los registros A1, A2, B0, B1 y B2 pueden ser utilizados como registros de condición. Los registros A4-A7 y B4-B7 pueden ser usados para el direccionamiento circular.

Los archivos de registros de propósito general soportan datos de 32 y 40 bits de punto fijo. Los datos de 32 bits, pueden estar contenidos en cualquier registro de propósito general. Los datos de 40 bits están contenidos en dos registros; los 32 bits menos significativos del dato (LSB) son colocados en un registro par y los restantes ocho bits mas significativos (MSB) son colocados en los ocho bits menos significativos del registro próximo superior (que es siempre un registro impar). El 'C67x también usa ese par de registros para colocar valores de punto flotante de doble precisión de 64 bits.

2.2.2 Unidades funcionales

Las ocho unidades funcionales en los caminos de datos del 'C62x/C67x pueden ser divididas en dos grupos de cuatro; cada unidad funcional, en un camino de datos, es casi idéntica a la unidad correspondiente, en el otro camino de datos p. ej., .L es muy similar a .L2). Las unidades funcionales son descritas en la tabla 2.1

La mayoría de los caminos en el CPU, soportan operaciones de 32 bits, y algunas soportan operaciones largas (40 bits). Cada unidad funcional tiene su propio puerto de escritura de 32 bits, en un archivo de registros de propósito general. Todas las unidades terminan en 1 (por ejemplo, .L1) cuando se refiere al archivo de registros A y en 2 cuando se refiere al archivo de registros B. Cada unidad funcional tiene dos puertos de lectura de 32 bits, para cada operando *src1 y src2*. Cuatro unidades (.L1, .L2, .S1 y .S2) tienen un puerto extra de 8 bits para ejecutar operaciones de 40 bits. Debido a que cada unidad tiene su propio puerto de escritura de 32 bits, las ocho unidades pueden ser usadas en paralelo en cada ciclo.

2.2.3 Archivos de registros de control del TMS320C62x/C67x

Una unidad (.S2) puede leer de y escribir hacia los registros de control. Mostrados en la figura 2.2. La tabla 2.2, menciona y describe, los registros de control contenidos en el archivo de registros de control. Cada registro es accesado con la instrucción MVC.

El 'C67x posee tres registros de configuración adicionales, para soportar operaciones de punto flotante (vea la tabla 2.3). Los registros especifican los modos de redondeo de punto flotante para las unidades .M y .L. También contienen campos de bit para advertir si *src1* y *src2* son NaN (no es un número) o números desnormalizados. Además si resulta *overflow* o *underflow*, es inexacto, infinito o invalido. Hay campos que advierten si una división por cero fue ejecutada o si una comparación fue ejecutada con un NaN.

Unidad Funcional	Operaciones en punto fijo	Operaciones en punto flotante
Unidad .L(.L1, .L2)	Operaciones aritméticas y comparación de 32 y 40 bits. Cuenta de 0's o 1's mas a la izguierda, para 32 bits. Normalización de 32 y 40 bits. Operaciones lógicas de 32 bits	Operaciones aritméticas. Operaciones de conversión: DP->SP, INT->DP, INT->SP
Unidad .S(.S1, .S2)	Operaciones aritméticas de 32 bits Corrimientos de 32/40 bits y operaciones campos de bits en 32 bits. Operaciones lógicas de 32 bits. Saltos. Generación de constantes. Transferencia de registros de/hacia registros (solamente .S2)	Comparación recíproca. Operaciones de raíz cuadrada. Operaciones de valor absoluto. Operaciones de Conversión SP a DP
Unidad .M(.M1,.M2)	Operaciónes de multiplicación de 16x16 bits	Operaciones de multiplicación de 32x32 bits. Operaciones de multiplicación de punto flotante
Unidad .D(.D1,.D2)	Sumas, restas y cálculos de direccionamiento circular de 32 bits Carga y almacenamiento con offset constante de 5 bits. Carga y almacenamiento con offset constante de 15 bits. (solo .D2)	Lectura de palabras dobles con offset constante de 5 bits.

Tabla 2.1: Unidades funcionales y las operaciones rea izádas

Abreviatura	Nombre	Descripción
AMR	Registro de modo de direccionamiento	Especifica si utilisa direccionamiento lineal o circular para cada uno de los ocho registros; también contiene el tamaño para el direccionamiento circular
CSR	Registro de control de estado	Contiene el bit de interrupción global, los bits de control del cache y otros bits de control de estado misceláneos
IFR	Registro de bandera de interrupción	Despliega el estado de las interrupciones
ISR	Registros de para activar interrupción	Permite activar interrupciones manualmente
ICR	Registro para interrupción	Permite limpiar interrupciones pendientes manualmente
IER	Registro para retorno de interrupción	Permite habilitar / deshabilitar interrupciones individuales
NRP	Puntero de retorno de interrupción no mascarable	Contiene la dirección de retorno de una interrupción no ma carable
PCE1	Contador del programa, fase E1	Contiene la dirección del paquete fetch (contiene el paquete de ejecución del pipeline) en la etapa E1.

Tabla 2.2: Registros de control

Abreviatura	Nombre	Descripción
FADCR	Registro de configuración del sumador del punto flotante	Especifica el modo underflow, modo de redondeo, NaN y otras excepciones para la unidad .L
FAUCR	Registro de configuración auxiliar de punto flotante	Especifica modos de underflow, modos de redondeo, NaN y otras excepciones para la unidad .S
FMCR	Registro de configuración del multiplicador de punto flotante	Especifica modos de underflow, modos de redondeo, NaN y otras excepciones para la unidad .M

Tabla 2.3: Registros de control extendidos para el TMS320C67x

2.2.4 Caminos entre archivos de registros *(Register File Cross Paths)*

Cada unidad funcional lee directamente de y escribe directamente hacia el archivo de registros, dentro de su propio camino de datos. Esto es, las unidades .L1, .S1, .D1 y .M1 escriben en el archivo de registros A y las unidades .L2, .S2, .D2 y .M2 escriben en el archivo de registros B. Los archivos de registros son conectados a las unidades funcionales del archivo de registros opuesto, através de los caminos cruzados 1X y 2X. Esos caminos cruzados permiten a las unidades funcionales, de un camino de datos, accesar a operandos de 32 bits, del lado opuesto. El camino cruzado 1X permite a las unidades funcionales del camino de datos A, leer su operando fuente del archivo de registros B. El camino cruzado 2X permite a las unidades funcionales del camino de datos B, leer su operando fuente del archivo de registros A.

2.2.5 Caminos de Memoria, Cargas y Almacenamiento

Hay dos caminos de 32 bits, para leer los datos de memoria en los registros de almacenamiento: LD1 para el archivo de registros A y LD2 para el archivo de registros B. El 'C67x también tiene un segundo camino de carga de 32 bits para ambos archivos de registros A y B. Este segundo camino permite a la instrucción LDDW leer simultáneamente dos registros de 32 bits en los lados A y B. Existen además dos caminos de 32 bits, ST1 y ST2, para almacenar valores de los registros a la memoria, para cada archivo de registros. Los caminos de lectura largos .L y .S son compartidos con los caminos de almacenamiento.

2.2.6 Caminos de direccionamiento de datos

Los caminos de direccionamiento de datos (DA1 y DA2) mostrados en la figura 2.2 colocados fuera de las unidades .D, permiten generar direcciones de datos de un archivo de registros. Con eso se sostienen cargas y almacenamientos en memoria, desde el otro archivo de registros. Sin embargo, las cargas y almacenamientos ejecutados en paralelo, debe cargar a y de el mismo archivo de registro. Aunque también existe la alternativa de que ambos usen un camino cruzado al registro opuesto.

2.2.7 Mapeo Entre Instrucciones y Unidades Funcionales

La tabla 2.4 muestra el mapeo entre las instrucciones y las unidades funcionales para las instrucciones de punto fijo del TMS320C62x/C67x. La tabla 2.5 muestra el mapeo entre las instrucciones y las unidades funcionales para las instrucciones de punto flotante del TMS320C67x.

2.3 Modos de direccionamiento

Los modos de direccionamiento son lineales por default para los 'C62x y 'C67x aunque también existe el modo de direccionamiento circular. El modo de direccionamiento se especifica con el registro modo de direccionamiento (AMR).

Con todos los registros se puede ejecutar el direccionamiento lineal. Solo en ocho de ellos se puede ejecutar el direccionamiento circular: del A4 a A7 (que son usados por la unidad

Unidad .L	Unidad .M	Unidad .S		Unidad .D	
ABS	MPY	ADD	SET	ADD	STB(15-bitt offset) Solo .S2
ADD	MPYU	ADDK	SHL	ADDAB	STH (15-bitt offset) Solo .S2
ADDU	MPYUS	ADD2	SHR	ADDAH	STW(15-bitt offset) Solo .S2
AND	MPYSU	AND	SHRU	ADDAW	SUB
CMPEQ	MPYH	B disp	SHRL	LDB	SUBAB
CMPGT	MPYHU	B IRP	SUB	LDBU	SUBAH
CMPGTU	MPYHUS	B NRP	SUBU	LDH	SUBAW
CMPLT	MPYHSU	B reg	SUB2	LDHU	ZERO
CMPLTU	MPYHL	CLR	XOR	LDW	
LMBD	MPYHLU	EXT	ZERO	LDB	
MV	MPYHULS	EXTU		LDBU	
NEG	MPYHSLU	MV		LDH	
NORM	MPYLH	MVC		LDHU	
NOT	MPYLHU	MVK		LDW	
OR	MPYLUHS	MVKH		MV	
SADD	MPYLSHU	MVLKH		STB	
SAT	SMPY	NEG		STH	
SSUB	SMPYHL	NOT		STW	
SUB	SMPYLH	OR			
SUBU	SMPYH				
SUBC					
XOR					
ZERO					

Tabla 2.4: Mapeo de instrucciones de punto fijo y unidades funcionales

Unidad .L	Unidad .M	Unidad .S	Unidad .D
ADDDP	MPYDP	ABSDP	ADDAD
ADDSP	MPYI	ABSSP	LDDW
DPINT	MPYID	CMPEQDP	
DPSP	MPYSP	CMPEQSP	
INTDP		CMPGTDP	
INTDPU	CMPGTSP		
INTSP		CMLTDP	
INTSPU	CMPLTSP		
SPINT		RCPDP	
SPTRUNC		RCPSP	
SUBDP		RSQRDP	
SUBSP		RSQRSP	
SPDP			

Tabla 2.5: Mapeo de instrucciones de punto flotante y unidades funcionales

Tipo de direccionamiento	Ninguna modificación del registro de dirección	Preincremento o predecremento del registro del dirección	Postincremento o postdecremento del registro de dirección
Registro indirecto	*R	*++R	*R++
		*- -R	*R- -
Registro relativo	*+R[*ucst5*]	*++R[*ucst5*]	*R++[*ucst5*]
	*-R[*ucst5*]	*- -R[*ucst5*]	*R- -[*ucst5*]
Base + index	*+R[*offsetR*]	*++R[*offsetR*]	*R++[*offsetR*]
	*-R[*offsetR*]	*- -R[*offsetR*]	*R- -[*offsetR*]

Tabla 2.6: Generación de direccionamiento indirecto para Load/Store

.D1) y del B4 a B7 (que son usados por la unidad D2). Ninguna otra unidades puede ejecutar direccionamiento circular. Las instrucciones LDB/LDH/LDW, STB/STH/STW, ADDAB/ADDAH/ADDAW, y SUBAB/SUBAH/SUBAW se apoyan en el registro AMR, para determinar que tipo de cálculo del direccionamiento es ejecutado por esos registros.

Los CPUs 'C62x/'C67x tienen arquitectura de carga/almacenamiento, lo que significa que la única manera de accesar datos en memoria es con la instrucción de carga o almacenamiento. La tabla 2.6 muestra la síntaxis de un direccionamiento indirecto, para una localización en memoria.

2.4 Interrupciones

Los CPUs 'C62x/C67x tienen 14 interrupciones. Estas son reset, la interrupción no mascarable (NMI) e interrupciones de la 4 a la 15. Estas interrupcion s corr sponden a las señales RESET, NMI e INT4-INT15 respectivamente, sobre los límites del CPU. En los mismos dispositivos 'C62x/C67x, estas señales pueden estar ligadas directamente a lo pines del dispositivo, conectando periféricos al chip, o pueden ser desactivadas permanentemente, cuando están ligadas e inactivas en el chip. Generalmente, RESET y NMI son conectadas directamente a los pines del dispositivo. Las características del servicio de interrupción incluyen:

- El pin IACK del CPU es usado para confirmar la recepción de una petición de interrupción
- Los pines INUM0 INUM3 indican el vector de interrupción que está siendo utilizado
- Los vectores de interrupción son reubicables
- Los vectores de interrupción consisten de un paquete fetch. Con los paquetes se proporciona un rápido servicio.

Para obtener más información sobre la arquit ctura del TMS320C62x/C67x, revisar el manual TMS320C62x/C67x CPU e *Instruction Set Reference Guide,* de Texas Instrument.

Capítulo 3

Periféricos

Los periféricos disponibles en los dispositivos TMS320C6000 se muestran en la tabla 3.1 Los periféricos que son accesibles al usuario se configuran con un conjunto de registros de control mapeados en memoria. El controlador del bus de periféricos realiza el albitraje para el acceso a los periféricos. La lógica de configuración de Boot está conectada por señales externas y la lógica de baja energía es accesible directamente por el CPU. La figura 3.1 muestra un diagrama de bloques, con los periféricos de los dispositivos 'C6211/'C6711.

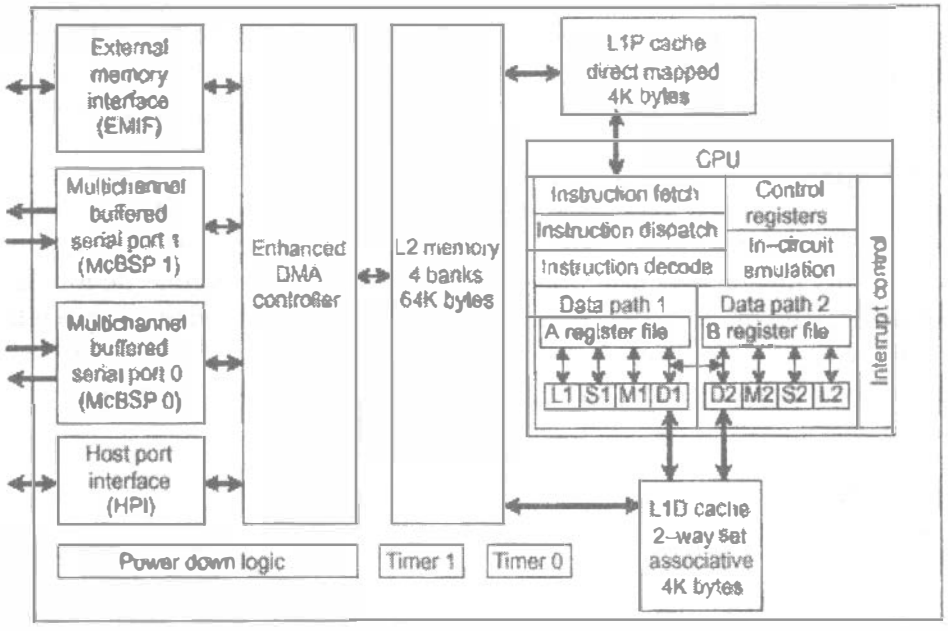

Figura 3.1: Diagrama de bloques de los dispositivos TMS320C6211/C6711

1. **Controlador DMA.** El controlador DMA transfiere datos entre rangos de direccionamiento en el mapa de memoria, sin intervención del CPU. El controlador DMA tiene cuatro canales programables y cinco canales auxiliares.

2. **Controlador EDMA.** El controlador EDMA mejora las mismas funciones del controlador DMA. El EDMA tiene dieciséis canales programables, así como un espacio de

Periféricos	C6201	C6202	C6211	C6701	C6711
Controlador de Acceso Directo a Memoria(DMA)	Y	Y	N	Y	N
Controlador de Acceso Directo a Memoria Mejorado(EDMA)	N	N	Y	N	Y
Interfase al Puerto Host (HPI)	Y	N	Y	Y	Y
Bus de Expansión	N	Y	N	N	N
Interfase de Memoria Externa (EMIF)	Y	Y	Y	Y	Y
Cinfiguración del Boot	Y	Y	Y	Y	Y
Puertos Seriales de Multicanal (McBSPs)	2	3	2	2	2
Selector de Interrupción	Y	Y	Y	Y	Y
Timers de 32-bits	2	2	2	2	2
Lógica de Energía baja	Y	Y	Y	Y	Y

Tabla 3.1: Periféricos de los dispositivos TMS320C6000

RAM para soportar múltiples configuraciones de futuras transferencias.

3. **HPI.** El HPI es un puerto paralelo por medio de cual, un procesador host puede accesar directamente al espacio en memoria del CPU. El dispositivo host tiene facilidad de acceso debido a que es el maestro de la interfase. El host y el CPU pueden intercambiar información a través de la memoria interna o externa. En suma, el host tiene acceso directo a los periféricos de memoria mapeada.

4. **Bus de Expansión.** El bus de expansión es un remplazo para el HPI, así como una expansión del EMIF. La expansión proporciona dos áreas distintas de funcionalidad, (puerto host y puertos de Entrada /Salida) que pueden coexistir en un sistema. El puerto host del bus de expansión puede operar en modo asíncrono de forma (esclavo), similar al HPI o en modo síncrono (maestro/esclavo) Estos modos permiten la interfase de dispositivos para una variedad de protocolos del bus del host. FIFOs síncronos y los dispositivos periféricos de Entrada/Salida asíncronos pueden conectarse al bus de expansión

5. **EMIF**. El EMIF soporta una interfaz de baja adherencia *(glueless)* para varios dispositivos externos, incluye

   (a) SRAM de refága síncrona (SBSRAM)

   (b) DRAM síncrona (SDRAM)

   (c) Dispositivos asíncronos, incluyendo SRAM, ROM y FIFOs

   (d) Dispositivo externo de memoria compartida

6. **Configuración del Boot.** Los dispositivos TMS320C62x/C67x proporcionan una variedad de configuraciones del boot, que determinan los acciones de inicialización que ejecuta el DSP, después del reset del dispositivo. Estas incluyen: cargas de código de un espacio externo de ROM sobre el EMIF y cargas de código a través del HPI/bus de un host externo.

7. **McBSP.** El puerto serial multicanal con buffer (McBSP) está basado en las interfases estándar del puerto serie, encontrada en los dispositivos con plataformas TMS320C2000 y 'C5000. Resumiendo, el puerto puede almacenar muestras seriales en un buffer de memoria automáticamente, con la ayuda del controlador DMA/EDMA. Este también tiene capacidad de multicanal, compatible con los estándares de conexión de redes T1, E1, SCSA y MVIP. Proporciona:

   (a) Comunicación full-Duplex

   (b) Registros de datos de doble buffer para flujo continuo de datos

   (c) Tramado independiente y temporización para dispositivos y transmisión

   (d) Interfase directa a codecs estándar, chips de interface analógica (AICs) y otros dispositivos A/D y D/A conectados serialmente

8. Tiene las siguientes capacidades:

   (a) Interfase directa a:

      i. Tramas T1/E1

      ii. Dispositivos conforme a $ST - BUS^{TM}$

      iii. Dispositivos conforme a IOM-2

      iv. Dispositivos conforme a AC97

      v. Dispositivos conforme a IIS

      vi. Dispositivos SPI^{TM}

   (b) Transmisión y recepción multicanal de 128 canales

   (c) Un selector del ancho del tamaño del dato, que incluye 8, 12, 16, 20, 24 y 32 bits

   (d) $Ley - \mu$ y Ley-A de compansión

   (e) Transferencia inicial de 8 bits con LSB (bit meuos significativo) o MSB (bit mas significativo)

   (f) Polaridad programable para ambas tramas de sincronización y relojes de datos

   (g) Reloj interno altamente programable y generación de trama

9. **TIMER.** Los dispositivos 'C6000 tienen dos timer de propósito general que son usados para:

   (a) Eventos del timer

   (b) Eventos de contador

   (c) Generador de pulsos

   (d) Interrupción del CPU

   (e) Enviar eventos de sincronización a el controlador DMA/EDMA

10. **Selector de interrupción.** El conjunto de periféricos del 'C6000 producen 14 a 16 fuentes de interrupción. El CPU tiene 12 interrupciones disponibles. El selector de interrupción permite elegir entre las 12 interrupciones, la que necesita su sistema. El selector de interrupción, también permite cambiar la polaridad de entrada para la interrupción externa.

11. **Lógica de bajo consumo de energía.** La lógica de bajo consumo de energía permite reducir el reloj para disminuir el consumo de energía. La mayoría de la potencia de operación de la lógica CMOS, se disipa durante la conmutación del circuito de un estado lógico a otro. Para prevenir algo o toda la lógica del chip de conmutación, se puede realizar ahorros significativos de energía, sin perder datos ni contexto operacional.

Capítulo 4

Code Composer Studio

El *Code Composer Studio* (CCS) mejora y acelera el proceso de desarrollo, para los programadores, que crean y prueban, en tiempo real, aplicaciones de procesamiento de señales incrustadas *(embedded)*. El CCS proporciona herramientas para la configuración, construccin, depuración, mensajes del programa *(tracing)* y análisis del programa. El *Code Composer Studio* incluye los siguientes componentes

- Herramientas de generación de código paraTMS320C6000
- Entorno en Desarrollo integrado (IDE) del *Code Composer Studio*
- DSP/BIOS *plug-ins* y API's
- RTDX *plug-ins*, interfase *host* y API's

Estos componentes trabajando en conjunto se muestran en la figura 4.1

4.1 Herramientas de dasarrollo para generación de código

Los TMS320C6000 soportan un conjunto de herramientas, para el desarrollo del software, qué incluyen: compilador C/C++ optimizado, ensamblador optimizado, ensamblador, ligador y las herramientas asociadas a ellos. Además, el TMS320C6000 soporta las siguientes herramientas, para el desarrollo del lenguaje ensamblador.

- Ensamblador *(Assembler)*
- Archivador *(Archiver)*
- Ligador *(Linker)*
- Listado Absoluto *(Absolute lister)*
- Listado de referencias cruzadas *(Cross-reference lister)*
- Utilería de conversión hexadecimal *(Hex conversion utility)*

La figura 4.2 muestra el flujo para desarrollo de software, con el TMS320C6000. La parte sombreada, representa el camino más común de desarrollo; las demás partes son opcionales. Estas otras partes, representan funciones periféricas que enlazan el proceso de desarrollo.

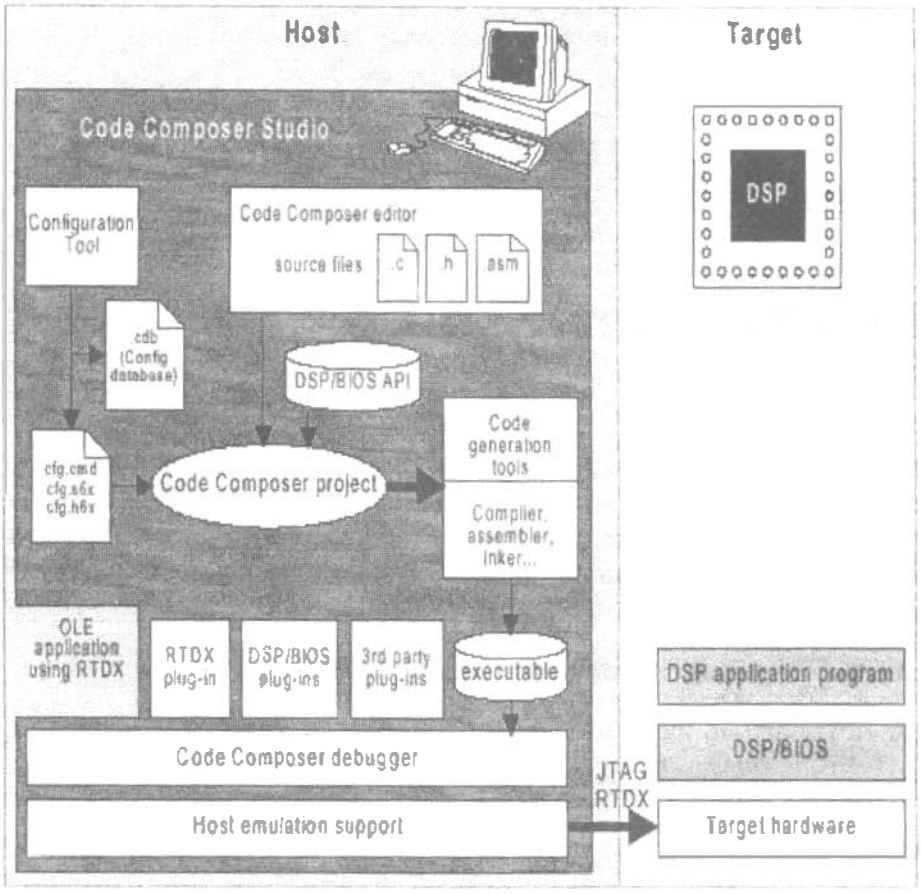

Figura 4.1: Componentes de trabajo del CCS.

4.1.1 Descripción de Herramientas

- El **optimizador de ensamble** *(assembly optimizer)*, permite escribir código en lenguaje ensamblador lineal, sin importar la estructura del pipeline o la asignación de registros. Asigna registros y usa optimización de ciclos, para convertir el código, de lenguaje ensamblador lineal a lenguaje ensamblador en paralelo.

- **Compilador** $C/C++$. Acepta código fuente en lenguaje $C/C++$ y produce código fuente en lenguaje ensamblador para el TMS320C6000. Para invocar al shell del compilador la instrucción tiene la forma siguiente:

$$\boxed{\mathbf{cl6x}[options][filenames][-\mathbf{z}[linker\ options][object\ files]]}$$

- El **ensamblador** *(assembler)* traduce los archivos de lenguaje ensamblador fuente en lenguaje de máquina. Invoque el ensamblador de la siguiente forma:

$$\boxed{\mathbf{asm6x}[input\ file[object\ file[listing\ file]]][options]}$$

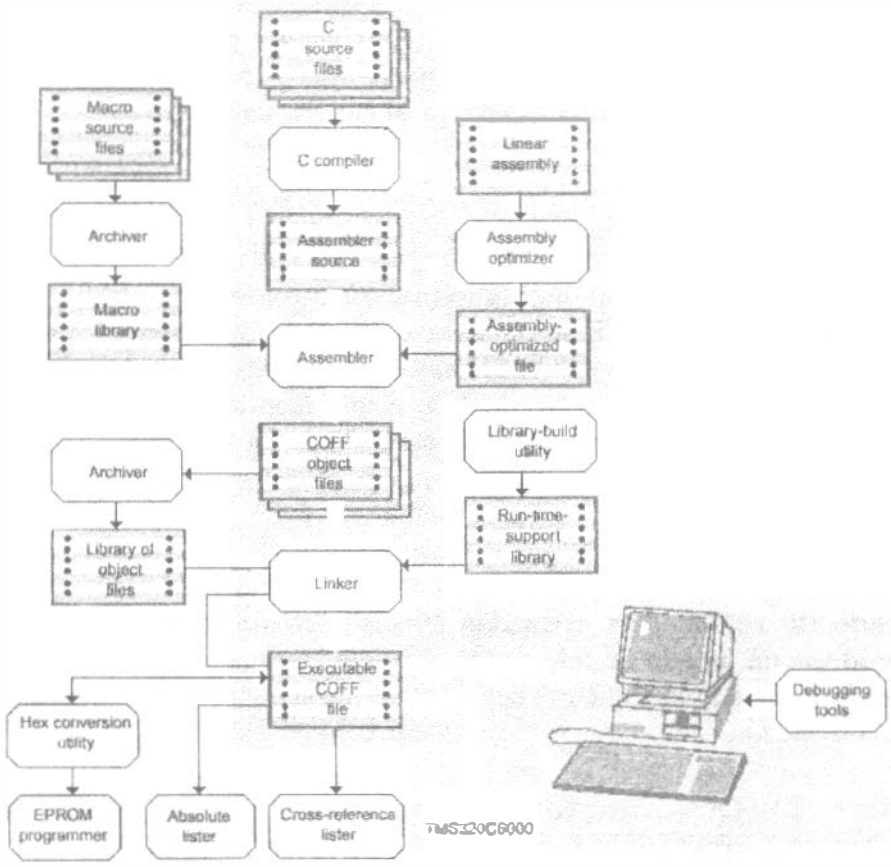

Figura 4.2: Flujo para el desarrollo de programas para el TMS320C6000.

- El **ligador** *(linker)* Combina los archivos objeto a un solo archivo ejecutable para el DSP. También accepta archivos de librerías y módulos de salidas, creaddos por una ejecución previa del mismo. La síntaxis general para invocar al ligador es la siguiente

$$\boxed{\textbf{lnk6}[options]filenames1...filenamen}$$

- El **archivador** *(archiver)* Permite juntar un grupo de archivos dentro de un solo archivo, llamado librería. Por ejemplo, se puede juntar distintas macros, dentro de una librería de macros. Invoque al archiver de la siguiente forma

$$\boxed{\textbf{ar6x}[-]command[options]libname[filename1...filenamen]}$$

- También se puede usar **la utilería de construcción de librerías** *(library-build utility)* para construir su propia librería de soporte en tiempo real.

- El **Listado absoluto** *(absolute lister)* es una heramienta para depuración de programas. Aceptan como entrada, archivos objetos ligados y crea archivos con extensión .abs que son ensamblados para producir una lista que muestra las direcciones absolutas de código objeto. La síntaxis para invocar el listado alabsoluto es:

abs6x[*−options*]*input files*

- La **utilería de conversión hexadecimal** *(hex conversion utility.)* Convierte un archivo objeto de tipo COFF en un archivo cuyo formato objeto se puede seleccionar entre los siguientes: TI-Tagged, ASCII hexadecimal, Intel, Motorola-S o Tektronix. El archivo convertido, puede ser transmitido a una memoria EPROM. Para invocar la utileria de conversión hexadecimal es de la siguiente forma

hex6x[*options*]*filename*

- El **listado de referencias cruzadas** *(Cross-reference lister).* Usa archivos objeto para producir un listado de referencias cruzadas, mostrando símbolos, definiciónes y sus referencia en el archivo fuente ligado. Invoque al listado de referencias cruzadas de la siguiente forma:

xref6x[*options*][*input filename*[*output filename*]]

4.1.2 Estructura del código en ensamblador

Un programa de lenguaje en ensamblador debe ser un archivo de texto en código ASCII. Cualquier línea del código ensamblador puede incluir como máximo, los siete elementos

- Etiqueta
- Barras paralelas
- Condiciones
- Instrucciones
- Unidad Funcional
- Operandos
- Comentarios

• Etiquetas

Una etiqueta identifica una línea de código, o una variable y representa una dirección de memoria, que contiene cualquiera instrucción o dato. A continuación se muestra la posición de la etiqueta, en una línea de código ensamblador. Los dos puntos posteriores a la etiqueta son opcionales.

label : [*condition*] *instruction unit operands comments parallel bars*

Las etiquetas deben reunir las siguientes condiciones:

- El primer caracter de la etiqueta debe ser una letra o un guión bajo (_) seguido por una letra
- La etiqueta debe estar en la primer columna del archivo de texto
- La etiqueta puede incluir hasta 32 caracteres alfanuméricos

• Barras Paralelas

Para indicar que una instrucción se ejecuta en paralelo con la instrucción previa, se indica con las barras paralelas ||. Este campo es un espacio en blanco, para una instrucción que no se ejecuta en paralelo, con la instrucción previa.

label : **paralel barrs** [*condition*] *instruction unit operands* ; *comments*

• Condiciones

El 'C6000 tiene cinco registros disponibles para las condiciones: A1, A2, B0, B1 y B2. A continuación se muestra la posición de una condición de una línea de código ensamblador

label : *paralel barrs* [**condition**] *instruction unit operands* ; *comments*

Todas las instrucciones del 'C6000 son condicionales:

- Si no se especifica ningúna condición, la instrucción siempre será ejecutada.
- Si se especifica y esa condición es verdadera, la instrucción se ejecuta. Por ejemplo:

Con esta condición.....	**La instrucción se executa sí...**
[A1]	$A1! = 0$
[!A1]	$A1 = 0$

- Si se especifica una condición y es falsa, la instrucción no se ejecuta.

Con esta condición.....	**La instrucción se ejecuta sí...**
[A1]	$A1 = 0$
[!A1]	$A1! = 0$

• Instrucciones

Las instrucciones en código ensamblador, son directivas o mnemónicos

Directivas	Descripción
.sect"name"	Crea sección de información (datos o código)
.double value	Reserva dos palabras consecutivas de 32 bits (64 bits) en memoria y las llena con la representación de punto flotante IEEE de doble precisión (64 bits) del valor especificado
.float value	Reserva 32 bits en memoria y los llena con la representación de punto flotante IEEE de precisión simple, del valor especificado
.int value .long value .word value	Reserva 32 bits en memoria y los llena con el especificado valor
.short value .half value	Reserva 16 bits en memoria y los llena con el especificado valor
.byte value	Reserva 8 bits en memoria y los llena con el valor especificado

Tabla 4.1: Directivas del TMS320C6x

- *Directivas.* Son comandos para el ensamblador **asm6x** que controlan el proceso de ensamblado o que definen la estructura de los datos (constantes o variables), en el programa de lenguaje ensamblador. Todas las directivas del ensamblador comienzan con un punto, como muestra en el listado de la tabla 4.2

- *mnemónicos.* Son las instrucciones verdaderas del microprocesador que se encuentran en rutinas y ejecutan la operación del programa. Los mnemónicos comienzan a partir de la columna 2. A continuación se muestra la posición de la instrucción, en una línea de código en ensamblador.

label : *parallel bars* [*condition*] **instruction** *unit operands* ; *comments*

- Unidades Funcionales

El CPU 'C6000 contiene ocho unidades funcionales, que se muestran en la figura 4.3 y se describen en la tabla 4.2 Es opcional especificar la unidad funcional en el código. La especificación puede ser usada para documentar, que recurso (s) utiliza cada instrucción

label parallelbars [*condition*] *instruction* unit *operands comments*

- Operandos

Las instrucciones tienen los siguientes requerimientos para manejar los operandos del código ensamblador

- Todas las instrucciones requieren un operando destino

- La mayoría de las instrucciones requieren uno o dos operandos fuente

Unidad Funcional	Operaciones en punto fijo	Operaciones en punto flotante
Unidad .L(.L1, .L2)	Operaciones aritméticas y comparación de 32 y 40 bits. Cuenta de 0's o 1's mas a la izguierda, para 32 bits. Normalización de 32 y 40 bits. Operaciones lógicas de 32 bits	Operaciones aritméticas. Operaciones de conversión: DP–>SP, INT–>DP, INT–>SP
Unidad .S(.S1, .S2)	Operaciones aritméticas de 32 bits Corrimientos de 32/40 bits y operaciones campos de bits en 32 bits. Operaciones lógicas de 32 bits. Saltos. Generación de constantes. Transferencia de registros de/hacia registros (solamente .S2)	Comparación recíproca. Operaciones de raíz cuadrada. Operaciones de valor absoluto. Operaciones de Conversión SP a DP
Unidad .M(.M1,.M2)	Operaciónes de multiplicación de 16x16 bits	Operaciones de multiplicación de 32x32 bits. Operaciones de multiplicación de punto flotante
Unidad .D(.D1,.D2)	Sumas, restas y cálculos de direccionamiento circular de 32 bits Carga y almacenamiento con offset constante de 5 bits. Carga y almacenamiento con offset constante de 15 bits. (solo .D2)	Lectura de palabras dobles con offset constante de 5 bits.

Tabla 4.2: Unidades funcionales y las operaciones realizádas

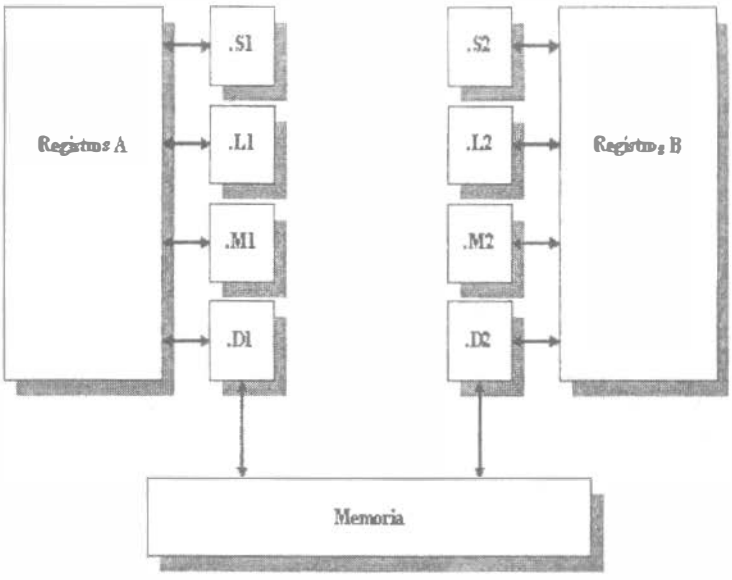

Figura 4.3: Unidades Funcionales del TMS320C6x.

ADD	.L1	A0,A1,A3
ADD	.L1X	A0,B1,A3

Tabla 4.3: Operandos en instrucciones

- El operando destino debe estar en el misma unidad de registros que el operando fuente
- Un operando fuente de cada archivo de registros por paquete de ejecución, puede llegar del archivo de registros opuesto del otro operando fuente.

Cuando un operando llega de otro archivo de registro, la unidad incluye una X, como se muestra en la tabla 4.3, indicando que la instrucción utiliza uno de los caminos cruzados. Las instrucciones del 'C6000 usan tres tipos de operandos para el acceso a datos:

- *Operandos de Registro.* Señalan al registro que contiene el dato
- *Operando Constante.* Especifica el dato dentro del código ensamblador
- *Operando Puntero.* Contiene la dirección del valor de datos

Únicamente las instrucciones de carga y almacenamiento requieren y usan operandos puntero para mover valores de datos entre memoria y un registro. Enseguida se muestra la posición de los operandos en una línea de código ensamblador.

label parallel bars [condition] instruction unit **operands**; *comments*

• Comentarios

Como ocurre con todos los lenguajes de programación, los comentarios proporcionan la documentación del código. A continuación se muestra la posición del comentario, en una línea de código ensamblador.

label parallelbars [condition] instruction unit operands; **comments**

Las siguientes son directrices, para usar comentarios en código en ensamblador

- Un comentario puede comenzar en cualquira columna, cuando se precede por un punto y coma (;)
- Un comentario debe comenzar en la primera columna, cuando se precede por un asterisco (*)
- Los comentarios no son indispensables, pero se recomienda su uso.

Ejemplo Código en ensamblador en paralelo del Producto Punto en Aritmetica de punto fijo.

```
        MVK   .S1   100,  A1        ;set up loop counter
||      ZERO  .L1   A7              ;zero out accumulator
LOOP:   LDH   .D1   *A4++,A2        ;load ai from memory
||      LDH   .D2   *B4++,B2        ;load bi from memory
        SUB   .S1   A1,1,A1         ;decrement loop counter
        B     .S2   LOOP            ;branch to loop
        NOP   2                     ;delay slots for LDH
        MPY   .M1X  A2,B2,A6        ;ai*bi--$>$A6
        NOP                         ;delay slots for MPY
        ADD   .L1   A6,A7,A7        ;sum += (ai*bi)
```

4.2 Codigo en C/C++

El compilador C6000 $C/C++$ traduce programas en ANSI C estándar a lenguaje ensamblador de C6000. Soporta todas las funciones de las librerias que conforman el estándar ANSI C. Las librerías incluyen funciones para entrada y salida estándar, manipulación de cadenas, manipulación de asignación dinámica de memoria, conversión de datos, temporización, funciones trigonométricas, exponenciales e hiperbólicas. Las funciones de manejo de señales, no están incluidas porque son específicas para cada sistema.

4.2.1 Tipos de datos

La tabla 4.4 muestra los diferentes tipos de datos que se pueden manejar con los dispositivos TMS320C6x, su tamaño respectivo, su representación y el rango de valores que alcanza cada tipo de datos.

Type	Size	Representation	Minimum	Maximum
char, signed char	8 bits	ASCII	-128	127
unsigned char	8 bits	ASCII	0	255
short	16 bits	Complemento a 2	-32767	32767
unsigned short	16 bits	Binario	0	65535
int, signed int.	32 bits	Complement a 2	-2 147 483 648	2 147 483 647
long, signed long	32 bits	Binario	0	4 294 967 295
unsigned int.	40 bits	Complement a 2	-549 755 813 888	549 755 813 887
unsigned long	40 bits	Binario	0	1 099 511 627 775
enum	32 bits	Complement a 2	-2 147 483 648	2 147 483 647
float	32 bits	IEEE 32-bit	1.175 494e-38	3.40 282 346e+38
double	64 bits	IEEE 64-bit	2.22 507 385e-308	1.79 769 313e+308
long double	64 bits	IEEE 64-bit	2.22 507 385e-308	1.79 769 313e+308
pointers, references pointer to data	32 bits	Binario	0	0xFFFFFFFF

Tabla 4.4: Tipos de datos

4.2.2 Ejemplo

- Llamando funciones en ensamblador en el código $C/C++$

El siguiente ejemplo muestra una función en C *main*, que llama a otra función en ensamblador *asmfun*. La función *asmfun* toma un solo argumento añade este a la variable gloval llamada *gvar* y regresa el resultado.

(a) C program

```
extern  ''C''  {
extern int asmfunc(int a);  /*declaracion de la funcion*/
                            /*como externa*/
int gvar = 4;               /*define la variable global*/
}
void main(){
            int i = 5;
            i = asmfunc(i)  /*llamada de funcion normal*/
}
```

(b) Assembly language program

```
.global _asmfunc
.global _gvar


_asmfunc:
LDW     *+b14(_gvar),A3
NOP     4
```

```
ADD     a3,a4,a3
STW     a3,*b14(_gvar)
MV      a3,a4
B       b3
NOP     5
```

- Producto en aritmética de punto flotante

```
float dotpl(const float a[], const float b[]
{
int i;
float sum = 0;
          for (i=0; i<512; i++)
               sum += a(i) * b(i);
return sum;
}
```

4.3 Entorno de Desarrollo Integrado del Code Composer Studio (IDE)

El IDE del Code Composer Studio esta diseñado para permitir editar, construir y depurar programas del DSP.

4.3.1 Características del editor de código de programas

El *Code Composer Studio* le permite editar código en C y en ensamblador. Puede verse el código fuente C con las instrucciones correspondientes en ensamblador, mostrando las sentencias de C después (utilizando de menú *View-> Mixed Source/ASM*), como se muestra en la figura 4.4

El editor integrado proporciona soporte a las siguientes actividades:

- Resalte palabras clave, comentarios y cadenas en color
- Marcar bloques de C en paréntesis y corchetes, encontrando el par o próximo paréntesis o corchete
- Niveles de sangrado
- Búsqueda y remplazo en uno o mas archivos
- Deshaciendo y rehaciendo múltiples acciones
- Obtención de ayuda sensible al contexto

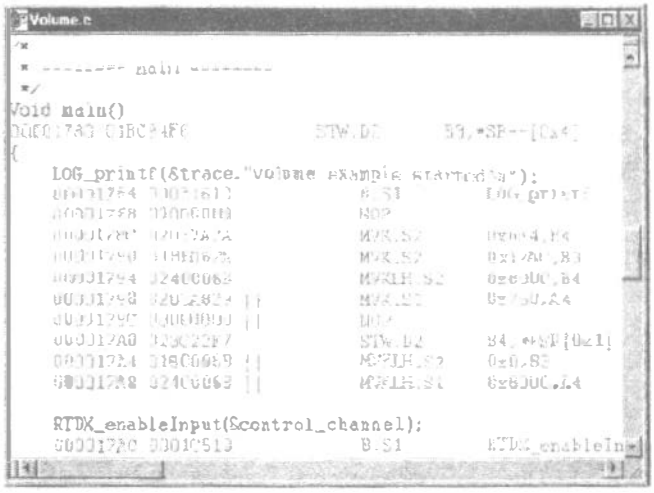

Figura 4.4: Código fuente en C, mezclado con código ensamblador.

4.3.2 Características de construcción de aplicaciones

Con el Code Composer Studio se puede crear un proyecto de trabajo, que es usado para construir la aplicación. Los archivos en el proyecto pueden ser archivos del código fuente en C, archivos en ensamblador, archivos objeto, librerías, archivos de comando del ligador y archivos de declaración *(include)*. En la siguiente figura se muestra un proyecto de trabajo CCS.

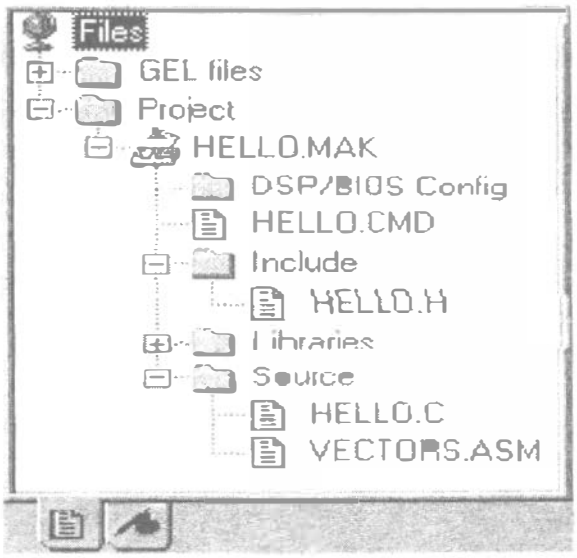

Figura 4.5: Proyecto de trabajo.

4.3.3 Características de depuración de aplicaciones

El Code Composer Studio provee soporte para las siguientes actividades de depuración:

- Establecer puntos de raptura
- Actualizar automáticamente las ventanas en los puntos de raptura
- Visualizar el valor de las variables
- Ver y editar registros y memoria
- Ver el stack de las llamadas a funciones
- Usar herramientas punto de prueba, para flujo de datos de y a la tarjeta
- Graficar las señales de la terjeta
- Generar estadísticas de ejecución
- Ver instrucciones desensambladas e instrucciones en C ejecutándose sobre la tarjeta
- Proporciona un lenguaje GEL. Este lenguaje permite añadir funciones al menú para optimizar las tarcas comúnes

Para obtener una mejor perspectiva de las herramientas de edición y depuración, además del entorno de desarrollo del Code Composer Studio, consultar en la documentación: *Code Composer Studio User's Guide de Texas Instrument.*

4.4 DSP/BIOS plug-ins

Los *plug-ins* de Code Composer Studio proporcionan con el DSP/BIOS, soporte para el análisis en tiempo real. Se pueden usar para visualmente: probar, señalar y monitorizar una aplicación DSP con el mínimo impacto en el performance. Las API's DSP/BIOS proporcionan las siguientes capacidades en tiempo real:

- **Mensajes del programa** *(program tracing)*: Despliega los eventos escritos en registros designados y refleja dinámicamente el control de flujo durante la ejecución del programa.
- **Monitoreo del performance** *(performance monitoring)*: Rastrea las estadísticas que reflejan el uso de los recursos, como la carga del procesador y los tiempos de procesos.
- **Archivos de Flujo** *(flow file)*: Liga archivos de datos en la PC, a objetos de Entrada/Salida en el programa del DSP.

4.4.1 Configuración del DSP/BIOS

Se pueden crear archivos de configuración utilizando el entorno del *Code Composer Studio*. Con ellos se definen objetos que son usados para las API's del DSP/BIOS. Estos archivos también simplifican el mapeo de memoria y el mapeo de los vectores, en las rutinas de atención de interrupción.

Cuando se abre un archivo de configuración del DSP/BIOS, el Code Composer Studio muestra un editor visual, que permite crear y establecer propiedades para objetos de tiempo real. Estos objetos son usados en las llamadas a las API's DSP/BIOS. También incluyen interrupciones por software, tuberías de I/O, mensajes de eventos (*logs*), etc. La figura 4.6 muestra el editor visual DSP/BIOS.

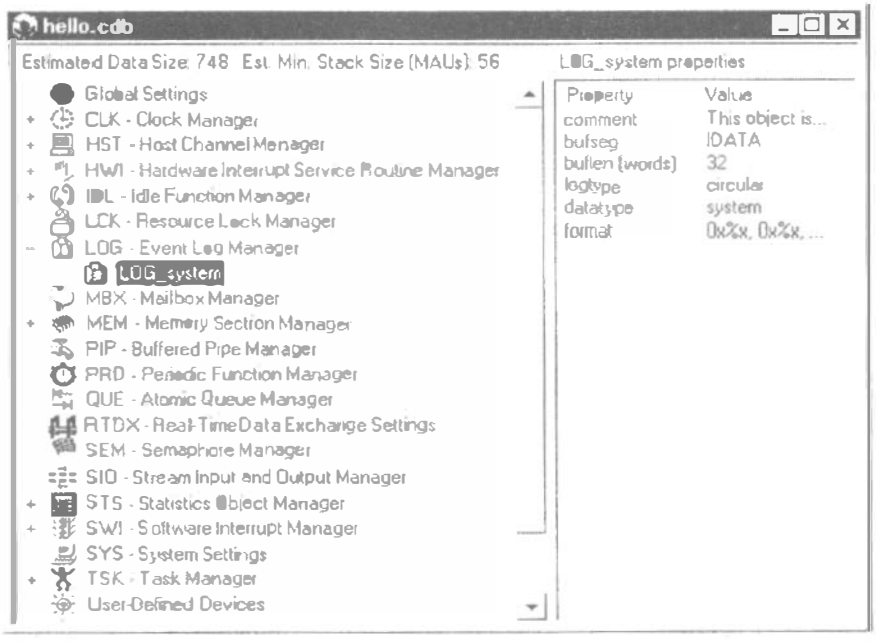

Figura 4.6: Ventana de configuración de los objetos del DSP/BIOS.

4.4.2 Módulos del DSP/BIOS

Las API's del DSP/BIOS están divididas en los siguientes módulos:

- **ATM:** Funciones atómicas que pueden ser usadas para manipular datos compartidos.
- **C62:** Este módulo proporciona funciones específicas del DSP, para manejo interrupciones.
- **CLK:** Este módulo controla el timer interno del DSP y proporciona un reloj lógico de 32 bits en tiempo real con alta resolución.
- **DEV:** Este módulo permite crear y usar sus propios controladores de dispositivos.

- **HST:** El módulo *host*, manejas este tipo de objetos de canal que permiten a una aplicación transmitir flujo de datos entre al DSP y un *host*. Los canales *host* son configurados estáticamente para entrada o salida.

- **HWI:** El módulo de interrupciones por hardware proporciona soporte para rutinas que atienden las interrupciones.

- **IDL:** Este módulo maneja funciones *idle*, que corren cuando no existe ninguna función de mayor prioridad ejecutando.

- **LCK:** El módulo *lock* maneja recursos globales compartidos y es usado para controlar el acceso a estos recursos, entre varias tareas que los disputen.

- **LOG:** Este módulo maneja objetos tipo *LOG*, que capturan eventos en tiempo real mientras el programa objeto se ejecuta. Se puede usar *logs* de sistema o se puede definir *logs* propios. Con el *Code Composer Studio* se puede visualizar estos mensajes *logs* mientras se ejecuta el programa.

- **MBX:** El módulo *mailbox* maneja objetos que pasan mensajes de una tarea a otra.

- **MEM:** El módulo de memoria permite especificar los segmentos de memoria requeridos.

- **PIP:** Este módulo maneja tuberías de datos *(pipe)*, que son usadas como flujos de *buffers* para entrada y salida de datos. Estas tuberías de datos proporcionan una estructura de datos consistente, para manejar entradas/salidas entre el DSP y otros dispositivos periféricos, en tiempo real.

- **PRD:** El módulo de manejo de funciones periódicas administra objetos de este tipo. Permite activar ejecuciones cíclicas de una función. La velocidad de ejecución para estos objetos puede ser controlada, por la frecuencia de reloj mantenida por el módulo CLK ó por llamadas regulares a la función *PRD_tick*

- **QUE:** Este módulo maneja estructuras de colas de datos.

- **RTDX:** Permite el intercambio de datos en tiempo real entre la PC y el DSP, y además analizar y desplegar los datos en la PC usando una automatización OLE cliente (esta puede estar programada en Visual C++, *Visual Basic, Excel, Matlab, LabView, etc*).

- **SEM** El módulo de semáforos permite sincronizar tareas y realizar exclución mutua.

- **SIO:** El módulo *stream* maneja objetos que proveen eficencia en tiempo real de dispositivos de I/O

- **STS:** Módulo de estadísticas, administra acumulación de estadísticas clave en tiempo real, mientras el programa se ejecuta.

- **SWI:** Este módulo administra las interrupciones por software. Estas interrupciones son procesos que tiene menor prioridad que las interrupciones por hardware y mayor prioridad que el módulo de tareas. Cuando un función anuncia a un objeto SWI, con una llamadade de API, el módulo SWI programa para ejecución la función correspondiente.

- **SYS:** Módulo de dispositivos de sistema proporcionan funciones de propósito general.
- **TRC:** El módulo *trace* envía mensajes a la ventana de depuración en tiempo real.
- **TSK:** Módulo de tareas (las tareas procesos con prioridad más que las interrupciones por software.)

4.5 Emulación de Hardware e Intercambio de Datos en Tiempo Real (RTDX)

Los DSP de Texas Intrument proporcionan emulación en el chip, habilitadas por el Code Composer Studio, para la ejecución de programas de control y monitoreo de la actividad del programa, en tiempo real. La comunicación con este soporte de emulación, ocurre a través de un enlace mejorado JTAG. Este enlace es una vía de baja intromisión de conexión, en cualquier sistema DSP. Una interfase de emulación, como TI XDS510, proporciona la conexión JTAG al lado del host.

El hardware de emulación, proporciona una variedad de capacidades:

- Iniciación, detención o reset del DSP
- Carga de código o datos dentro del DSP
- Examina registros o memoria del DSP
- Puntos de ruptura de instrucciones de hardware o dependencia de datos.
- Soporte de conteo, incluyendo perfiles exactos de ciclos
- Intercambio de datos en tiempo real (RTDX) entre el host y el DSP

El RTDX proporciona, en tiempo real, visibilidad continua en el trayecto de la operación de aplicaciones. El RTDX permite desarrollar sistemas, para transferir datos entre una computadora host y el DSP, sin parar la aplicación designada. Los datos pueden ser analizados y visualizados sobre el host usando cualquier automatización OLE. Esto acorta el tiempo de desarrollo, dando al diseñador una representación realista del trayecto de la operación, que realmente sigue el sistema.

El RTDX consta de ambos componentes: host y DSP. Una pequeña librería de software RTDX corre sobre el DSP. El diseñador de aplicaciones DSP, realiza llamadas de funciones a estas libreras APIs, para pasar datos a o del DSP. Las libreras usan emulación de hardware en el chip, para mover datos a o de la plataforma host, a través de una interfase JTAG. La transferencia de datos al host, ocurre en tiempo real, mientras la aplicación del DSP está corriendo. La figura 4.7 muestra en un diagrama de bloques, con los componentes que intervienen en un intercambio de datos, en tiempo real.
Sobre la plataforma host, una librera RTDX opera en conjunto con el Code Composer Studio. Las herramientas de despliegue y análisis, pueden comunicarse con el RTDX a través de una API COM, que obtiene o envía el dato de o a la aplicación DSP. Los diseñadores pueden usar paquetes estándar de software de despliegue, tales como LabView de National

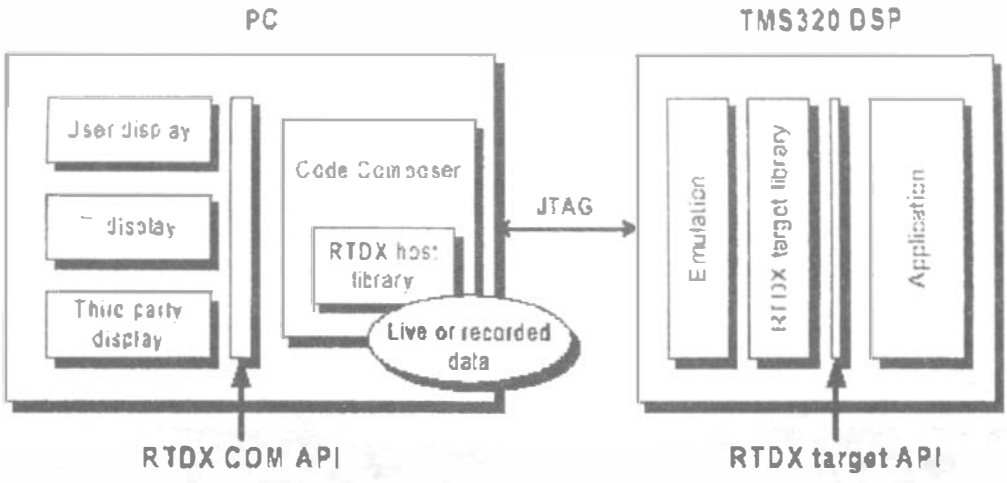

Figura 4.7: Diagrama de bloques de los componentes que intervienen en un RTDX.

Instrument, herramientas de graficación en tiempo real Quinn-Curtis o Microsoft Excel. Alternativamente, pueden desarrollar sus propias aplicaciones en Visual C++ o Visual Basic.

El RTDX también puede grabar datos en tiempo real y representar estos en análisis de tiempo no real. La siguiente figura muestra las características de LabView de National Intruments. Indica del lado izquierdo, la gráficas de una señal y debajo, la misma señal procesada con un filtro FIR. En la parte derecha de la figura muestra el espectro de las señales sin procesar y filtrada respectivamente.

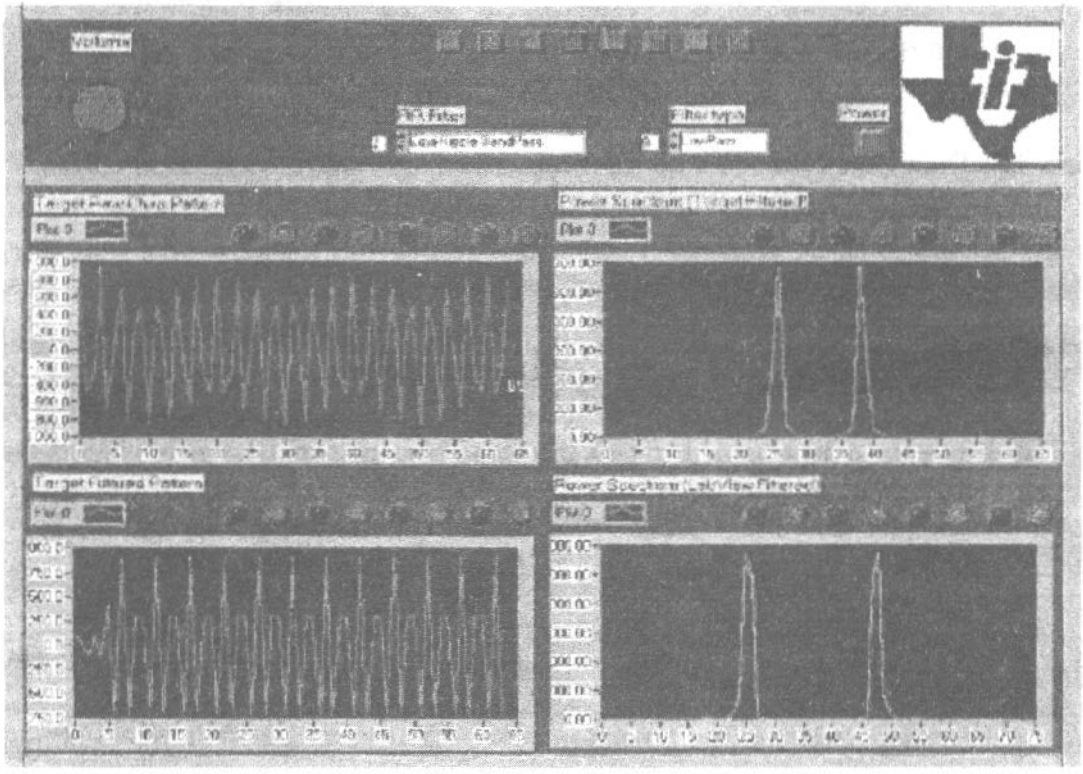

Figura 4.8: Utilización de RTDX mediante el paquete LabView

Capítulo 5

Desarrollo de un proyecto en el Code Composer Studio

La siguiente aplicación muestra un ejemplo para la realización de un filtro, con estructura de onda, en tiempo real. Utilizando el entorno de desarrollo de Code Studio Composer.

5.1 Extensiones de archivos

El Code Composer Studio (CCS), trabaja con los siguientes tipos de archivos:

- **project.mak:** Archivos de proyecto, que utiliza el CCS para definir un proyecto y construir un programa.
- **program.c:** Archivo fuente de programa en C.
- **program.asm:** Archivo fuente de programas en ensamblador.
- **filename.h:** Archivo de cabecera para programas en C, incluye archivos de cabeceras para módulos de la API del DSP/BIOS
- **filename.lib:** Archivos de librería.
- **proyect.cmd:** Archivos de comandos del Linker.
- **program.obj:** Archivo objeto compilado o ensamblado de su archivo fuente.
- **program.out:** Un programa ejecutable para el DSP.
- **project.wks:** Archivo de espacio de trabajo usado por el CCS, para almacenar información acerca de su entorno de trabajo.
- **program.cdb:** Archivo de base de datos para la configuración del DSP/BIOS creado por CCS. Este archivo es requerido para aplicaciones que usan las API's del DSP/BIOS. Los siguientes archivos son también generados cuando se crea un archivo de configuración del DSP/BIOS.
  1. **programcfg.cmd** Archivo comando de Linker.
  2. **programcfg.h62** Archivo de cabecera.
  3. **programcfg.s62** Archivo fuente en ensamblador.

5.2 Crear un nuevo proyecto para el filtro de onda

A continuación se dan los pasos para crear un proyecto utilizando el CCS. El proyecto usa el estándar C para generar el código.

1. Seleccionar del menú **Proyect->New**

2. Guardar el nuevo proyecto en una carpeta con el nombre FiltOnda.mak

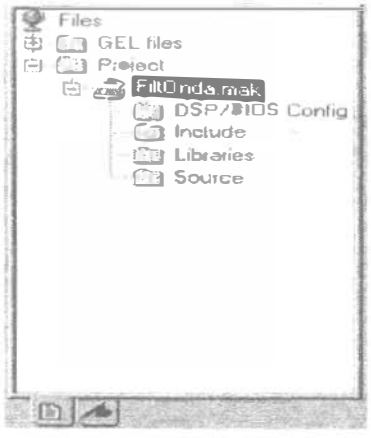

Figura 5.1: Ventana del proyecto con el archivo FiltOnda.mak

3. Realice una nueva configuración del DSP/BIOS

   - Seleccione del menú **File->New->DSP/BIOS Configuration**
   - Del cuadro de diálogo mostrado seleccione **dsk6711.cdb** (Configuración para el DSP C6711.
   - Salve el nuevo archivo de configuración como filtOnd_dsk67.cdb. Cuando realiza esta operación el CCS genera los siguientes archivos:

     $$filtOnd\_dsk67.ccf,\ filtOnd\_dsk67cfg.cmd,\ filtOnd\_dsk67cfg.s62\ y$$

     $$filtOnd\_dsk67cfg.h62$$

   - Agregue los siguientes archivos al proyecto:

     $$filtOnd\_dsk67.cdb,\ filtOnd\_dsk67cfg.cmd,$$

     con la opción **Proyect->Add File to Prtoyect.**

   - En la ventana de configuración del DSP/BIOS realice los siguientes pasos:

     (a) Modifique la interrupción del McBSP0 (Multichannel Buffered Serial port 0). En el módulo HWI extienda sus componentes, en la opción HWI_INT11 se da un clic con el botón derecho del ratón, se elige properties, aparece un cuadro diálogo que muestra las propiedades de la interrupción. En el campo

interrupt source verifique que tiene seleccionada la opción MCSP_0_Receive (recepción del puerto serie). En el campo **function** escriba _DSS_isr que es el nombre de la rutina o función que atiende a esa interrupción (el guión bajo es por convención de C en el manejo de sus funciones). Guarde el archivo con File->Save.

(b) Cree un nuevo objeto LOG. Extienda el módulo LOG, con el botón derecho del ratón de un clic en el manejador de eventos LOG y seleccione **insert LOG** (esto crea un nuevo objeto log), cambie el nombre de este objeto haciendo doble clic en este o con el botón derecho y seleccionando **Rename**. Reuombre al objeto como trace. Cambie las propiedades de este objeto, de un clic con el botón derecho del ratón y elija **Propertics**, cambie el tamaño del buffer *buflen* a 512. Guarde el archivo con File-- >Save

4. Escribir el código fuente en C que configura el McBSP0 (Multichannel Buffered Serial Port) y al convertidor A/D. El siguiente listado muestra el código del archivos **dss_dsk6211.c** que inicializa al McBSP0 para la recepción. Una vez escrito este código agregue este archivo al proyecto.

```
/*===dss_dsk6211.c===*/
#include <std.h>
#include <log.h>

#include <c6x.h>
#include "c6211dsk.h"

#define MCSP_RXINT_BIT 0x0800       /* define McBSP interrupt */

extern far LOG_Obj trace;           /* application printf() log */

/* function prototypes ... */
Void codec_init(void);
Void codec_error(Int id);
Void mcbsp0_init(void);
Uns mcbsp0_read(void);
Void mcbsp0_write(Uns out_data);
Void patchOldBoard(Void);

/*
 * ======== DSS_init ========
 */
Void DSS_init(void)
{

    mcbsp0_init();

    codec_init();
```

```
    /* Enable McBSP interrupt */
    IER |= MCSP_RXINT_BIT;
}

/*
 * ======== mcbsp0_init ========
 */
Void mcbsp0_init(void)
{
    /* set up McBSP0 */
    *(volatile Uns *)McBSP0_SPCR = 0x0; /*  reset serial port */
    *(volatile Uns *)McBSP0_PCR = 0x0;  /*  set pin control reg. */

    /* set RX and TX control registers to 16 bit data/frame */
    *(volatile Uns *)McBSP0_RCR = 0x10040;
    *(volatile Uns *)McBSP0_XCR = 0x10040;

    /* setup SP control register */
    *(volatile Uns *)McBSP0_SPCR = 0x00010001;
}

/*
 * ======== mcbsp0_write =======
 */
Void mcbsp0_write(Uns out_data)
{
    volatile Uns temp;

    temp = *(volatile Uns *)McBSP0_SPCR & 0x20000;
    while (temp == 0) {
        temp = *(volatile Uns *)McBSP0_SPCR & 0x20000;
    }

    *(volatile Uns *)McBSP0_DXR = out_data;
}

/*
 * ======== mcbsp0_read ========
 */
Uns mcbsp0_read(void)
{
    volatile Uns temp;

    temp = *(volatile Uns *)McBSP0_SPCR & 0x2;
    while (temp == 0) {
        temp = *(volatile Uns *)McBSP0_SPCR & 0x2;
    }
```

```
    temp = *(volatile Uns *)McBSP0_DRR;

    return (temp);
}

/*
 * ======== codec_init ========
 */
Void codec_init(void)
{
    volatile Uns temp;

    /* For more details on the setup for the TLC320AD535 Analog Interface,
       refer to Appendix A of SLAS202a.pdf. To use Code Composer Studio Help:
       Help->General Help, then in Contents tab double click TMS320C6211 DSK,
       double click Hardware, double click Analog Interface, then
       double click 'C6211 DSK Analog Interface Configuration, at the bottom
       is a link to the data manual.
     */

    /* set up control register 3 for S/W reset */
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
    mcbsp0_write(1);
    mcbsp0_read();
    mcbsp0_write(0x0386);
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();

    /* set up control register 3 for mic input */
    mcbsp0_write(0);
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
    mcbsp0_write(1);
    mcbsp0_read();
    mcbsp0_write(0x0306);
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();
```

```
/* read control register 3 to verify mic input */
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(1);
mcbsp0_read();
mcbsp0_write(0x2330);
temp = mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
if((temp & 0xff) != 0x06) {
    codec_error(3);
}

/* set up control register 4 to select Voice Channel Input 0db gain */
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(1);
mcbsp0_read();
mcbsp0_write(0x0400);
mcbsp0_read();
mcbsp0_write(0);
mcbsp0_read();

/* read and verify control register 4 */
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(1);
mcbsp0_read();
mcbsp0_write(0x2430);
temp = mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
mcbsp0_write(0x0);
mcbsp0_read();
if((temp & 0xff) != 0x00) {
    codec_error(4);
}

/* set up control register 5 to select Voice Channel Output 0db gain */
mcbsp0_write(0);
mcbsp0_read();
mcbsp0_write(0);
```

```
    mcbsp0_read();
    mcbsp0_write(1);
    mcbsp0_read();
    mcbsp0_write(0x0502);
    mcbsp0_read();
    mcbsp0_write(0);
    mcbsp0_read();

    /* read and verify control register 5 */
    mcbsp0_write(0);
    mcbsp0_read();
    mcbsp0_write(1);
    mcbsp0_read();
    mcbsp0_write(0x2530);
    temp = mcbsp0_read();
    mcbsp0_write(0x0);
    mcbsp0_read();
    mcbsp0_write(0x0);
    mcbsp0_read();
    if((temp & 0xfe) != 0x2) {
        codec_error(5);
    }
}

/*
 * ======== codec_error ========
 */
Void codec_error(id)
{
    LOG_error("Error setting up register %d", id);

    for (;;) {
        /* loop forever */
    }
}
```

5. Programar el filtro de onda de orden sinco que se muestra en la figura 5.2. Los parámetros de entrada son los coeficientes del filtro LC que se obtienen de los tablas del filtro Chebychev o Butterworth. El programa calcula los multiplicadores del filtro de onda A1, B2, A3, B4, A51 y A52. En el siguiente código se tiene la rutina que atiende la interrupción del McBSP0, en esta misma rutina se ha programado el filtro de onda. Despues de terminar de escribir el código es necesario incluir este archivo al proyecto.

```
/*
```

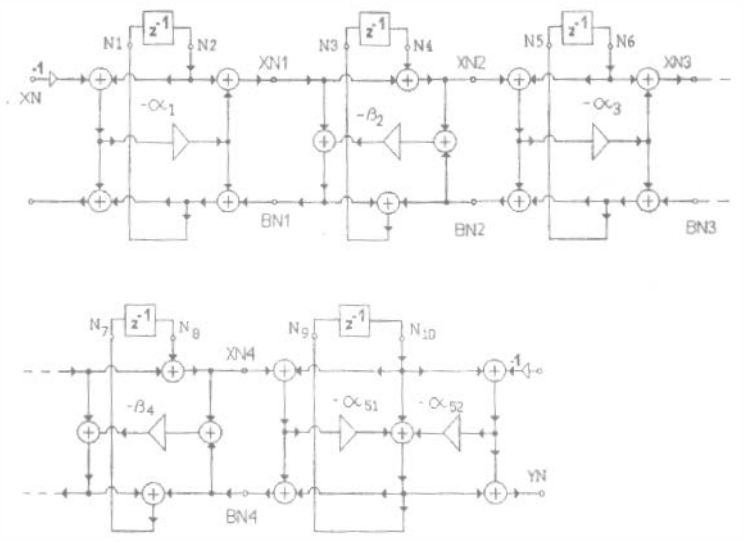

Figura 5.2: Filtro de onda de orden 5.

```
 *  ======== principal.c ========
 */

#include <std.h>
#include <tsk.h>
#include <c6x.h>
#include <log.h>

#include "c6211dsk.h"

#define MCSP_RXINT_BIT 0x0800       /* define McBSP interrupt */


/* function prototypes ... */
Void DSS_init(Void);      /* Initialize codec and serial port */
Void DSS_isr(Void);
Void codec_init(void);
Void codec_error(Int id);
Void mcbsp0_init(void);
Uns mcbsp0_read(void);
Void mcbsp0_write(Uns out_data);


/*
 *  ======== main ========
 */
Void main()
```

```
{

    DSS_init();  /* Esta funcion se definio en el archivo */
                 /* dss_dsk6211.c descrito previamente */
    return;
}

/*
a  = 0.0109 [dB]
as = 49.2 [dB]
fm = 2
*/


    DSS_init();

    return;
}


#define     C1  0.707229
#define     C3  1.434881
#define     C5  0.588442

#define     C2  0.077200
#define     L2  1.225747

#define     C4  0.220643
#define     L4  1.038317

#define     X2  (L2/(1 + L2*C2))
#define     X4  (L4/(1 + L4*C4))

#define     K2  ((L2*C2 - 1)/(L2*C2 + 1))
#define     K4  ((L4*C4 - 1)/(L4*C4 + 1))

#define     G1  1
#define     G2  (G1 + C1)
#define     R2  (1/G2)
#define     R3  (1/G2 + X2)
#define     G3  (1/R3)
#define     G4  (1/R3 + C3)
#define     R4  (1/G4)
#define     R5  (1/G4 + X4)
#define     G5  (1/R5)

#define     A1  (G1/(G1 + C1))
```

```
#define     B2   (R2/(R2 + X2))
#define     A3   (G3/(G3 + C3))
#define     B4   (R4/(R4 + X4))

#define     A51   (2*G5/(G5 + C5 + 1))
#define     A52   (2/(G5 + C5 + 1))

interrupt Void DSS_isr(Void)
{
short x;

static float
N1 = 0,   N2 = 0, N3 = 0, N4 = 0, N5 = 0, N6 = 0, N7 = 0,
N8 = 0,    N9 = 0, N10 = 0, N11 = 0, N12 = 0, N13 = 0, N14 = 0;

float
XN1 = 0, XN2 = 0, XN3 = 0, XN4 = 0, XN5 = 0,

YN = 0,

BN2 = 0, BN3 = 0, BN4 = 0, BN5 = 0;

x = mcbsp0_read();

  XN1 = (float)x;
  XN2 = -XN1*A1 - N2*A1 + N2;
  XN3 = XN2 + N6;
  XN4 = -XN3*A3 - N8*A3 + N8;
  XN5 = XN4 + N12;

    BN5 = XN5 - XN5*A51 + 2*N14 - N14*A51 -N14*A52;
    BN4 = XN4 -XN5*B4 -BN5*B4;
    BN3 = XN3 - XN3*A3 + BN4 +N8 - N8*A3;
    BN2 = XN2 - XN3*B2 - BN3*B2;

    N1 = -XN1*A1  - N2*A1 + BN2;
    N3 = BN2 + BN3;
    N5 = - N3*K2 + N6*K2 + N4;
    N7 = -XN3*A3 - N8*A3 + BN4;
    N9 = BN4 + BN5;
    N11 = - N9*K4 + N12*K4 + N10;
    N13 = -XN5*A51 + N14 - A51*N14 -A52*N14;

    YN = N13 + N14;

     N2 = N1;
    N4 = N3;
```

```
        N6 = N5;
        N8 = N7;
        N10 = N9;
        N12 = N11;
        N14 = N13;

        x = (short)(YN);
        x = x & 0xfffe;
        mcbsp0_write(x);

}
```

Sev Agregan los constantes al archivo dss_dsk6211. El archivo de cabecera c6211dsk.h tiene definido constantes para el DSP/BIOS, el archivo se tiene que incluir en el directorio de trabajo.

6. Compile, cargue el programa y ejecutelo. Ya teniendo los archivos guardados y agregados al proyecto es necesario compilarlos con la opción **Proyect->Rebuild All**, si el programa no tiene errores, ya puede cargar el programa a la memoria del DSP utilizando la opción **File->Load Program**, y por último para ejecutar el programa con la opción **Debug->Run**.

Capítulo 6

Ejemplos

6.1 El filtro canónico en paralelo

Ejemplo 1:

Escriba el programa principal para el filtro digital de cuatro orden en paralelo que se muestra en la figura 6.1. Los valores del filtro se muestran en la tabla.

	K=0.699474
b01=-0.44284	b02=-0.0693
b11=0.74359	b12=-0.12749
a11=0.747687	a12=-0.00808
a21=-0.30831	a22=-0.86777

```
/*
 *  ======== principal.c ========
 */

#include <std.h>
#include <tsk.h>
#include <c6x.h>
#include <log.h>

#include "c6211dsk.h"

#define MCSP_RXINT_BIT 0x0800      /* define McBSP interrupt */


/* function prototypes ... */
extern Void DSS_init(Void);     /* Initialize codec y puerto serial*/
extern Void DSS_isr(Void);
extern Void codec_init(void);
extern Void mcbsp0_init(void);
extern Uns mcbsp0_read(void);
```

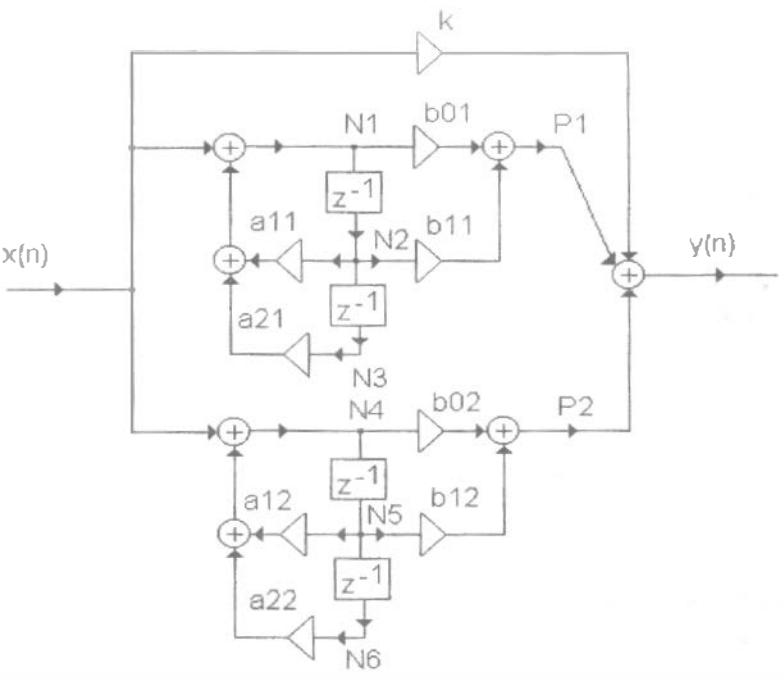

Figura 6.1: El filtro digital en la forma paralela.

```
extern Void mcbsp0_write(Uns out_data);


/*
 *  ======== main ========
 */
Void main()
{

    DSS_init();    /* Esta funcion se definio en el archivo */
                   /* dss_dsk6211.c, descrito previamente */

    return;
}


/*
N = 4
Rp = 2 [dB]
Rs = 15 [dB]
Wn = 0.5 -> 2 [KHz]
*/
```

```
#define K 0.699474

#define b01     -0.4428446
#define b11      0.743591
#define b02      -0.0693031
#define b12       -0.1274959
#define a11      0.447687
#define a21      -0.308310
#define a12      -0.00808
#define a22      -0.867723


interrupt Void interrupcionSerie(Void)
{
short x;
float y = 0;
float XN;
static float N[7] = {0,0,0,0,0,0,0};

x = mcbsp0_read();

XN = (float)x;

N[1] = XN + N[2]*a11 + N[3]*a12;
N[4] = XN + N[5]*a12 + N[6]*a22;
y =K*XN + N[1]*b01 + N[2]*b11+N[4]*b02+N[5]*b12;
N[3] = N[2];
N[2] = N[1];
N[6] = N[5];
N[5] = N[4];


    x = (short)(y);
    x = x & 0xfffe;
    mcbsp0_write(x);
}
```

6.2 El filtro canónico en cascada

Ejemplo 2:

Escriba el programa principal para el filtro digital de cuatro orden en cascada que se muestra en la figura 6.2. Los valores de los elementos se indican en la tabla siguiente.

K1=0.2932	
b01=1.0000	b02=1.0000
b11=0.0740	b12=0.8930
b21=1.0000	b22=1.0000
a11=0.3280	a12=0.0050
a21=-0.4601	a22=-0.9620

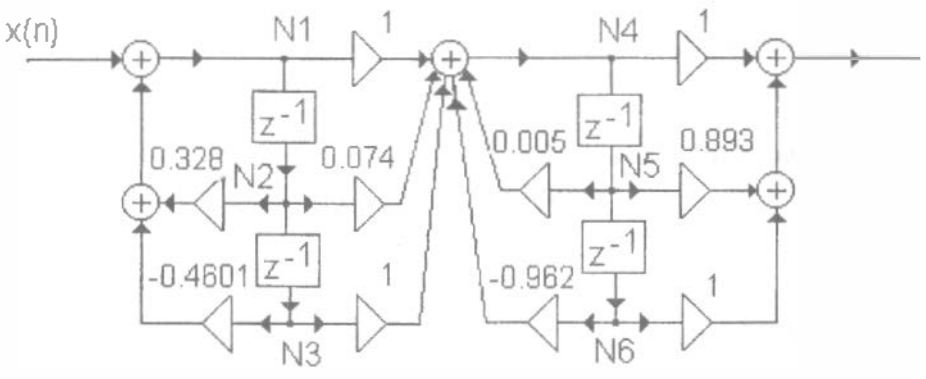

Figura 6.2: El filtro digital en la forma de cascada.

Solución:

Se recomienda primero analizar el filtro mediantze en MATLAB y si los resultados obtenidos son correctos escribir el fuente principal.c. El programa para la respuesta del filtro en la figura 6.2 es la siguiente:

```
n2=0;
n3=0;
n5=0;
n6=0;
k=0.2932;
b01=1;
b11=0.074;
b12=0.8932;
a11=0.3282;
a21=-0.4601;
a12=0.005;
a22=-0.96178;
xn=1;
for i=1:1:200
   n1=xn*k+n2*a11+n3*a21;
   n4=n1*b01+n2*b11+n3*b01+n5*a12+n6*a22;
   yn(i)=n4*b01+n5*b12+n6*b01;
   n3=n2;
   n2=n1;
```

```
    n6=n5;
    n5=n4;
    xn=0;
end
[h,w]=freqz(yn,1,200);
plot(w,20*log10(abs(h)))
zoom
```

La respuesta del filtro en casacada se muestra en la figura 6.3. De la figura 6.3 se ve, que los ecuaciones que realizan el filtro son correctos y se puede crear el programa principal.c.

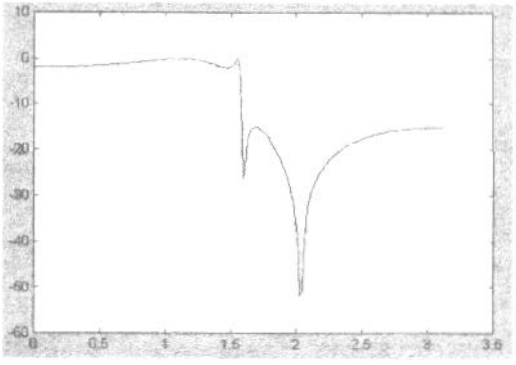

Figura 6.3: La reaspuesta del filtro a un impulso unitario.

```
/*
 *  ======== principal.c ========
 */

#include <std.h>
#include <tsk.h>
#include <c6x.h>
#include <log.h>

#include "c6211dsk.h"

#define MCSP_RXINT_BIT 0x0800       /* define McBSP interrupt */


/* function prototypes ... */
extern Void DSS_init(Void);      /* Initialize codec and serial port */
extern Void DSS_isr(Void);
extern Void codec_init(void);
extern Void mcbsp0_init(void);
extern Uns mcbsp0_read(void);
extern Void mcbsp0_write(Uns out_data);
```

```
/*
 *  ======== main ========
 */
Void main()
{

    DSS_init();    /* Esta funcion se definio en el archivo*/
                   /* dss_dsk6211.c, descrito previamente */

    return;
}


/*
N = 4
Rp = 2 [dB]
Rs = 15 [dB]
Wn = 0.5 -> 2 [KHz]
*/

const float K = 0.2932;
const float A[2][2] = { 0.3282,-0.4601, 0.005,-0.96178};

const float B[2][3] = { 1,0.079,1,1,0.8932, 1};


interrupt Void DSS_isr(Void)
{
short x;
static float x_tmp, N[2][3] = {0,0,0,0,0,0};
short n;

x = mcbsp0_read();
x_tmp = K*(float)(x);
for (n = 0; n<2; n++){
N[n][0] = x_tmp + A[n][0]*N[n][1] + A[n][1]*N[n][2];
x_tmp = B[n][0]*N[n][0] + B[n][1]*N[n][1] + B[n][2]*N[n][2];
N[n][2] = N[n][1];
N[n][1] = N[n][0];
}
x = (short)x_tmp;

    x = x & 0xfffe;
    mcbsp0_write(x);
}
```

6.3 El filtro digital Markel y Gray en cascada

Ejemplo 3:

Escriba el programa principal para el filtro digital Markel y Gray de cuatro orden en paralelo que se muestra en la figura 6.4. Los valores del filtro son escritos en la tabla siguiente:

K=0.2932	
a01=-0.2247	a02=0.0405
a11=0.4615	a12=0.96177
c01=0.63016	c02=0.0405
c11=0.4022	c12=0.8982
c21=1.0000	c22=1.0000

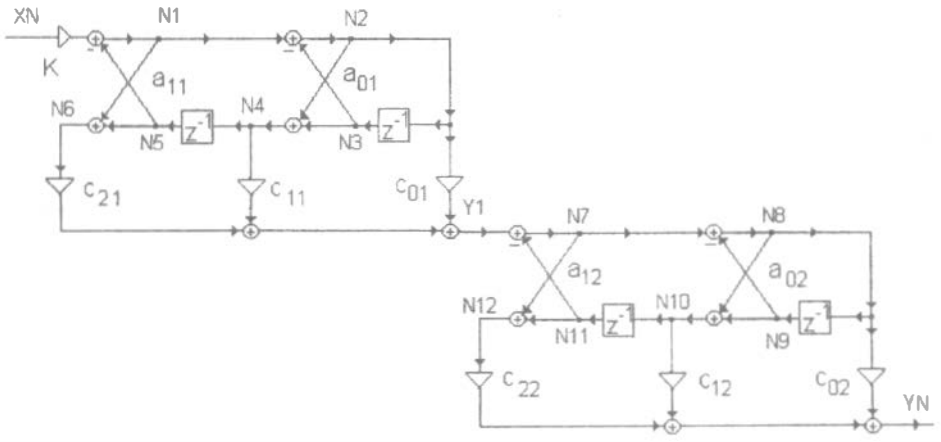

Figura 6.4: El filtro digital en la forma Markel y Gray.

Solución:

```
/*
 *  ======== principal.c ========
 */

#include <std.h>
#include <tsk.h>
#include <c6x.h>
#include <log.h>

#include "c6211dsk.h"

#define MCSP_RXINT_BIT 0x0800       /* define McBSP interrupt */
```

```
/* function prototypes ... */
extern Void DSS_init(Void);      /* Initialize codec and serial port */
extern Void DSS_isr(Void);
extern Void codec_init(void);
extern Void mcbsp0_init(void);
extern Uns mcbsp0_read(void);
extern Void mcbsp0_write(Uns out_data);


/*
 *  ======== main ========
 */
Void main()
{

    DSS_init();    /* Esta funcion se definio en el archivo */
                   /* dss_dsk6211.c descrito previamente */

    return;
}


/*
N = 4
Rp = 2 [dB]
Rs = 15 [dB]
Wn = 0.5 -> 2 [KHz]
*/

#define K 0.2932

#define a01 -0.2247
#define a11 0.46015
#define a02 -0.002548
#define a12 0.96177

#define c01 0.63016
#define c11 0.4022
#define c21 1
#define c02 0.040519
#define c12 0.8982
#define c22 1


interrupt Void DSS_isr(Void)
{
short x;
```

```
float y[2] = {0,0};
float XN;
static float N[13] = {0,0,0,0,0,0,0,0,0,0,0,0,0};

x = mcbsp0_read();

XN = (float)x;
N[1] = XN*K - N[5]*a11;
N[2] = N[1] - N[3]*a01;
N[4] = N[3] + N[2]*a01;
N[6] = N[5] + N[1]*a11;
y[1] = N[2]*c01 + N[4]*c11 + N[6]*c21;
N[3] = N[2];
N[5] = N[4];


N[7] = y[1] - N[11]*a12;
N[8] = N[7] - N[9]*a02;
N[10] = N[9] + N[8]*a02;
N[12] = N[11] + N[7]*a12;
y[0] = N[8]*c02 + N[10]*c12 + N[12]*c22;
N[9] = N[8];
N[11] = N[10];

    x = (short)(y[0]);
    x = x & 0xfffe;
    mcbsp0_write(x);
}
```

6.4 El filtro digital de estado en cascada

Ejemplo 4:

Escriba el programa principal para el filtro digital de cuatro orden en cascada que se muestra en la figura 6.5. Los elementos del filtro se muestran en la tabla siguiente:

d11=1.0000	d12=1.0000
a111=0.1641	a112=0.0025
a221=0.1641	a222=0.0025
a121=0.6582	a122=0.9807
a211=-0.6582	a212=-0.9807
c21=0.5488	c22=0.0218
c11=-0.8387	c12=-0.9486
b21=-0.5488	b22=-0.0218
b11=-0.8387	b12=-0.9486

Solución:

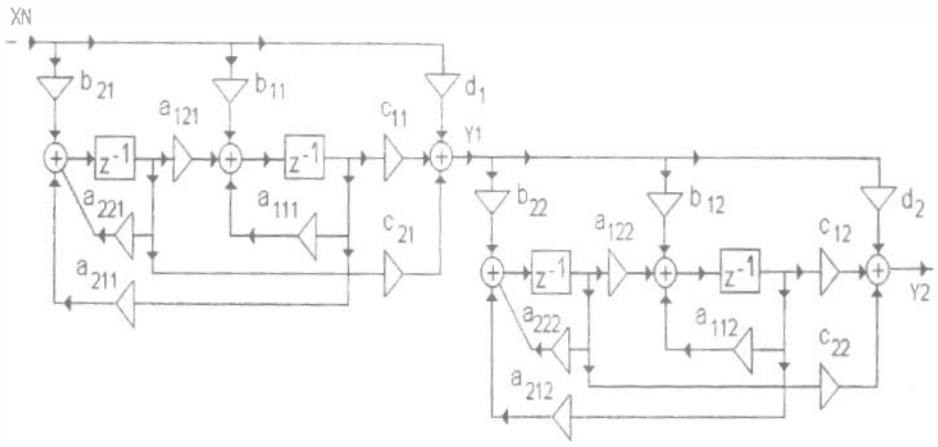

Figura 6.5: El filtro digital de estado en cascada.

```
/*
 *  ======== principal.c ========
 */

#include <std.h>
#include <tsk.h>
#include <c6x.h>
#include <log.h>

#include "c6211dsk.h"

#define MCSP_RXINT_BIT 0x0800       /* define McBSP interrupt */


/* function prototypes ... */
extern Void DSS_init(Void);      /* Initialize codec and serial port */
extern Void DSS_isr(Void);
extern Void codec_init(void);
extern Void mcbsp0_init(void);
extern Uns mcbsp0_read(void);
extern Void mcbsp0_write(Uns out_data);


/*
 *  ======== main ========
 */
Void main()
{

    DSS_init();
```

```
    return;
}


/*
N = 4
Rp = 2 [dB]
Rs = 15 [dB]
Wn = 0.5 -> 2 [KHz]
*/

#define  K 0.2932

#define  a111    0.1641
#define  a221    0.1641
#define  a121     0.6582
#define  a211    -0.6582
#define  c21 0.5488
#define  b21 -0.5488
#define  c11 -0.8387
#define b11      -0.8387
#define  d1      1

#define  a112  0.0025
#define  a222  0.0025
#define   a122   0.9807
#define  a212  -0.9807
#define   c22 0.0218
#define  b22 -0.0218
#define   c12 -0.9480
#define  b12 -0.9480
#define   d2
  1


interrupt Void DSS_isr(Void)
{
short x;
float y[3] = {0,0,0};
float XN;
static float N[9] = {0,0,0,0,0,0,0,0,0};

x = mcbsp0_read();

XN = (float)x;
```

```
N[1] = XN*b21 + N[2]*a221 + N[4]*a211;
N[3] = XN*b11 + N[2]*a121 + N[4]*a111;
y[1] = XN*d1 + N[2]*c21 + N[4]*c11;
N[2] = N[1];
N[4] = N[3];

N[5] = y[1]*b22 + N[6]*a222 + N[8]*a212;
N[7] = y[1]*b12 + N[6]*a122 + N[8]*a112;
y[2] = y[1]*d2 + N[6]*c22 + N[8]*c12;
N[6] = N[5];
N[8] = N[7];

y[0] = y[2]*K;

    x = (short)(y[0]);
    x = x & 0xfffe;
    mcbsp0_write(x);
}
```

Ejemplo 5:

Escriba el programa principal para el el filtro paso banda Cauer de Onda, si se conoce la estructura que se muestra en la figura 6.6. Los valores del filtro se presentan en la tabla siguiente:

K1=0.5431048	A1=ALFA1=0.227432
K2=-0.11659	B2=BETA2=0.490310
K3=0.870270	B3=BETA3=0.798227
K4=0.543100	A31=ALFA31=0.562578
	A32=ALFA32=0.326916

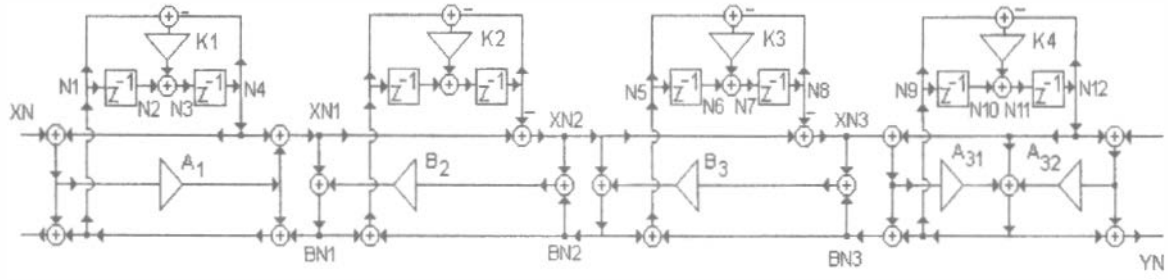

Figura 6.6: El filtro paso banda de Cauer en la forma de Onda.

```
/*
 *  ======= principal.c ========
```

```
 */

#include <std.h>
#include <tsk.h>
#include <c6x.h>
#include <log.h>
#include <math.h>


#include "c6211dsk.h"

#define MCSP_RXINT_BIT 0x0800      /* define McBSP interrupt */


/* function prototypes ... */
extern Void DSS_init(Void);     /* Initialize codec and serial port */
extern Void DSS_isr(Void);
extern Void codec_init(void);
extern Void mcbsp0_init(void);
extern Uns mcbsp0_read(void);
extern Void mcbsp0_write(Uns out_data);


/*
 *  ======== main ========
 */
Void main()
{

    DSS_init();    /* Esta funcion se definio en el archivo */
                   /* dss.dsk6211.c descrito previamente */

    return;
}


/*
El programa implementa el filtro de Onda paso banda de
sexto orden. Los coeficientes de los adaptadores se
calcularon de los elementos LC obtenidos de las tablas
Rudolf Saal Handbuch zum Filterenenrwurf C0350 theta 53
pagina 26.
*/



#define ALFA1 0.227432
```

```
#define BETA2 0.490310
#define BETA3 0.798227
#define ALFA31 0.562578
#define ALFA32 0.326916
#define K1 0.5431048
#define K2 -0.11659679
#define K3 0.8702786
#define K4 0.5431048

interrupt Void DSS_isr(Void)
{
short x;


float        XN=0, XN1=0, XN2=0, XN3=0, XN4=0, BN1=0, BN2=0,
             BN3=0, BN4=0, YN=0;
static float N1=0, N2=0, N3=0, N4=0, N5=0, N6=0, N7=0, N8=0,
             N9=0, N10=0, N11=0, N12=0, N13=0, N14=0, N15=0,
             N16=0;

x = mcbsp0_read();



   XN = (float)x;
   XN1=XN*ALFA1+N4*ALFA1-N4;
   XN2=XN1+N8;
   XN3=-XN2;
   XN4=XN3+N12;
   BN4=XN4-2*N16+ALFA31*N16+ALFA32*N16-XN4*ALFA31;
   BN3=XN3-BETA3*BN4-BETA3*XN4;
   BN2=BN3;
   BN1=XN1-BETA2*XN2-BETA2*BN2;
   N1=XN*ALFA1+ALFA1*N4+BN1;
   N3=N1*K1-N4*K1+N2;
   N5=BN1+BN2;
   N7=N5*K2-N8*K2+N6;
   N9=BN4+BN3;
   N11=N9*K3-N12*K3+N10;
   N13=-XN4*ALFA31-N16+N16*ALFA32+N16*ALFA31;
   N15=N13*K4-N16*K4+N14;
   YN=-ALFA31*XN4-2*N16+ALFA31*N16+ALFA32*N16;
   N2=N1;
   N4=N3;
   N6=N5;
   N8=N7;
   N10=N9;
```

```
    N12=N11;
    N14=N13;
    N16=N15;

    x=(short)(YN);
    x = x & 0xfffe;
    mcbsp0_write(x);
}
```

Ejemplo 6:

Escriba el programa principal para el el filtro paso banda Cauer de Onda, si se conoce la estructura que se muestra en la figura 6.7 y los elementos del filtro de la tabla siguiente. El filtro de Onda se calcula de los coeficientes LC obtenidos del libro Rudolf Saal "Handbuch zum Filterentwurf".

K1=-0.5431048	A1=ALFA1=0.258290
K2=0.651900	B2=BETA2=0.353206
K3=-0.96370	B3=BETA3=0.352410
K4=-0.5431	A4=ALFA4=0.101070
K5= 0.3896	B5=BETA5=0.384238
K6=-0.9258	B6=BETA6=0.399896
K7=-0.5431	A71=ALFA71=0.335519
	A72=ALFA72=0.457947

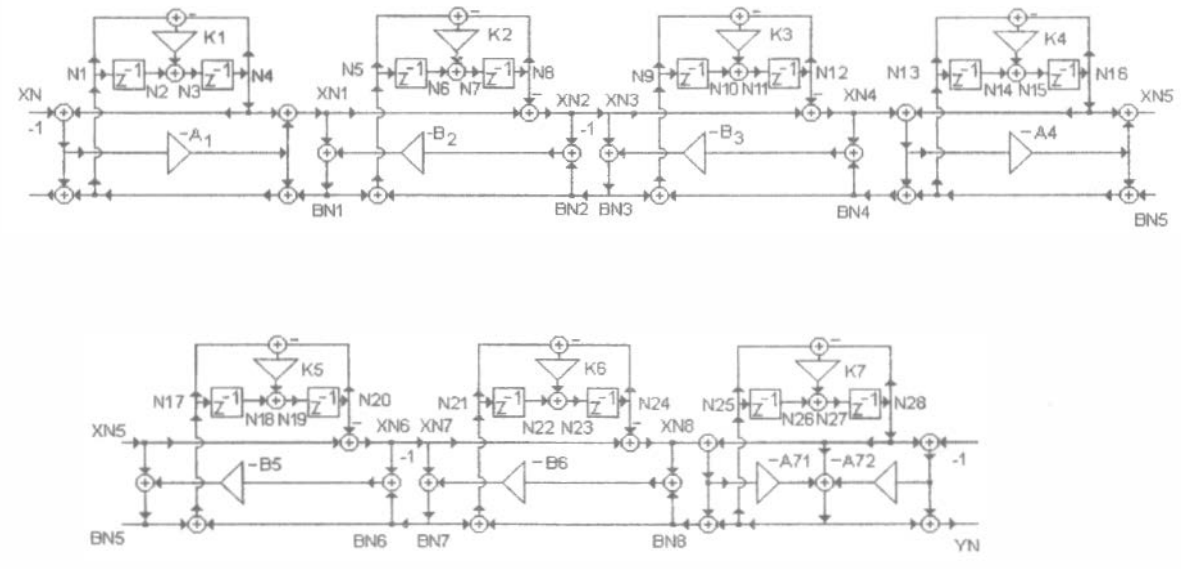

Figura 6.7: El filtro paso banda de Cauer n=10 en la forma de Onda.

```
/*
 *  ======== principal.c ========
 */

#include <std.h>
#include <tsk.h>
#include <c6x.h>
#include <log.h>
#include <math.h>

#include "c6211dsk.h"

#define MCSP_RXINT_BIT 0x0800       /* define McBSP interrupt */


/* function prototypes ... */
extern Void DSS_init(Void);      /* Initialize codec and serial port */
extern Void DSS_isr(Void);
extern Void codec_init(void);
extern Void mcbsp0_init(void);
extern Uns mcbsp0_read(void);
extern Void mcbsp0_write(Uns out_data);


/*
 *  ======== main ========
 */
Void main()
{

    DSS_init();

    return;
}


/*
N = 3
Los elementos del filtro LC n=5 son de las tablas
Rudolf Saal, Handbooch zum Filterentwurf...C0350
theta 53 pagina 26. El programa implementa el
filtro de Onda paso banda de ordemn N=10.
*/

#define   ALFA1 0.258290
#define   BETA2 0.353206
#define   BETA3 0.352410
```

```
#define   ALFA4 0.101070
#define   BETA5 0.384238
#define   BETA6 0.399896
#define   ALFA71 0.335514
#define   ALFA72 0.457947
#define   K1  -0.5431048
#define   K2   0.6519
#define   K3  -0.9637
#define   K4  -0.5431
#define   K5   0.3896
#define   K6  -0.9258
#define   K7  -0.5431


interrupt Void DSS_isr(Void)
{

short x;



float  XN=0, XN1=0, XN2=0, XN3=0, XN4=0, XN5=0, XN6=0,
       XN7=0, XN8=0, BN1=0, BN2=0, BN3=0, BN4=0,
       BN5=0, BN6=0, BN7=0, BN8=0, YN=0;

static float N1=0, N2=0, N3=0, N4=0, N5=0, N6=0, N7=0,
             N8=0, N9=0, N10=0, N11=0, N12=0, N13=0,
             N14=0, N15=0, N16=0, N17=0, N18=0, N20=0,
             N22=0, N24=0, N26=0, N28=0, N19=0, N21=0,
             N23=0, N25=0, N27=0;




 x = mcbsp0_read();

   XN = (float)x;
   XN1=XN*ALFA1+N4*ALFA1-N4;
   XN2=XN1+N8;
   XN3=-XN2;
   XN4=XN3+N12;
   XN5=-XN4*ALFA4+N16*ALFA4-N16;
   XN6=XN5+N20;
   XN7=-XN6;
   XN8=XN7+N24;
   BN8=XN8-XN8*ALFA71-2*N28+ALFA72*N28+ALFA71*N28;
   BN7=XN7-BETA6*BN8-BETA6*XN8;
```

```
    BN6=BN7;
    BN5=XN5-BETA5*XN6-BETA5*BN6;
    BN4=XN4-XN4*ALFA4+BN5;
    BN3=XN3-BN4*BETA3-XN4*BETA3;
    BN2=BN3;
    BN1=XN1-BN2*BETA2-XN2*BETA2;
    N1=XN*ALFA1+ALFA1*N4+BN1;
    N3=N1*K1-N4*K1+N2;
    N5=BN1+BN2;
    N7=N5*K2-N8*K2+N6;
    N9=BN4+BN3;
    N11=N9*K3-N12*K3+N10;
    N13=-XN4*ALFA4+N16*ALFA4+BN5;
    N15=N13*K4-N16*K4+N14;
    N17=BN5+BN6;
    N19=N17*K5-N20*K5+N18;
    N21=BN7+BN8;
    N23=N21*K6-N24*K6+N22;
    N25=-XN8*ALFA71-N28+N28*ALFA71+N28*ALFA72;
    N27=N25*K7-N28*K7+N26;
    YN=(-ALFA71*XN8-2*N28+ALFA71*N28+ALFA72*N28);
    N2=N1;
    N4=N3;
    N6=N5;
    N8=N7;
    N10=N9;
    N12=N11;
    N14=N13;
    N16=N15;
    N18=N17;
    N20=N19;
    N22=N21;
    N24=N23;
    N26=N25;
    N28=N27;

    x=(short)(YN);
    x=x & 0xfffe;
    mcbsp0_write(x);

}
```

6.5 El generador de la señal senoidal

Ejemplo 7:

Escriba el programa principal para el generador de senos para los parámetros $f_m = 8000$ Hz, frecuencia del corte $f_1 = 300$ H(z) si la función de transferencia es

$$H(z) = \frac{b_1 z^{-1}}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

El generador de senos se muestra en la figura 6.8. Los coeficientes de la función de transferencia son $b_1 = 0.233445$, $a_1 = 1,944739$, $a_2 = -1$.

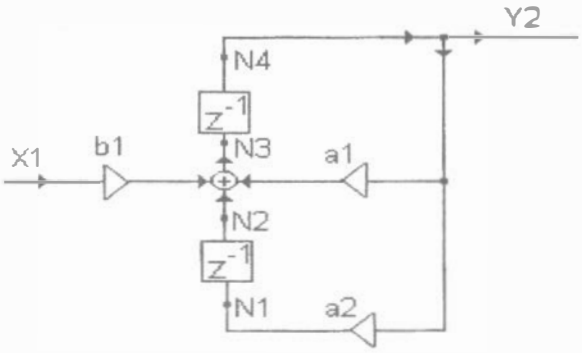

Figura 6.8: Generador de senos.

Solución:

```
/*
 *  ======== principal.c ========
 */

#include <std.h>
#include <tsk.h>
#include <c6x.h>
#include <log.h>

#include "c6211dsk.h"

#define MCSP_RXINT_BIT 0x0800       /* define McBSP interrupt */


/* function prototypes ... */
extern Void DSS_init(Void);      /* Initialize codec and serial port */
```

```
extern Void DSS_isr(Void);
extern Void codec_init(void);
extern Void mcbsp0_init(void);
extern Uns mcbsp0_read(void);
extern Void mcbsp0_write(Uns out_data);


/*
 *  ======== main ========
 */
Void main()
{

    DSS_init();    /* Esta funcion se definio en el archivo */
                   /* dss_dsk6211.c descrito previamente */

    return;
}


/*
Generador de senos
A1=1.944739
A2=-1
B1=0.233445
*/



#define a1 1.944739
#define a2 - 1
#define b1  0.233445


interrupt Void DSS_isr(Void)
{
short x;
float y[2] = {0,0};
static float n[5] = {0,0,0,0,0};

static float x1=1;

   y[0]=n[4];
   n[3]=x1*b1+n[4]*a1+n[2];
   n[1]=n[4]*a2;
   n[4]=n[3];
   n[2]=n[1];
```

```
    x1=0;


     x = (short)(y[0]);
     x = x & 0xfffe;
     mcbsp0_write(x);
}
```

6.6 El generador de la señal coseno

Ejemplo 8:

Escriba el programa principal para el generador de cosenos para los parámetros $f_m = 8000$ Hz, frecuencia delo corte $f_1 = 60$ H(z) si la función de transferencia es

$$H(z) = \frac{b_0 + b_1 z^{-1}}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

El generador de coseno se muestra en la figura 6.9. Los coeficientes de la función de transferencia son $b_1 = -0.99888987$, $a_1 = 1,9977797$, $a_2 = -1$ y $b_0 = 1$.

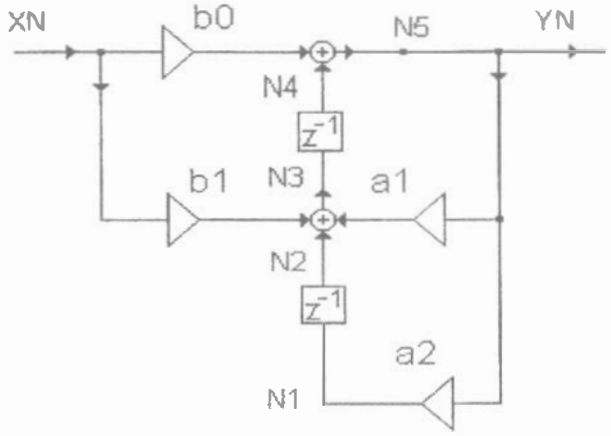

Figura 6.9: Generador de coseno.

Solución:

```
/*
 *  ======== principal.c ========
 */
```

```
#include <std.h>
#include <tsk.h>
#include <c6x.h>
#include <log.h>

#include "c6211dsk.h"

#define MCSP_RXINT_BIT 0x0800       /* define McBSP interrupt */


/* function prototypes ... */
extern Void DSS_init(Void);      /* Initialize codec and serial port */
extern Void DSS_isr(Void);
extern Void codec_init(void);
extern Void mcbsp0_init(void);
extern Uns mcbsp0_read(void);
extern Void mcbsp0_write(Uns out_data);


/*
 *  ======== main ========
 */
Void main()
{

    DSS_init();    /* Esta funcion se definio en el archivo */
                   /* dss_dsk6211.c descrito previamente */

    return;
}


/*
Generador de coseno
a1=1.9977794
a2=-1
b1=-0.99888987
b0=1
*/


#define a1 1.9977797
#define a2 -1
#define b0  1
#define b1 -0.99888987

interrupt Void DSS_isr(Void)
```

```
{


short x;
float yn;
static float n1=0,n2=0,n3=0,n4=0,n5=0,xn=1;

      n5=n4+xn*b0;
      n3=xn*b1+n5*a1+n2;
      n1=n5*a2;
      yn=60*n5;
      n4=n3;
      n2=n1;
      xn=0;
x=(short)(yn);
x=x & 0xfffe;
mcbsp0_write(x);

}
```

6.7 Filtro FIR adaptable

ejemplo 9

Escriba el programa para el filtro FIR adaptable de orden n=50. El filtro se muestra en la figura ??, tiene dos entradas, para la señal de ruido y la senal fuente con ruido. Para implementarlo el circuito en DSP TMS320C6711 es necesario eq uipar el segundo puerto serial con el convertidor A/D.
Primero se escribe en un archivo el programa que configura los puertos serial y los convertidores. El programa es el siguiente.

```
/*
 *  ======== dss_dsk6211.c ========
 */

#include <std.h>
#include <c6x.h>


/* Define McBSP0 Registers */
#define McBSP0_DRR      0x18c0000   /* Address of data receive reg.        */
#define McBSP0_DXR      0x18c0004   /* Address of data transmit reg.       */
#define McBSP0_SPCR     0x18c0008   /* Address of serial port contl. reg.  */
#define McBSP0_RCR      0x18c000C   /* Address of receive control reg.     */
```

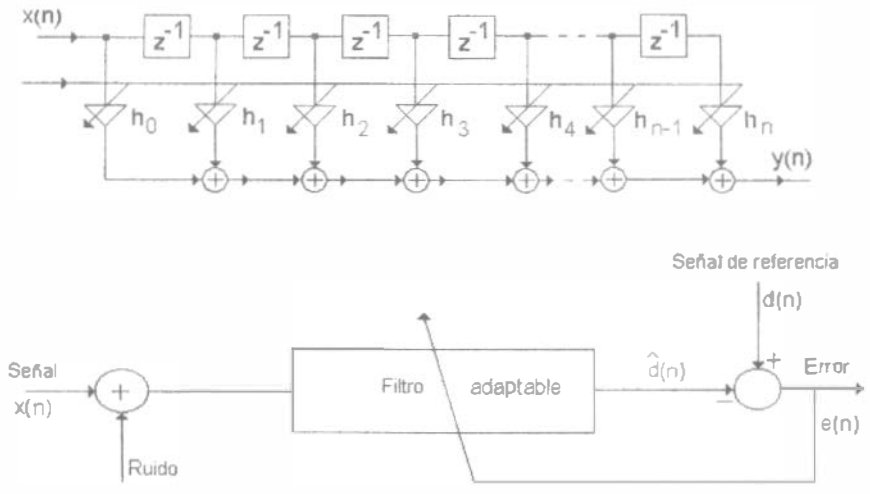

Figura 6.10: Filtro FIR adaptable y el sistema con un filtro adaptable.

```
#define McBSP0_XCR      0x18c0010   /* Address of transmit control reg.      */
#define McBSP0_SRGR     0x18c0014   /* Address of sample rate generator      */
#define McBSP0_MCR      0x18c0018   /* Address of multichannel reg.          */
#define McBSP0_RCER     0x18c001C   /* Address of receive channel enable.    */
#define McBSP0_XCER     0x18c0020   /* Address of transmit channel enable.   */
#define McBSP0_PCR      0x18c0024   /* Address of pin control reg.           */

/* Define McBSP1 Registers */
#define McBSP1_DRR      0x1900000   /* Address of data receive reg.          */
#define McBSP1_DXR      0x1900004   /* Address of data transmit reg.         */
#define McBSP1_SPCR     0x1900008   /* Address of serial port contl. reg.    */
#define McBSP1_RCR      0x190000C   /* Address of receive control reg.       */
#define McBSP1_XCR      0x1900010   /* Address of transmit control reg.      */
#define McBSP1_SRGR     0x1900014   /* Address of sample rate generator      */
#define McBSP1_MCR      0x1900018   /* Address of multichannel reg.          */
#define McBSP1_RCER     0x190001C   /* Address of receive channel enable.    */
#define McBSP1_XCER     0x1900020   /* Address of transmit channel enable.   */
#define McBSP1_PCR      0x1900024   /* Address of pin control reg.           */

#define MCSP_RXINT_BIT 0x0800       /* define McBSP interrupt */


/* funciones prototipo para el CODEC 0 y el McBSP0 */
Void codec0_init(void);
Void codec0_error(Int id);
Void mcbsp0_init(void);
Uns mcbsp0_read(void);
Void mcbsp0_write(Uns out_data);
```

```
/* funciones prototipo para el CODEC 1 y el McBSP1 */
Void codec1_init(void);
Void codec1_error(Int id);
Void mcbsp1_init(void);
Uns mcbsp1_read(void);
Void mcbsp1_write(Uns out_data);


/*
 * ======== DSS_init ========
 */
Void DSS_init(void)
{

    mcbsp0_init();
    codec0_init();

    mcbsp1_init();
    codec1_init();

    /* Enable McBSP interrupt */
    IER |= MCSP_RXINT_BIT;
}


/*****************************************************************
 *  *
 * Funciones para configurar el McBSP 0                  *
 *                                                              *
 *****************************************************************/

 /*
 * ======== mcbsp0_init ========
 */
Void mcbsp0_init(void)
{
    /* set up McBSP0 */
    *(volatile Uns *)McBSP0_SPCR = 0x0; /*  reset serial port */
    *(volatile Uns *)McBSP0_PCR = 0x0;  /*  set pin control reg. */

    /* set RX and TX control registers to 16 bit data/frame */
    *(volatile Uns *)McBSP0_RCR = 0x10040;
    *(volatile Uns *)McBSP0_XCR = 0x10040;

    /* setup SP control register */
    *(volatile Uns *)McBSP0_SPCR = 0x00010001;
```

```
}

/*
 * ======== mcbsp0_write ========
 */
Void mcbsp0_write(Uns out_data)
{
    volatile Uns temp;

    temp = *(volatile Uns *)McBSP0_SPCR & 0x20000;
    while (temp == 0) {
        temp = *(volatile Uns *)McBSP0_SPCR & 0x20000;
    }

    *(volatile Uns *)McBSP0_DXR = out_data;
}

/*
 * ======== mcbsp0_read ========
 */
Uns mcbsp0_read(void)
{
    volatile Uns temp;

    temp = *(volatile Uns *)McBSP0_SPCR & 0x2;
    while (temp == 0) {
        temp = *(volatile Uns *)McBSP0_SPCR & 0x2;
    }

    temp = *(volatile Uns *)McBSP0_DRR;

    return (temp);
}



/*****************************************************************
 *  *
 *   Funciones para configurar el McBSP 1                *
 *                                                               *
 *****************************************************************/
 /*
 * ======== mcbsp1_init ========
 */
Void mcbsp1_init(void)
{
    /* set up McBSP1 */
```

```
    *(volatile Uns *)McBSP1_SPCR = 0x0; /*  reset serial port */
    *(volatile Uns *)McBSP1_PCR = 0x0;  /*  set pin control reg. */

    /* set RX and TX control registers to 16 bit data/frame */
    *(volatile Uns *)McBSP1_RCR = 0x10040;
    *(volatile Uns *)McBSP1_XCR = 0x10040;

    /* setup SP control register */
    *(volatile Uns *)McBSP1_SPCR = 0x00010001;
}

/*
 * ======== mcbsp1_write ========
 */
Void mcbsp1_write(Uns out_data)
{
    volatile Uns temp;

    temp = *(volatile Uns *)McBSP1_SPCR & 0x20000;
    while (temp == 0) {
        temp = *(volatile Uns *)McBSP1_SPCR & 0x20000;
    }

    *(volatile Uns *)McBSP1_DXR = out_data;
}

/*
 * ======== mcbsp1_read ========
 */
Uns mcbsp1_read(void)
{
    volatile Uns temp;

    temp = *(volatile Uns *)McBSP1_SPCR & 0x2;
    while (temp == 0) {
        temp = *(volatile Uns *)McBSP1_SPCR & 0x2;
    }

    temp = *(volatile Uns *)McBSP1_DRR;

    return (temp);
}

/*****************************************************************
 *  *
 *   Las funciones para configura los CODEC0 y CODEC1,      *
 *   solo se muestran en prototipo, debido a que el cdigo   *
```

```
 *  depende de los codecs utilizados. El Starter Kit     *
 *   TMS320C6711 solo contiene un convertidor, la            *
 *   configuracin de este, ya se explico en programas    *
 *   previos                                                 *
 *  *
 ************************************************************/


/*
 * ======== codec0_init ========
 */
Void codec0_init(void)
{
   /* aqu se introducen el cdigo para
      la configuracin del CODEC0 */
}



/*
 * ======== codec1_init ========
 */
Void codec1_init(void)
{
   /* aqu se introducen el cdigo para
      la configuracin del CODEC1 */
}
```

Programa principal que se muestra enseguida se guarda en el archivo *principal.c*. El Starter Kit TMS320C6711, solo cuenta con un solo codec, es necesario integrar otro, para comprobar este algoritmo.

```
/*
 *  ======== principal.c ========
 */

#include <std.h>
#include <tsk.h>
#include <c6x.h>
#include <log.h>



/* function prototypes ... */
extern Void DSS_init(Void);      /* Initialize codec and serial port */
extern Void DSS_isr(Void);
extern Void mcbsp0_init(void);
```

```
extern Uns mcbsp0_read(void);
extern Void mcbsp0_write(Uns out_data);

extern Void mcbsp1_init(void);
extern Uns mcbsp1_read(void);
extern Void mcbsp1_write(Uns out_data);

/*
 *  ======== main ========
 */
Void main()
{

/* Se inicializa los puestos serie McBSP0 y McBSP1,
   as como los CODECS correspondientes

   NOTA: El Starter Kit TMS320C6711, solo cuenta
     con un solo codec, es necesario integrar
     otro, para comprobar este algoritmo*/


    DSS_init();

    return;
}
```



```
#define beta 2.5E-10 /* Velocidad de convergencia*/
#define N 50          /* # de coeficientes */


interrupt Void DSS_isr(Void)
{
short x, i;
float Y, E, D;
static float W[N+1]= {0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,
      0,0,0,0,0,0,0,0,0,0,0};

static float Delay[N+1] = {0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,
```

```
   0,0,0,0,0,0,0,0,0,0,0};


x = mcbsp0_read();  /* Seal de ruido*/
Delay[0] = (float)x;

x = mcbsp1_read();  /* Seal deseada + ruido d+n */
D = (float)x;

Y = 0;

for(i=0; i<N; i++){
Y += W[i]*Delay[i];
}

E = D-Y;

for(i=N; i>0; i--){
W[i] = W[i] + beta*E*Delay[i];
if (i != 0)
Delay[i] = Delay[i-1];
}


    x = (short)E;
    x = x & 0xfffe;
    mcbsp1_write(x);
}
```

Índice de Materias

Bibliografía

[1] Texas Instruments.: *TMS320C6000 CPU and Instruction SET - Reference Guide.* USA 1998

[2] Texas Instruments.: *TMS320C6000 Peripherals Reference Guide.* USA 1998

[3] Texas Instruments.: *Code Composer Studio - Users Guide.* USA 1998

[4] Texas Instruments.: *TMS320C6000 Assembly Language Tools - Users Guide.* USA 1998

[5] Texas Instruments.: *TMS320C6000 DSP/BIOS User's Guide.* USA 1998

[6] Texas Instruments.: *TMS320C6000 DSP/BIOS Application Programming Interface (API) Reference Guide.* USA 1998

[7] Texas Instruments.: *TMS320C6000 Programmers Guide.* USA 1998

[8] Texas Instruments.: *TLC320AD535C/I Data Manual.* USA 1998

Esta obra se terminó de imprimir
en diciembre de 2002
en el taller de imprenta del
Departamento de Publicaciones
de la Facultad de Ingeniería
Ciudad Universitaria, México, D.F.
C.P. 04510

Secretaría de Servicios Académicos

El tiraje consta de 300 ejemplares
más sobrantes de reposición