UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD INGENIERÍA

PRÁCTICAS DE LABORATORIO CON MICROPROCESADORES TMS320C30

Dr. BOHUMIL PŠENIČKA

Dr. SALVADOR LANDEROS AYALA

Dr. MILAN KARPF

DIVISIÓN DE INGENIERÍA ELÉCTRICA

DEPARTAMENTO DE INGENIERÍA EN TELECOMUNICACIONES

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 TMS320C30*, elaboradas por Bohumil Psenicka, Salvador Landeros y Milan Karpf.

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.

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.

Es importante señalar que este trabajo fue concluido durante la estancia que realizó el profesor Karpf Milan de la Universidad Técnica Checa, en el laboratorio de procesamiento digital de señales de la Facultad de Ingeniería en el invierno del 2000. Durante la estancia probó y comprobó los ejemplos que se ilustran en el libro.

En la primera parte se hace una presentación de la estructura básica del microcontrolador DSP TMS320C30, se explican características generales del módulo de evaluación (EVM) y se realiza una breve descripción del conjunto de instrucciones del DSP TMS320C30.

En la segunda parte, capítulo 4, se presentan programas elementales para que el alumno los ensamble y los corra en el módulo de evaluación, 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
Dr. Salvador Landeros Ayala

Profesores de la Facultad de Ingeniería de la UNAM

Contentenido

Índice abreviaturas

ACC- Acumulador.
ACCH- Parte alta del acumulador.
ACCL- Parte baja del acumulador.
ARAU- Unidad aritmético lógica de los registros auxiliares.
ARB- *Buffer* de los registros auxiliares.
ARP- Apuntador de los registreos auxiliares.
ARx- Registros auxiliares x=0,1,...,7.
CALU- Unidad central aritmético lógica.
CNFD- Configura bloque B0 en la memoria de datos.
CNFP- Configura bloque B0 en la memoria de programa.
COFF- El archivo ejecutable para EVM de formato común.
DAB- *Bus* de datos.
DP- Apuntador de página.
DR- Pin del puerto de serie para recepción.
DRR- Registro para recepción en la forma serial.
DSP- Procesamiento digital de señales.
DSK- Archivo ejecutable para starter-kit.
DX- Pin del puerto de serie para transmisión
DXR- Registro para transmisión en la forma serial.
EVM- Módulo de evaluación.
FIR- Filtros con la respuesta finita.
GREG- Registro de memoria global.
IE- Registro de habilitación.
IF- Registro de banderas.
IFR- Registro de bandera para interrupción.
IMR- Registro de interrupción protejido.
IIR- Filtros con la respuesta infinita.
IR0- Registro índice 0.
IR1- Registro índice 1.
MIPS- Millones de instrucciones por segundo.
MUX- Multiplexor.
PAB- *Bus* de programa.
PFC- Registro para direccionar B0.
PC- Contador de programa.
PR- Registro P.
Rx- Registro de precisión extendida x=0,1,...,7.
RE- Registro de dir. final de repetición.
RPTC- Registro de repetición.
RS- Registro de dir. inicial de repetición.
RSR- Registro de corrimiento para recepción serial.
ST0- Registro de estado 0.
ST1- Registro de estado 1.
TBC- Controlador de canal de pruebas.
TR- Registro T.
VLSI- Muy alta escala de integración.

Capítulo 1

Microprocesador TMS320C30

1.1 Introducción

El microprocesador TMS320C30 forma parte de la tercera generación de la familia TMS320 de procesadores digitales de señales (DSP) desarrollados con tecnología 1μm CMOS, capaces de manejar operaciones con punto flotante de 32 bits. Se fabrica en un encapsulado de tipo PGA (*Pin Grid Array*) de 181 pins.

El ciclo de reloj es de 60 ns, palabra de datos y programa de 32 bits, con *bus* de direcciones de 24 bits, lo que resulta en un espacio de memoria de 2^{24} = 16.7 megapalabras. Ejecuta instrucciones a una tasa de 33.3 MFLOPS [1] y 16.7 MIPS [2]. Funciona con un reloj externo de 66 MHz.

Otra característica importante de un DSP con respecto a los demás microprocesadores es la multiplicación en paralelo y carga al *acumulador*, todo esto en un solo ciclo (instrucción tipo MAC, *Multiply and Accumulate*).

Este microprocesador posee un conjunto de registros, algunos de propósito general denominados archivos de registros, que son utilizados por el CPU para llevar el control de sus operaciones. Tiene un depósito de 64 palabras de longitud empleado por la memoria de programa. Para realizar operaciones aritméticas, usa dos unidades aritméticas que trabajan en paralelo, cada una de las cuales posee un registro auxiliar asociado (ARAU0 y ARAU1). Los bloques de memoria internos son de acceso dual (pueden accesar dos operandos en un ciclo de máquina). Existe un canal destinado para DMA (acceso directo a memoria) que disminuye la carga al CPU, el cual soporta operaciones de entrada/salida concurrente.

En cuanto a los periféricos desarrollados dentro del circuito integrado, se consideran dos temporizadores y dos puertos seriales independientes. Debido a su arquitectura basada en registros, facilitan la implantación de compiladores de alto nivel, como es el caso del lenguaje C. El bloque de memoria ROM es de 4k-palabras y existen además dos bloques de memoria RAM de 1k. La unidad aritmética lógica y el multiplicador son de 40 bits para punto flotante y de 32 bits para números enteros. El CPU tiene un registro de corrimiento de hasta 32 bits, ocho registros o acumuladores de precisión extendida, así como dos generadores de direcciones con ocho registros auxiliares. Para las unidades aritméticas se tienen dos registros

[1] MFLOPS = millones de operaciones de punto flotante por segundo.

[2] MIPS = millones de instrucciones por segundo.

auxiliares. Puede ejecutar instrucciones de dos o tres operandos. Posee dos banderas externas de propósito general y cuatro interrupciones externas. El controlador de DMA se utiliza para leer o escribir en la memoria sin que se interrumpa al CPU.

1.2 Arquitectura

La arquitectura interna del TMS320C30 se estructuró con la finalidad de poder ejecutar grandes volúmenes de operaciones aritméticas, utilizando tanto hardware como software. Posee una extensa memoria interna y un alto grado de paralelismo en las operaciones, lográndose así una reducción significativa en el tiempo de ejecución de un bloque de instrucciones.

1.3 Unidad de procesamiento central (CPU)

La administración de los recursos del CPU se realiza a través de registros agrupados en un archivo de registros que contiene los siguientes componentes:

1. Multiplicador de enteros y números de punto flotante.
2. Unidad aritmética lógica (ALU), la cual ejecuta operaciones de punto flotante, de enteros y operaciones lógicas.
3. Registro de corrimiento de hasta 32 bits.
4. *Buses* internos (CPU1/CPU2 y REG1/REG2).
5. Unidades aritméticas con registros auxiliares asociados (ARAU).
6. Conjunto de 28 registros con diversos propósitos (register file).

Multiplicador para enteros y de punto flotante

Ejecuta instrucciones de duración de un ciclo de máquina de la siguiente manera:

- Multiplicación entera de 24 bits con resultado de 32 bits.
- Multiplicación de punto flotante de 32 bits con resultado de 40 bits.

Unidad aritmética lógica (ALU)

Aquí se ejecutan operaciones de duración de un ciclo de máquina para datos enteros y lógicos de 32 bits, y datos de punto flotante de 40 bits. La extensión de la palabra de datos que resulta tiene la misma extensión que los operandos.

Registro de corrimiento de 32 bits

Es utilizado para realizar desplazamientos de hasta 32 bits hacia la izquierda o hacia la derecha en un solo ciclo.

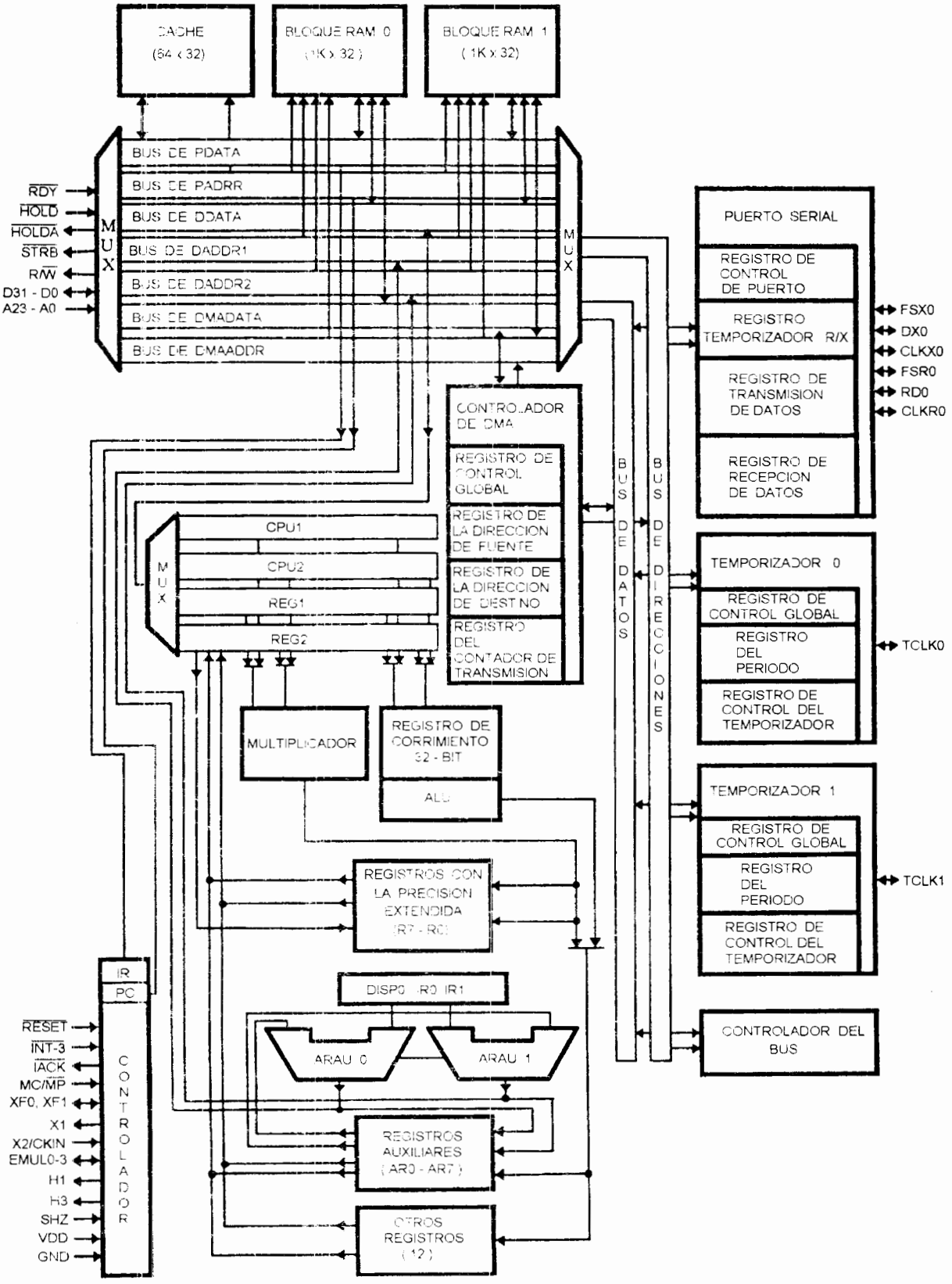

Figura 1.1: Arquitectura del microprocesador TMS320C30

Buses internos CPU1/CPU2 y REG1/REG2

La concepción separada de estos *buses* permite extraer de dos operandos de memoria y dos operandos desde los registros para que se realicen multiplicaciones paralelas, así como sumas y restas de cuatro operandos en un solo ciclo de máquina.

Unidades aritméticas con registros auxiliares asociados ARAU0 y ARAU1

Es posible generar dos direcciones en un solo ciclo que opera en paralelo con el multiplicador y la ALU, esto permite usar un modo de direccionamiento indexado, circular y de inversión de bit.

1.3.1 Archivo de registros del CPU

Éstos se pueden utilizar como registros de propósito general, aunque también cada uno de ellos realiza una función específica; por ejemplo, se tienen ocho registros de precisión extendida y ocho registros auxiliares, entre otros, que se presentan a continuación.

Registros de precisión extendida R0-R7

Almacenan operandos y realizan operaciones con enteros de 32 bits y números de punto flotante de 40 bits. Se asume que para operaciones con enteros, usan los 32 bits menos significativos y en operaciones de punto flotante, 40 bits.

Registros auxiliares AR0-AR7

Se accesan por el CPU y por la ARAU. Su función principal es la generación de direcciones de 24 bits, pero se pueden usar también como contadores de cuenta cerrada o como registros de propósito general.

Apuntador a página de datos DP

Registro de 32 bits, donde los ocho menos significativos se utilizan para direccionamiento directo como un apuntador a la página de datos direccionada. Las páginas son de 64 k-palabras, teniéndose un total de 256 páginas.

Registros índice IR0, IR1

Contienen el valor usado por la correspondiente ARAU para calcular una dirección indexada.

Registros de tamaño de bloque BK

Son utilizados por la ARAU para especificar el tamaño del bloque en el direccionamiento circular.

Nombre del registro	*Función*
R0	Registro de precisión extendida 0
R1	Registro de precisión extendida 1
R2	Registro de precisión extendida 2
R3	Registro de precisión extendida 3
R4	Registro de precisión extendida 4
R5	Registro de precisión extendida 5
R6	Registro de precisión extendida 6
R7	Registro de precisión extendida 7
AR0	Registro auxiliar 0
AR1	Registro auxiliar 1
AR2	Registro auxiliar 2
AR3	Registro auxiliar 3
AR4	Registro auxiliar 4
AR5	Registro auxiliar 5
AR6	Registro auxiliar 6
AR7	Registro auxiliar 7
DP	Apuntador a página de datos
IR0	Registro índice 0
IR1	Registro índice 1
BK	Registro de tamaño de bloque
SP	Apuntador al stack
ST	Registro de estado
IE	Registro de habilitación de int. CPU/DMA
IF	Registro de banderas de int. del CPU
IOF	Registro de banderas de E/S
RS	Registro de dir. inicial de repetición
RE	Registro de dir. final de repetición
RC	Contador de repetición
PC	Contador de programa

Tabla 1.1: Registros del TMS320C30

Apuntador al paquete del sistema SP

Registro de 32 bits que contiene la dirección guardada en la localidad más alta del paquete. Este registro siempre apunta al último elemento que entra. El paquete es intervenido por interrupciones, llamadas y retornos de subrutinas, y por las instrucciones PUSH y POP.

Registro de estado ST

Guarda información global sobre el estado del CPU, es decir, contiene las banderas que indican si el resultado de una operación fue cero o negativo. Mediante software, es posible salvar y restaurar este registro.

Registro de habilitación de interrupciones de CPU/DMA IE

Registro de 32 bits donde un "1"en la posición adecuada habilita la interrupción correspondiente. Las interrupciones para DMA están en los bits 26 a 16 de este registro. Un 0 deshabilita la interrupción.

Registro de banderas de interrupciones del CPU IF

Registro de 32 bits en el cual, de la misma manera que el anterior, con un "1"habilita y con un "0" deshabilita la interrupción correspondiente.

Registro de banderas de E/S IOF

Controla los pines externos XF0 y XF1, los cuales pueden ser configurados como entradas o salidas.

Registro de dirección inicial de repetición RS

Cuando el procesador se encuentra operando en el modo de repetición, este registro contiene la dirección inicial del bloque de instrucciones que van a ser repetidas.

Registro de direccionamiento final de repetición RE

Contiene la última dirección del bloque de programa que va a ser repetido.

Contador de repetición RC

Registro de 32 bits que especifica el número de veces que un bloque de código se repetirá.

Contador de programa PC

Registro de 32 bits que contiene la dirección de la siguiente instrucción que será accesada por el CPU. Aunque el PC no es parte de los registros del CPU, sí es un registro que puede ser modificado por instrucciones que alteran el flujo del programa.

1.4 Organización de la memoria

El espacio total de memoria del procesador definido por un *bus* de direcciones de 24 bits da por resultado 16M-palabras. El programa, datos y espacio de E/S, están contenidos dentro de este espacio de 16M-palabras, lo cual permite que tablas, coeficientes, código de programa o datos puedan estar guardados ya sea en RAM o en ROM.

Memorias RAM, ROM y Depósito

El procesador tiene dos bloques de memoria RAM (Bloque 0 y Bloque 1), cada uno de ellos es de 1k-palabras, y un bloque de ROM de 4k-palabras. Cada bloque permite dos accesos en un solo ciclo de máquina.

Dado que los *buses* de programa (PADDR, PDATA), de datos (DADDR1, DADDR2) y de DMA (DMAADDR, DMADATA) están separados, pueden realizarse paralelamente lecturas y escrituras de datos, acceso al código del programa y operaciones de DMA. Existe un depósito de instrucciones de 64 palabras que almacena las secciones de código más utilizadas, reduciendo así el número necesario de accesos.

Mapa de memoria

La configuración del mapa de memoria depende del estado del pin $MC/\overline{MP}$.

Modo de microcomputadora ($MC/\overline{MP} = 1$). El espacio en ROM de 4k-palabras está mapeado de las localidades 0h a la 0FFFh; las primeras 192 localidades se destinan a vectores de interrupción, vectores de "trampa", y a un espacio reservado. Las localidades de la 1000h a la 7FFFFFh son para memoria externa, cuando está activo el pin $\overline{STRB}$.

En el modo de microprocesador ($MC/\overline{MP} = 0$) no existe el bloque de 4k de ROM. Las primeras 192 localidades están destinadas a interrupciones, trampas y algunas localidades reservadas. En este espacio se accesa a memoria externa cuando está activo el pin $\overline{STRB}$. A partir de la localidad C0h, el espacio de memoria restante, corresponde a memoria externa.

Para ambos modos de operación, microcomputadora y microprocesador, las localidades 800000h a la 801FFFh están mapeadas al *bus* de expansión cuando se activa el pin $\overline{MSTRB}$. Los registros de control de los periféricos se encuentran entre las localidades 808000h a la 8097FFh para ambos modos de operación. El bloque 0 de RAM está mapeado de la 809800h a la 809BFFh y el bloque 1 de la 809C00h a la 809FFFh, y las localidades 80A000h a la 0FFFFFFh están destinadas a memoria externa.

Modos de direccionamiento

Este microprocesador soporta seis tipos de direccionamiento:

- *Por registro:* el operando es siempre un registro del CPU, pudiendo ser un registro de precisión extendida.
- *Corto-inmediato:* el operando es un valor inmediato de 16 bits.
- *Largo-inmediato:* el operando es un valor inmediato de 24 bits.

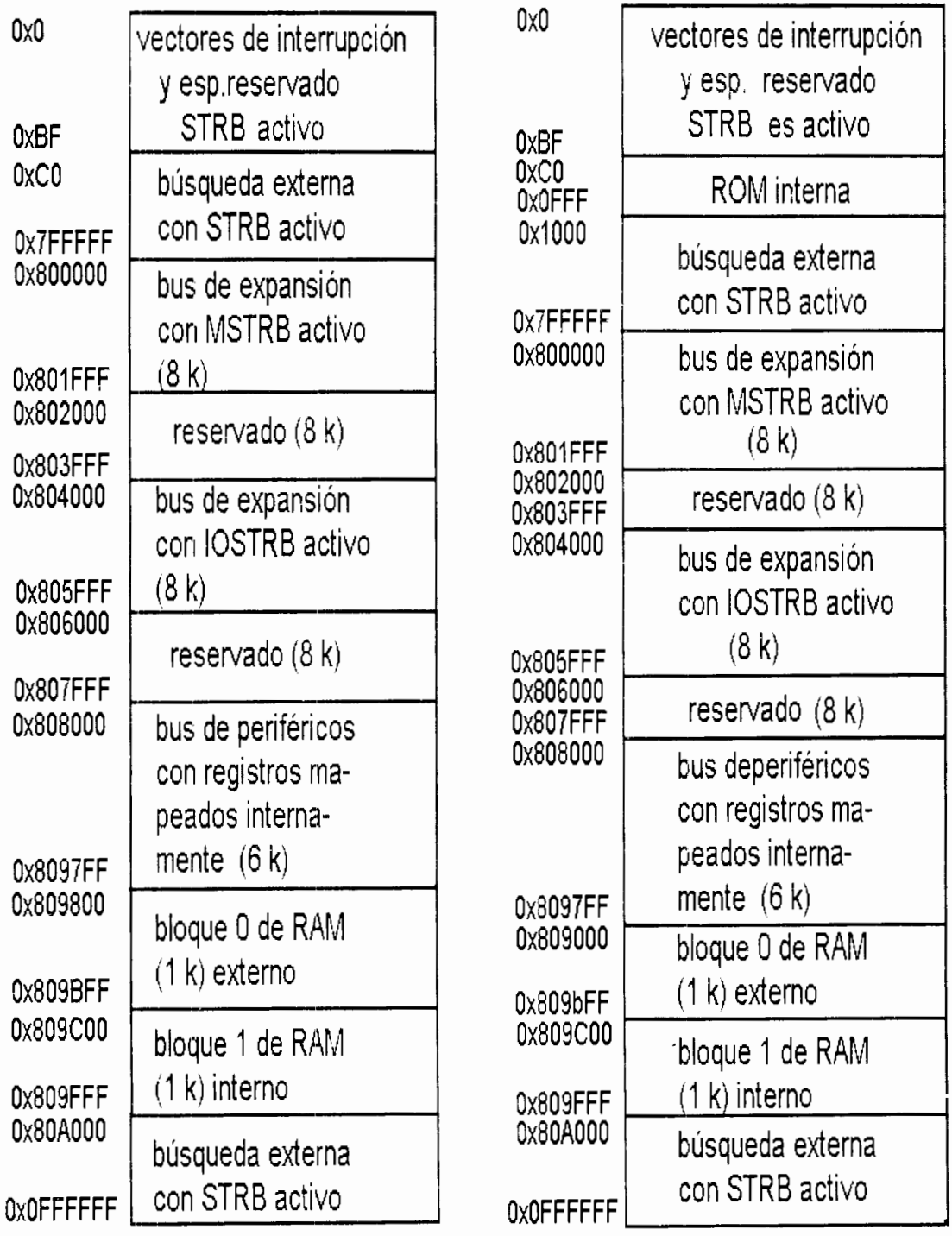

Figura 1.2: Mapa de memoria

- *Directo:* el operando es el contenido de una dirección de 16 bits.
- *Indirecto:* un registro auxiliar indica la dirección del operando.
- *Relativo al PC:* se suma al PC un desplazamiento signado de 16 bits.

1.5 Operación del *bus* interno

El contador de programa (PC) se encuentra conectado al *bus* de direcciones de programa, y el registro de instrucción (IR) al *bus* de datos de programa; ambos *buses* pueden accesar sólo una palabra cada ciclo de máquina. En cuanto al *bus* de direcciones de datos, éste permite dos accesos en un ciclo de máquina.

El controlador de DMA posee sus propios *buses* de direcciones (DMAADDR) y de datos (DMADATA), lo cual permite al controlador accesar a memoria en paralelo con los accesos a memoria de los *buses* de programa y datos.

1.6 Operación del *bus* externo

Está constituido por un *bus* primario de direcciones de 24 bits y un *bus* de expansión de 13 bits. Este *bus* accesa memoria de programa, datos y espacio de E/S. Se utiliza una señal externa $\overline{RDY}$ para generar estados de espera, cuando se accesan memorias o dispositivos lentos.

Interrupciones externas

El CPU soporta cuatro interrupciones externas ($\overline{INT3} - \overline{INT0}$). Maneja también interrupciones internas y una señal externa de $\overline{RESET}$ no protejida. Estas señales pueden interrumpir, ya sea al CPU o al controlador de DMA. Cuando el CPU responde a la interrupción, el pin $\overline{IACK}$ puede utilizarse como una contraseña externa a la interrupción.

Señalización para ejecución de instrucciones sincronizadas

EL CPU utiliza dos banderas externas XF0 y XF1 configurables mediante software como entradas o salidas. Se utilizan por las operaciones de sincronización del procesador, las cuales soportan comunicación entre microprocesadores, así como también es posible establecer tiempos de espera, entre otras aplicaciones.

1.7 Control de periféricos

Los periféricos incluidos en el circuito integrado son dos puertos seriales y dos temporizadores. El control de ellos se realiza a través de registros mapeados en memoria, para lo cual se utiliza un *bus* periférico específico. Dichos registros de control se encuentran en las localidades 808000h hasta la 8097FFh.

Temporizadores

Los dos módulos de temporización de 32 bits pueden funcionar como temporizadores, o bien, como contadores de eventos con dos modos de señalización y una señal de reloj generada, ya sea interna o externamente. Asociado a cada módulo, hay un pin que puede ser configurado como entrada de reloj para el temporizador o como una salida controlada por el mismo procesador.

Puertos seriales

Ambos puertos son completamente independientes uno del otro, pero idénticos en su configuración de registros y también configurables para operar con palabras de datos de 8, 16, 24 o 32 bits.

La señal de reloj para cada puerto serial puede ser generada en forma interna o externa, contándose con un divisor de frecuencia.

1.8 Acceso directo a memoria (DMA)

El controlador de DMA, como ya se mencionó, puede leer o escribir en la memoria sin interferir con la operación del CPU. Esto es de gran utilidad cuando se interactúa con dispositivos externos de baja velocidad, evitando disminuir el desempeño del CPU. El controlador de DMA contiene sus propios generadores de direcciones, registro fuente y destino, y contadores de transferencia. Dado que el controlador posee sus propios *buses* de datos y direcciones, se minimizan los conflictos con el CPU. Una operación de DMA consiste en la transferencia de una palabra o un bloque de datos desde o hacia la memoria.

1.9 Operaciones con ductería (pipeline)

El microprocesador, al accesar una instrucción de memoria y proceder a ejecutarla, realiza básicamente cuatro operaciones, las cuales son ejecutadas en el modo ductería como sigue:

Unidad de búsqueda (FETCH) (F)

Obtiene las palabras de instrucción de memoria y actualiza el contador de programa (PC).

Unidad de decodificación (D)

Realiza la decodificación de la instrucción y genera una dirección, también controla las modificaciones realizadas a los registros auxiliares y al apuntador del apilado.

Unidad de lectura (R)

En caso de ser necesario, realiza lectura de operandos.

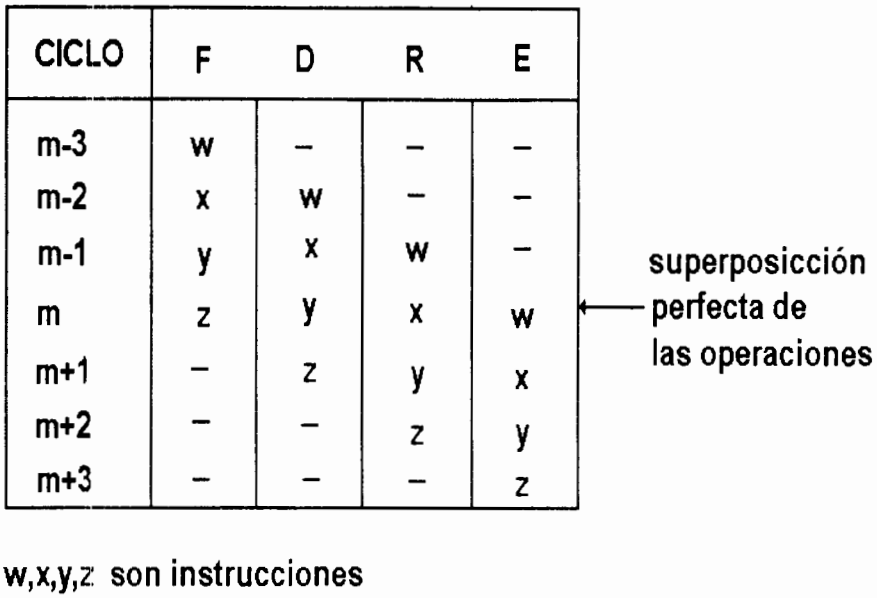

Figura 1.3: Ductería

Unidad de ejecución (E)

De requerirse, lee los operandos del archivo de registros, ejecuta la operación necesaria y actualiza el archivo de registros. En ocasiones también escribe resultados previos a memoria.

Cuando estas cuatro operaciones se superponen perfectamente en el mismo ciclo de ductería, operando en paralelo, la potencialidad del ductería se utiliza plenamente (figura 1.3). Para el programador, la ejecución de instrucciones en ductería es transparente, sin embargo, éste puede buscar que el acomodo de ellas favorezcan a que se realice óptimamente.

Durante la ejecución de operaciones en ductería pueden generarse conflictos debidos a los diferentes accesos que realiza una instrucción en particular, o bien, al uso del apilado que pueda realizar, entre otras causas.

Prioridad de las unidades de ductería

Se asigna una prioridad a las operaciones, esto es, el orden en que se llevan a cabo:

- Ejecución (E)
- Lectura (R)
- Decodificación (D)
- Búsqueda (F)
- Canal de DMA (DMA) (en caso de presentarse)

Cuando una instrucción está lista para entrar al siguiente nivel de ductería, pero ese nivel no está listo para aceptar a dicha instrucción se presenta un conflicto. En este caso, la unidad de más baja prioridad espera hasta que la siguiente unidad en prioridad complete su ejecución.

Los conflictos entre las operaciones de DMA y la estructura de ductería se minimizan debido a que el controlador de DMA posee sus propios *buses* de datos y direcciones.

Conflictos en ductería

Pueden agruparse en tres categorías principales:

1. *Conflictos de saltos.* Se generan cuando alguna instrucción lee o modifica el contador de programa (PC).

2. *Conflictos con registros.* Involucran retardos generados al leer o escribir los registros cuando sean usados para generar direcciones.

3. *Conflictos con memoria.* Ocurren cuando las diferentes unidades del microprocesador compiten por los recursos de memoria.

1.10 Uso de los recursos del sistema

El microprocesador soporta aritmética entera y de punto flotante, lo que permite al programador concentrarse en el algoritmo que debe programar y preocuparse lo menos posible de problemas como escalamiento, rango dinámico o sobreflujos.

Al efectuarse un *RESET*, el microprocesador finaliza la ejecución del programa en curso y coloca el valor del vector de *reset* en el contador de programa (PC). El *reset* por hardware (pin externo) también inicializa varios registros y bits de estado.

Después del *reset* deben inicializarse modos de operación, apuntadores de memoria, interrupciones y especificaciones propias de la aplicación. Dentro de la configuración inicial debe incluirse la asignación de valores iniciales a los registros mapeados en memoria y en la estructura de interrupciones.

Control de flujo dentro de un programa

El modelo de programación de este microprocesador posee diversas estructuras que facilitan la programación de algoritmos de procesamiento digital de señales, como son:

- Llamadas a subrutinas (llamadas, trampas y retornos).
- Uso del apilado por software.
- Saltos retardados.
- Operaciones en modo multiprocesador.
- Interrupciones.
- Modo de repetición.

Llamadas a subrutinas (llamadas, trampas y retornos)

Una vez diseñada la subrutina correspondiente, esta se ejecuta mediante las instrucciones CALL, CALLcond y TRAPcond. A través de las instrucciones RETScond y RETIcond se regresa de la subrutina invocada, lo que restaura automáticamente el *stack*.

Uso del apilado por software

El microprocesador tiene un apilado controlado por software cuya localización se determina por el contenido del apuntador al apilado (SP). Dicho apilado se accesa no solamente por las instrucciones CALL y RETS, sino también como una forma de almacenamiento temporal con las instrucciones PUSH y POP para números enteros, y PUSHF y POPF para números de punto flotante. El apuntador del apilado no se inicializa por hardware durante el *reset*, por lo que es importante asignarle un valor con una dirección de memoria predeterminada.

Saltos retardados

Este tipo de saltos funcionan como los saltos normales, sin embargo, no omiten el uso de la estructura de ductería, por lo que es conveniente procurar utilizarlos. Cuando un salto retardado es encontrado dentro de un programa, las tres instrucciones siguientes son ejecutadas. Las restricciones en este caso son que ninguna de estas tres instrucciones sea un salto, llamada a subrutina, retorno de subrutina o de interrupción, instrucción de repetición, instrucción trampa o instrucción de espera.

Operación en modo multiprocesador

Estas operaciones, auxiliadas mediante señalización externa, garantizan la integridad de la comunicación e incrementan la velocidad de ejecución.

Para mantener el acceso a una memoria global y compartir datos de una manera coherente, es necesario un protocolo para arbitrar este tipo de procesamiento. Este es el propósito de un pequeño conjunto de instrucciones que ayudan a la sincronización (LDFI, LDII, SIGI, STFI, STII).

Interrupciones

Se encuentran en forma vectorizada y existe un orden de prioridad entre ellas. Cuando una interrupción ocurre, se activa el correspondiente bit en el registro de banderas de interrupción (IF). Si en el registro de habilitación de interrupciones (IE), el bit correspondiente se encuentra activo, y las interrupciones a su vez se han habilitado por el bit GIE en el registro de estado, se ejecuta la rutina de interrupción respectiva. Existe también la posibilidad de escribir en el IE y de esta manera forzar la interrupción por software, o bien, deshabilitarla para que no se lleve a cabo.

Exceptuando rutinas de interrupción muy simples, es importante asegurar que el contexto del procesador (es decir, el archivo de registros) se haya respaldado previamente al dar el salto a la subrutina. Este conjunto de registros debe almacenarse en el apilado con anterioridad al brinco de subrutina y recuperarse de manera inversa (el apilado puede verse como una memoria LIFO), conservándose entonces el contexto original.

Modo de repetición

Se consideran dos modos de repetición: repetición de un bloque de código (RPTB) y repetición de una sola instrucción (RPTS). El control del número de repeticiones, así como las direcciones inicial y final del bloque que se van a repetir, se realiza manipulando los registros RS, RE y RC.

1.11 Conjunto de instrucciones

Con un total de 113 instrucciones, el microprocesador TMS320C30 provee un versátil juego de instrucciones, tanto de propósito general, como aquellas que están especialmente diseñadas para cálculo intensivo.
A su vez el conjunto se subdivide en grupos de instrucciones especializadas en carga almacenamiento, operaciones lógicas o aritméticas de dos o tres operandos, operaciones paralelas, justificando plenamente la existencia de dos ALU, operaciones de control de programa, donde sobresalen las instrucciones para repetición de instrucciones o bloques de ellas, así como cinco instrucciones previstas especialmente para ejecución compartida con otros microprocesadores C30. Todas las instrucciones ocupan al codificarse una palabra completa de 32 bits, y algunas de ellas se ejecutan en un solo ciclo. La eficiencia en la ejecución depende también de la optimación del código que favorezca las operaciones en ductería.

Instrucciones de carga y almacenamiento

Se consideran 12 instrucciones de este tipo para realizar cargas de memoria hacia un registro, cargas de un registro hacia memoria, o manipulación de datos en el apilado del sistema. Dos de estas instrucciones pueden efectuar cargas condicionales, tabla 1.2.

Instrucción	*Descripción*	*Ciclos*
LDE	Carga exponente de punto flotante.	1
LDF	Carga valor de punto flotante.	1
LDFcond	Carga v. de p. f. condicionalmente.	1
LDI	Carga un entero.	1
LDIcond	Carga un entero condicionalmente.	1
LDM	Carga mantisa de punto flotante.	1
POP	Extrae un entero del apilado.	1
POPF	Extrae un valor de p. f. del apilado.	1
PUSH	Deposita un entero en el apilado.	1
PUSHF	Deposita un valor p. f. al apilado.	1
STF	Almacena un valor de p.f.	1
STI	Almacena un entero.	1

Tabla 1.2: Instrucciones de carga y almacenamiento

Instrucciones de dos operandos

Son 35 instrucciones de este tipo, donde el operando fuente puede ser una localidad de memoria, un registro, o parte de la palabra de instrucción. El operando destino es siempre un registro del CPU. Se realizan operaciones con enteros, de punto flotante, lógicas y de aritmética de precisión múltiple, tabla 1.3.

Operaciones con tres operandos

Estas operaciones son un beneficio directo de la duplicación de los it buses internos en el CPU.

En este caso, el CPU toma dos operandos fuente, ya sea de registros o memoria, o bien, un operando y un contador, y deposita el resultado de la operación en un registro. Las 14 instrucciones de tres operandos se distinguen por terminar siempre en número 3, tabla 1.4.

Instrucciones para control de flujo

Este grupo consta de 15 instrucciones utilizadas para realizar saltos, llamar a subrutinas, ejecutar repeticiones de segmentos de código y efectuar ciclos. Cabe hacer notar la existencia de saltos retardados, en los cuales las siguientes tres instrucciones después del salto son cargadas sin incrementar el PC, dando por resultado un salto en un solo ciclo, lo que a su vez agiliza la ejecución en ductería, tabla 1.5.

Instrucciones para operar sincronizadamente

El microprocesador TMS320C30 posee cinco instrucciones diseñadas especialmente para permitir la comunicación entre procesadores y el manejo de señales externas de sincronización.

Instrucciones para operaciones en paralelo

Como consecuencia de la duplicación interna de it buses, cierto número de operaciones de carga, aritméticas y lógicas, están permitidas para su ejecución de manera paralela. Al ser escrito el código fuente, se indica la operación paralela con una doble barra (||) entre ambas operaciones. A continuación se enlistan los pares de instrucciones permitidos. Es claro que se debe buscar el paralelismo durante la programación, puesto que todas las operaciones paralelas permitidas se ejecutan en un solo ciclo.

Instrucción	*Descripción*	*Ciclos*
ABSF	Valor absoluto de un valor de p.f.	1
ABSI	Valor absoluto de un entero.	1
ADDC	Suma de enteros con acarreo.	1
ADDF	Suma de valores de p.f.	1
ADDI	Suma de enteros.	1
AND	Operación AND a nivel de bits.	1
ANDN	AND a nivel de bits con complemento.	1
	de uno de los operandos.	1
ASH	Corrimento aritmético.	1
CMPF	Compara valores de p. f.	1
CMPI	Compara enteros.	1
FIX	Convierte valor de p.f. a entero.	1
FLOAT	Convierte entero a p.f.	1
LSH	Corrimiento lógico.	1
MPYF	Multiplica valores de p.f.	1
MPYI	Multiplica enteros.	1
NEGB	Negación entera con préstamo.	1
NEGF	Negación de p.f.	1
NEGI	Negación entera.	1
NORM	Normalización de un valor de p.f.	1
NOT	Complemento lógico a nivel de bits.	1
OR	Operación OR a nivel de bits.	1
RND	Redondeo de un valor p.f.	1
ROL	Rotación a la izquierda de un bit.	1
ROLC	Rotación a la izquierda de un bit,	1
	incluyendo al bit de acarreo.	1
ROR	Rotación a la derecha de un bit.	1
RORC	Rotación a la derecha de un bit,	1
	incluyendo al bit de acarreo.	1
SUBB	Sustracción de enteros con préstamo.	1
SUBC	Sustracción condicional de enteros.	1
SUBF	Sustracción de valores de p.f.	1
SUBI	Sustracción de enteros.	1
SUBRB	Sustracción inversa de enteros con	1
	préstamo.	1
SUBRF	Sustracción inversa de valores	1
	de p.f.	1
SUBRI	Sustracción inversa de enteros.	1
TSTB	AND a nivel de bits entre dos ente-	1
	ros, pero sin sustitución de regs.	1
XOR	Operación OR exclusiva.	1

Tabla 1.3: Instrucciones de dos operandos

Instrucción	*Descripción*	*Ciclos*
ADDC3	Suma con acarreo.	1
ADDF3	Suma de valores de p.f.	1
ADDI3	Suma de enteros.	1
CMPF3	Compara valores de p. flotante.	1
CMPI3	Compara enteros.	1
LSH3	Corrimiento lógico.	1
MPYF3	Multiplicación de valores p.f.	1
MPYI3	Multiplicación de enteros.	1
OR3	Operación OR a nivel de bits.	1
SUBB3	Sustracción de enteros con préstamo.	1 1
SUBF3	Sustracción de valores de p.f.	1
SUBI3	Sustracción de enteros.	1
TSTB3	Operación AND a nivel de bits, en versión de 3 operandos.	1 1
XOR3	Operación OR exclusiva, en versión de 3 operandos.	1 1

Tabla 1.4: Operaciones con tres operandos

Instrucción	*Descripción*	*Ciclos*
Bcond	Salto condicional (estándar).	4
BcondD	Salto condicional (retardado).	1
BR	Salto incondicional (estándar).	4
BRD	Salto incondicional (retardado).	1
CALL	Llamada a subrutina.	4
CALLcond	Llamada condicional a subrutina.	5
DBcond	Decremento y salto condicional (estándar).	4
DBcondD	Decremento y salto condicional (retardado).	1
IDLE	Desocupado hasta interrupción.	1
NOP	No realiza ninguna operación.	1
RETIcond	Regreso de interrupción condicionado.	4
RETScond	Regreso de subrutina condicionado.	4
RPTB	Repite bloque de instrucciones.	4
RPTS	Repite una sola instrucción.	4
SWI	Ejecuta una interrupción del emulador (instr. reservada).	4

Tabla 1.5: Instrucciones para control de flujo

Instrucción	*Descripción*	*Ciclos*
LDFI	Carga un valor de punto flotante.	1*
LDII	Carga un entero.	1*
SIGI	Señalización externa a través de los	1
	pines XF0 y XF1.	1
STFI	Almacena un valor de punto flotante.	1
STII	Almacena un entero.	1

Tabla 1.6: Instrucciones para operar sincronizadamente

* La duración de estas instrucciones es de un ciclo, siempre y cuando la bandera $XF = 0$.

ABSF	|| STF	NEGI	|| STI
ABSI	|| STI	NOT3	|| STI
ADDF3	|| STF	OR3	|| STI
ADDI3	|| STI	STF	|| STF
FIX	|| STI	SUB3	|| STI
FLOAT	|| STF	XOR3	|| STI
LDF	|| STF	LDF	|| LDF
LDI	|| STI	LDI	|| LDI
LSH3	|| STI	MPYF3	|| ADDF3
MPYF3	|| STF	MPYF3	|| SUBF3
MPYI3	|| STI	MPYI3	|| ADDI3
NEGF	|| STF	MPYI3	|| SUBI3

Tabla 1.7: Instrucciones para operaciones en paralelo

Capítulo 2

Herramienta de desarrollo EVM

2.1 Introducción

En este capítulo se describe brevemente la herramienta de desarrollo para ejecutar algunos de los algoritmos que se proponen más adelante. El módulo de evaluación (EVM) es una tarjeta diseñada para usarse con el software depurador de código (*C Source Debugger*), el cual ofrece al usuario la posibilidad de realizar una emulación en pantalla, a la vez que los algoritmos se ejecutan en la tarjeta. La intervención del controlador de bus de prueba (TBC) permite realizar la emulación directa en la tarjeta, parar la ejecución del programa, y visualizar registros y memoria. Mediante el uso de un puerto de emulación MPSD y el TBC se carga el programa en la EVM.

Se define un protocolo de comunicación entre el *host* (una computadora IBM PC/AT compatible) y la tarjeta objetivo, es decir, la EVM. Dicho protocolo basa su operación en el intercambio de información entre registros mapeados en memoria, lo que permite desarrollar software de propósito especifico para una aplicación dada, utilizando un lenguaje de alto nivel que permita el acceso a memoria, como lo es el lenguaje C.

El programa debe estar previamente formateado según la especificación COFF (common object file format), para generar el código objeto a partir de segmentos de programa escritos, tanto en ensamblador como en ANSI C. De este código objeto se genera el código ejecutable en un archivo `.out`, que debe cargarse en la tarjeta ya sea mediante el programa `evmload.exe`, o bien, con el ambiente del depurador.

2.2 Características generales

Interfaz con el *bus* anfitrión

La tarjeta emuladora tiene las especificaciones necesarias para insertarse en una ranura libre del bus IBM PC/AT de tarjetas de 8 bits. En la figura 2.1 se esquematiza la tarjeta EVM y sus componentes principales.

La diferencia más notoria sobre otros emuladores existentes consiste en lo que sus diseñadores denominan emulación en el sistema mismo (*embedded in-system emulation*). Esto significa que la emulación se efectúa dentro del sistema objetivo (la EVM), realizándose la interfase

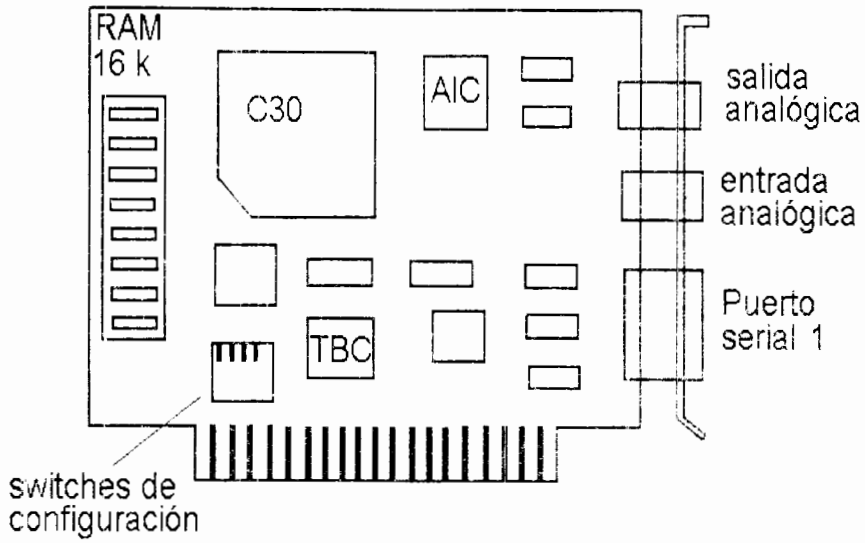

Figura 2.1: Módulo de evaluación (EVM) para el TMS320C30

con el *host* mediante un circuito destinado a ello: el 74ACT8990, *Test Bus Controller*, (TBC). Dicho circuito controla el intercambio de información entre el TMS320C30 y el *bus* de la computadora anfitriona.

En programa que se ejecuta desde la memoria de la computadora personal, se comunica en realidad con el TBC a través del poleo (monitoreo) de seis direcciones de 16 bits. El TBC interactúa con el TMS320C30 mediante interrupciones por hardware por medio de los pines INT0, INT1 e INT2. Esto quiere decir que la comunicación desde un programa anfitrión en la computadora opera con poleo, mientras que el TMS320C30 se comunica con el TBC a través de interrupciones de hardware, lo que le proporciona a este último total autonomía del anfitrión mientras no se realice una requisición de intercambio de información.

La configuración del sistema, como se ha referido, provee una tasa moderada de intercambio de información (alrededor de 200 kbytes por segundo), sin embargo, garantiza la sincronización de dicha transferencia, ya que si un dato tiene que ser escrito o leído por el host (la PC) es necesario hacer un poleo y borrado de banderas en los registros mapeados en memoria. En ciertas condiciones, es posible evitar dicho poleo, que eleva la eficiencia (*throughput*) [1] hasta 400 kbytes por segundo.

Como hardware adicional en la generación de señales de protócolo (*glue logic*) se incluyen un par de PAL's (dispositivos lógicos programables de uso generalizado), y dos transreceptores 74ALS652. El formato de interfase entre el TBC y el puerto de emulación del TMS320C30 sigue las especificaciones MPSD (*modular port scan device*) de Texas Instruments.

Memoria externa

El módulo de evaluación posee un banco de memoria SRAM de 16k-palabras con cero estados de espera y se halla directamente conectado al espacio de direcciones del bus primario. Las memorias utilizadas son de la marca Cypress, modelo CY7C164-35VC, con un tiempo

[1] *Throughput* es un término generalmente aceptado para referir la cantidad de información digital que puede obtenerse como salida de un sistema, se utiliza como un parmetro de desempeño.

de acceso de 35 ns, lo que requiere un reloj de 30 MHz como base de tiempo en el C30 para satisfacer los requerimientos de la SRAM. Este banco de memoria provee al módulo de cierta autonomía en cuanto a capacidad de almacenamiento se refiere. Es posible almacenar aproximadamente dos segundos de una señal muestreada a 8 kHz sin ser necesario el intercambio de información con el host.

Interfase analógica

De los dos puertos seriales de que dispone el TMS320C30, el primero de ellos (puerto 0) se ha utilizado para comunicarse con un controlador de interfase analógica: el TLC32044 *analog interface controller* (AIC). Dicha interfase necesita de una base de tiempo, por lo que se utilizó para ello el primer temporizador, además del pin XF0 como $\overline{RESET}$ del AIC. Este controlador provee de una interfaz A/D y D/A con 14 bits de resolución, tasa de muestreo y filtrado programables, que cubren satisfactoriamente los requerimientos de una arquitectura para procesar señales de voz.

La entrada analógica acepta voltajes en un rango de ± 1.5 V y de ± 3 V. Se incluye también una etapa de acondicionamiento de la señal proveniente del exterior que consta de un seguidor y un amplificador con ganancia 2, basados en amplificadores TL072 con entrada JFET. La salida analógica acepta una carga mínima de salida de 300Ω, sin embargo, para poder ser manejada por un equipo amplificador comercial, es necesario acoplarla para cargas de hasta 4Ω. Con una salida típica de 1.5 Vpp, el AIC se conecta a una etapa de atenuación con una ganancia de x0.2 para posteriormente ser amplificado por un factor de x20 en el amplificador de potencia LM386, de manera que la ganancia de voltaje resultante sea de x4. Por ejemplo, para una carga de 8Ω con una salida inicial de 1.5 Vpp, el voltaje resultante a la salida del LM386 debe ser de 6 Vpp.

Puerto serial externo

El puerto serial 1, completamente independiente de aquél que se usa como interfase A/D y D/A, se ha previsto como un medio de comunicación al exterior de la EVM. Se incluye además en el conector de salida de 10 pins la señal XF1. Los pins disponibles en dicho conector se presentan en la tabla 2.1.

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

Tabla 2.1: Señales de puerto serial

Los usos que pueden darse a dicho conector son diversos: conexión con otras EVM para ejecutar procesamiento distribuido, conexión a otro dispositivo serial para recibir información, como puede ser el *backplane* de un PBX, etc.

2.3 Configuración y generación de código

Configuración del mapa de memoria de la tarjeta

Es posible configurar algunas características de la tarjeta mediante microinterruptores. Como se muestra en la tabla 2.2, con los interruptores marcados SW1, SW2, se selecciona la dirección base de los registros utilizados por el programa de aplicación en el *host* para la comunicación hacia la EVM. La primera dirección base debe ser la opción por default, a menos que exista otro periférico (por ejemplo, un cursor) en ese espacio de memoria.

SW1	SW2	*Dirección base*
ON	ON	$0 \times 0240 - 0 \times 025F$
ON	OFF	$0 \times 0280 - 0 \times 029F$
OFF	ON	$0 \times 0320 - 0 \times 033F$
OFF	OFF	$0 \times 0340 - 0 \times 035F$

Tabla 2.2: Microinterruptores en EVM

Los interruptores SW3 y SW4, por lo general, se mantienen en ON para permitir el modo de microprocesador y la salida SBM en alta impedancia, respectivamente.

Configuración interna del TMS320C30

Al inicio de cualquier programa que sea cargado en la memoria de la tarjeta, es absolutamente necesario realizar una inicialización de algunas variables y banderas internas. A continuación se muestra un segmento de programa que efectúa dicha inicialización, habilitando sólo las interrupciones que son necesarias, como aquéllas utilizadas para comunicarse con el AIC. Esta secuencia de instrucciones puede variar un poco, dependiendo principalmente de los periféricos que se utilicen y de las interrupciones que se manejen.

```
elstack     .usect  "stack",100h
            .sect   "vectors"
PARMS:
reset       .word   sysinit         ;al arrancar ejecuta sysinit
int0        .word   interr0
int1        .word   interr1
int2        .word   interr2
int3        .word   interr3
xint0       .word   transmit0
rint0       .word   receive0       ;rint0 es causada por el AIC
xint1       .word   transmit1
rint1       .word   recieve1
tint0       .word   timer0
tint1       .word   timer1
```

```
dint0       .word   dmadone
            .sect   "comdata"
stack_addr  .word   elstack         ; direccion del apilado
dma_ctl     .word   000808000h      ; registro de control global del dma
mcntlr0     .word   000808064h      ; registro de control del bus primario
mcntlr1     .word   000808060h      ; registro de control del bus secundario
t0_ctladdr  .word   000808020h      ; reg. control global timer 0
t1_ctladdr  .word   000808030h      ; reg. control global timer 1
p0_addr     .word   000808040h      ; reg. control global puerto serial 0
enbl_sp0_r  .word   000000020h      ; palabra para habilitar int. por Rx en p.s. 0
t0_ctlinit  .word   0C00002C1h      ; habilita timer 0 como salida de reloj
                                    ; (H)1/2 (reloj interno), el timer
                                    ; continua funcionando
p0_global    word   00e970300h      ; palabra para configurar el p.s. 0
DEPOSITO            .set    1800h ; palabra para habilitar deposito
ENBL_GIE            .set    2000h ; palabra para habilitar interrupciones
ENBL_XINT0          .set    0010h ; habilita la interrupcion de Tx del p.s. 0
ENBL_RINT0          .set    0020h ; habilita la interrupcion de Rx del p.s. 0

    text
sysinit:
            xor     ie,ie           ; borra IE y deshabilita todas las ints.
            xor     if,if           ; tambien borra todas las banderas de
                                    ; int. en IF
            ldp     PARMS           ; carga DATA PAGE POINTER con los MSBs
                                    ; de PARMS para poder usar direccionamiento
                                    ; directo
            ldi     @stack_addr,sp  ; carga un 100h en el STACK POINTER
            ldi     DEPOSITO,st     ; habilita el bit de deposito en el
                                    ; STATUS REGISTER
            ldi     0,r0            ; carga 0 en R0
            ldi     @mcntlr0,ar0    ; carga AR0 con la direccion del PRIMARY BUS
                                    ; CONTROL REGISTER
            sti     r0,*ar0         ; pone el bus I/O en READY
            ldi     @mcntlr1,ar0    ; carga AR0 con la direccion del EXPANSION BUS
                                    ; CONTROL REGISTER
            sti     r0,*ar0         ; tambi\'en lo pone en READY
            or      80h,ST          ; habilita overflow mode en STATUS REGISTER
            or      ENBL_GIE,st     ; habilita bit GLOBAL INT, que es el 13 de ST

interr0:    reti
interr1:    reti
interr2:    reti
interr3:    reti
transmit0:  reti
transmit1:  reti
recieve1:   reti
```

```
timer0:    reti
timer1:    reti
dmadone:   reti

** como ejemplo, se propone una rutina de atencion de interrupcion, **
** que atienda a aquella que se genera cuando se recibe un dato en p.s.

receive0:  push    st                  ; salva en el apilado el valor de
           push    r0                  ; algunos registros
           push    R3
           push    ar0
           push    dp
           ldp     PARMS

**** instrucciones propias de la rutina ****

           pop     dp                  ; saca del apilado el valor de los
           pop     ar0                 ; registros salvados
           pop     R3
           pop     r0
           pop     st
           reti
           .end
```

Generación de código

Una vez que se ha compilado o ensamblado cualquier programa para el TMS320C30 se genera un archivo `.obj` el cual contiene ya el código de máquina en formato COFF (*Common Object File Format*) que es posible ejecutar en un sistema basado en este microprocesador.

Ensamblador

Si se trata de un programa escrito exclusivamente en lenguaje ensamblador, se utiliza el programa `asm30.exe` como sigue:

asm30 [archivo de entrada [archivo objeto [archivo de lista]]] [-opciones]

Las opciones más comunes son:

-v especifica la versión, puede ser -v30 para el TMS320C30 o -v40 para el TMS320C40.
-l produce un archivo de listado (.LST).
-i especifica un directorio para búsqueda de archivos especificados en las directivas `.copy`, `.include` o `.mlib`.
-c realiza la compilación con sensibilidad a la mayúsculas.
-x produce una tabla de referencias cruzadas al final del archivo de listado.

Una opción es invocar al programa sin ningún argumento, lo que ocasiona que el programa pida entonces los argumentos al usuario.

Ligador

Es necesaria una última etapa de ligado en la cual se asignan las direcciones definitivas para que el código ejecutable sea localizado dentro del mapa de memoria del sistema objetivo. En este caso, dicho sistema objetivo es la EVM, y es necesario realizar esta etapa de ligado con información adicional sobre la localización de los datos, del código de programa, memorias RAM y ROM, etc. Tal acción se ejecuta con el programa **lnk30.exe**, cuya sintaxis es la siguiente:

lnk30 [-opciones] archivo 1 ... archivo n

Las opciones más usuales son:

-m produce un archivo de mapeo (.MAP)
-o especifica a continuación el archivo de salida (por default A.OUT)
-q al momento de ligar no aparece el rótulo de TI
-x obliga a que relean las librerías y detecte referencias no encontradas en la primera pasada
-a produce código con direcciones absolutas
-r produce código con direcciones relocalizables

Un ejemplo común de ligado es el siguiente:

lnk30 archivo, archivo de comandos, -o archivo de salida

El ligador también tiene la opción de ser invocado sin argumentos, lo cual ocasiona que dicho programa pida los datos necesarios al usuario.

Archivos de comandos

Para que el ligador conozca la localización de la memoria dentro de un sistema objetivo, es necesario especificarle tal información en un archivo de texto con extensión `.cmd` (archivo de comandos). En dicho archivo de comandos se especifica la naturaleza de los bloques de memoria usados (lectura y escritura, sólo lectura) así como su localización y extensión dentro del mapa de memoria. A cada bloque se le da un nombre y a su vez se localizan en ellos las diferentes secciones del código. La división en secciones del código al momento de ligar obedece a que tal código puede tener diferentes características, siendo en algunas ocasiones código de programa, una tabla de datos fija o una tabla de datos variable.

En el formato **.coff** se definen tres secciones por default:

sección .text contiene código ejecutable.
sección .data contiene datos inicializados e invariables.
sección .bss es un espacio reservado para variables no inicializadas (por ejemplo, los estados internos de un filtro).

Además, el ligador permite que el usuario pueda crear otras secciones con características similares, y básicamente las clasifica en dos categorías: inicializadas y no inicializadas.

Secciones inicializadas

Contienen datos o código ejecutable, las secciones .text y .data son secciones inicializadas. El usuario puede crear nuevas secciones inicializadas con las directivas .sect y .asect.

Secciones no inicializadas

Se utilizan para reservar espacio en memoria para datos no inicializados. La sección .bss es de este tipo y el usuario puede definir nuevas secciones con la directiva .usect. Pueden encontrarse varios ejemplos de secciones no inicializadas creadas por el usuario con los nombres "stack", "vectors" y "comdata".

A continuación se presenta el contenido de un archivo de comandos utilizado para ligar el código de inicialización listado en la sección anterior.

```
MEMORY
{
INT_V : org = 0x000000, len = 0x40
SRAM  : org = 0x000040, len = 0x1FC0
a     : org = 0x002000, len = 0x2000
RAM0  : org = 0x809800, len = 0x400
RAM1  : org = 0x809C00, len = 0x400
}
SECTIONS
{
 vectors: {} > INT_V
 comdata: {} > SRAM
 text   : {} > SRAM
 buffer : {} > a
 stack  : {} > RAM1
}
```

2.4 Comunicación entre el *host* y la EVM

La microcomputadora anfitriona, es decir, el *host*, tiene comunicación con la EVM dentro de un esquema maestro-esclavo, donde obviamente el host ejecuta el papel de maestro. Como ya se ha mencionado, la interfase entre ambos la realiza un circuito denominado *Test Bus Controller* (TBC), el cual es direccionado desde el bus AT mediante los PAL que actúan como decodificadores de memoria, y que también se utilizan entre el TBC y el C30 para sincronizar adecuadamente la producción de las interrupciones con la base de tiempo propia del microprocesador C30. Un esquema de la interconexión del bus PC/AT, el TBC y el TMS320C30 se muestra en la figura 2.2.

Operaciones de lectura/escritura desde el *host* hacia el TBC

Se direccionan 6 registros desde el bus PC/AT, los cuales se localizan física y lógicamente en el TBC, y a través de los cuales un programa de aplicación interactúa con el TMS320C30. Aunque la mayor parte son de 16 bits, uno de ellos en realidad sólo existe como una dirección

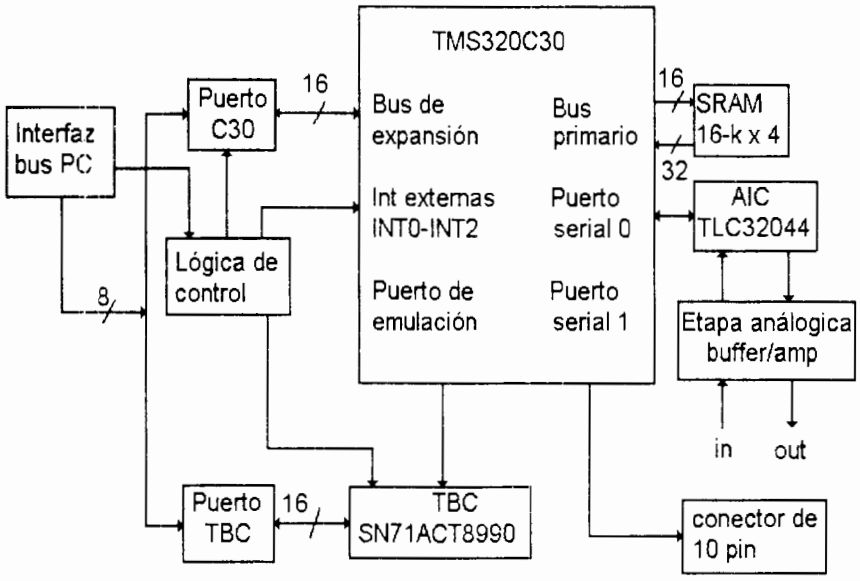

Figura 2.2: Interconexión entre el bus anfitrión, el TBC y el TMS320C30

y al realizarse una escritura, se ejecuta un reset por software (registro SOFT RESET). En la tabla 2.3 se presentan los desplazamientos a partir de una dirección base (ver configuración) de los registros mencionados, así como su nemónico asociado y su tamaño en bits.

Registro	*Desplazamiento*	*Tamaño (bits)*	*Acceso*
RESERVADO	$0 \times 0000 - 0 \times 0008$	-	-
CONTROL5	$0 \times 000A$	16	L/E
RESERVADO	$0 \times 000C - 0 \times 0012$	-	-
MINOR CMD	0×0014	16	L/E
RESERVADO	$0 \times 0016 - 0 \times 001E$	-	-
STATUS 0	0×0400	16	L
RESERVADO	$0 \times 0402 - 0 \times 041F$	-	-
COM CMD	0×0800	8	L/E
COM DATA	0×0808	16	L/E
SOFT RESET	0×818	0	E

Tabla 2.3: Registros desde el *bus* PC/AT

Manejo de eventos a través del TBC

El TBC considera cuatro eventos provenientes del *bus* PC/AT, los cuales a su vez disparan las correspondientes interrupciones al TMS320C30. Se enumeran de EVT0 a EVT3 y su utilización se describe a continuación:

EVT0: Embedded Simulation Support

Evento no modificable, se utiliza como parte del protocolo entre el TBC y el puerto de emulación del TMS320C30.

EVT1: Host Read Acknowledge

Se activa (EVT1=1) cuando el TMS320C30 escribe al registro de comunicaciones. Entonces el *host* puede leer el dato de dicho registro y poner en cero esta bandera para que una futura escritura del C30 pueda ser detectada.

EVT2: Host Write Acknowledge

Se activa (EVT2=1) cuando el TMS320C30 efectúa una lectura del registro de comunicaciones. Esto quiere decir que el C30 ha recogido el dato. Acto seguido, el *host* debe poner de nuevo en cero a esta bandera y puede escribir otra vez.

EVT3: TMS320C30 Reset Source

Mientras esta bandera sea mantenida en 1, el C30 permanece en estado de reset. Existe un registro cuyo direccionamiento provoca dicho estado.

Escritura de datos

Para realizar la escritura de un dato localizado en una variable en la memoria del host hacia el TMS320C30, se llevan a cabo una serie de pasos, los cuales se ejemplifican en el siguiente segmento de un programa en Microsoft C:

```
#define IOBASE      0x0240
#define MINOR_CMD   0x0014
#define COM_DATA    0x0808
#define STATUS0     0x0400
#define MASK1       0x0004
#define MASK2       0x6044

/* la variable 16_bit_data contiene el valor que va a ser escrito */
unsigned short 16_bit_data;

do {

/* 1. borra la bandera EVT2 localizada en el reg. MINOR_CMD */

outpw(IOBASE + MINOR_CMD, MASK1);

/* 2. escribe el dato en el registro de comunicaciones */

outpw(IOBASE + COM_DATA, 16_bit_data);

/* 3. actualiza el registro de estado del TBC */
```

```
outpw(IOBASE + MINOR_CMD, MASK2);

/* 4. verifica que el dato ha sido leido del lado del TMS320C30 */

}while( inpw(IOBASE + STATUS0) & MASK1);

/* actualiza el registro de estado */

outpw(IOBASE + MINOR_CMD, MASK2);

/* 5. una vez abandonado el ciclo anterior se puede empezar de
nuevo desde 1 */
```

Lectura de datos

Para recoger un dato del registro de comunicaciones se debe polear la bandera EVT2 hasta que se indique la presencia de un dato en el registro de comunicaciones, como se muestra en el siguiente segmento de código:

```
#define IOBASE      0x0240
#define MINOR_CMD   0x0014
#define COM_DATA    0x0808
#define STATUS0     0x0400
#define MASK0       0x0002
#define MASK2       0x6044

unsigned short 16_bit_data;

do {

/* actualiza el registro de estado del TBC */

outpw(IOBASE + MINOR_CMD, MASK2);

/* si la bandera EVT1 es activa, se borra, en otro caso
regresa a actualizar el registro de estado */

if (inpw(IOBASE + STATUS0) & MASK0)
outpw(IOBASE + MINOR_CMD, MASK0);

}while(!(inpw(IOBASE + STATUS0) & MASK0));

/* procede a leer el dato */

16_bit_data = inpw(IOBASE + COM_DATA);
```

Reinicialización por software

La inicialización del TMS320C30 se da de manera automática cuando se le aplica energía por primera vez. Sin embargo, el TBC provee la manera de aplicarle un reset por software, a través del registro CONTROL5, de la siguiente manera:

```
#define IOBASE      0x0240
#define CONTROL5    0x000A
#define RESET_ON    0x0808
#define RESET_OFF   0x0800

outpw(IOBASE + CONTROL5, RESET_ON);

outpw(IOBASE + CONTROL5, RESET_OFF);
```

Existe una tercera manera de reinicializar al TMS320C30 sin la intervención del TBC. Como se había mencionado, el registro SOFT RESET no existe físicamente, pero su direccionamiento provoca que una señal activa en bajo se genere en el PAL UA5, el cual aplica el *reset* al microprocesador.

```
#define IOBASE      0x0240
#define SOFT_RESET  0x0818

outpw(IOBASE + SOFT_RESET, 0);
```

2.5 Manejo de información desde el TMS320C30

En el caso del TMS320C30, el manejo del protocolo de comunicaciones se realiza por medio de interrupciones, a través de las cuales dicho dispositivo da servicio a los procesos de lectura/escritura por parte del *host*. Cuando el TMS320C30 desea efectuar una lectura o escritura, la lógica de direccionamiento utilizada en las PAL's accesa los registros del TBC. El registro COM DATA, visto desde el espacio de direcciones del C30, se encuentra en todo el *bus* de expansión como un solo registro de 16 bits habilitado para lectura y escritura. Cualquier dirección contenida dentro del espacio $0 \times 804000 - 0 \times 805FFF$ corresponde a este registro.

Mapa de memoria de la tarjeta

En la tabla 2.4 se presenta el mapa de memoria del módulo de evaluación, donde se incluyen todos aquellos registros mapeados en memoria para el control de periféricos, así como el espacio estándar de datos y programa. La memoria puede utilizarse para almacenar programa, datos, y también los vectores de *reset* y de interrupción.

Manejo de interrupciones en el TMS320C30

Como se ha visto, las interrupciones por hardware son el medio mediante el cual el TBC avisa al microprocesador que se realizan operaciones de lectura/escritura. En este caso, las

Dirección	*Función*
bus primario	
$0 \times 000000 - 0 \times 003FFF$	16k-palabras de SRAM
bus de expansión	
0×804000	registro COM DATA hacia el *host*
0×808000	control global del DMA
0×808004	dirección fuente de DMA
0×808006	dirección destino de DMA
0×808008	contador de transferencia de DMA
0×808020	control global del timer 0
0×808024	contador del timer 0
0×808028	periodo del timer 0
0×808030	control global del timer 1
0×808034	contador del timer 1
0×808038	periodo del timer 1
0×808040	control global del puerto serie 0
0×808042	puerto control Tx p. serie 0
0×808043	puerto control Rx p. serie 0
0×808044	control de timer Rx/Tx p. serie 0
0×808045	contador de timer Rx/Tx p. serie 0
0×808046	periodo de timer Tx/Rx p. serie 0
0×808048	dato de Tx puerto serie 0
$0 \times 80804C$	dato de Rx puerto serie 0
0×808050	control global del puerto serie 1
0×808052	puerto control Tx p. serie 1
0×808053	puerto control Rx p. serie 1
0×808054	control de timer Rx/Tx p. serie 1
0×808055	contador de timer Rx/Tx p. serie 1
0×808056	periodo de timer Tx/Rx p. serie 1
0×808058	dato de Tx puerto serie 1
$0 \times 80805C$	dato de Rx puerto serie 1
0×808060	control del *bus* de expansión
0×808064	control del *bus* primario
$0 \times 809800 - 809FFF$	2k-palabras de RAM interna, acceso dual

Tabla 2.4: Mapa de la memoria de la tarjeta

interrupciones se detectan por nivel durante el flanco de bajada de la señal H1, el C30 puede aceptar dos interrupciones de la misma fuente cada dos ciclos de reloj de H1.

Los PAL encargados de generar las fuentes de interrupción para el C30 se desarrollaron como máquinas de 4 estados (con 2 flip-flops) con entrada asíncrona, dichos PAL generan las señales $\overline{CNTLINT}$, $\overline{WRINT}$ y $\overline{RDINT}$ que controlan las correspondientes entradas INT0, INT1 e INT2 en el TMS320C30. La cuarta entrada disponible para interrupción se deshabilita mediante una resistencia de pull-up a +5V.

Para asegurar que sólo sea generada una interrupción en cada operación de lectura o escritura por parte del *host*, después de una interrupción, el generador de interrupciones espera a que se genere una señal de lectura o escritura y que ésta sea inactiva, para después pasar la correspondiente interrupción al TMS320C30 a través de los pines INT0 a INT2.

Cada una de estas entradas realiza una de las siguientes funciones en el protocolo de comunicaciones.

INT0: command interrupt

Su estado activo indica que el *host* depositó un comando en el registro de comandos. Se genera cuando el *host* escribe un dato de 8 bits, pero no cuando el *host* hace una lectura, lo que le posibilita a este último polear el registro de estado del C30 sin ocasionar cambios en algún registro.

INT1: data write interrupt

Su estado activo indica que el *host* realizó una escritura de 16 bits en COM DATA.

INT2: data read interrupt

Su estado activo indica que el *host* realizó una lectura de 16 bits de COM DATA.

El uso de interrupciones por hardware posibilita al controlador de DMA para dar servicio a la entrada y salida de datos.

2.6 Manejo del controlador de interfaz analógica

El controlador de interfaz analógica (AIC) es el dispositivo que hace posible que los algoritmos diseñados puedan ser realizados en tiempo real con las siguientes limitaciones:

- El rango de voltajes que acepta es cuando mucho de $\pm$ 3 volts.
- La frecuencia de muestreo máxima configurable es de 19200 Hz.

Asimismo, son programables las características anteriores:

- Un filtro corrector $\sin(x)/x$ en la salida del convertidor D/A.
- Un filtro paso bajas antialiasing de entrada de frecuencia programable.

El AIC se configura mediante un protocolo serial, posee un pin de reset y necesita también una base de tiempo. Estas cuestiones fueron resueltas en la arquitectura de la tarjeta como sigue:

1. Se empleó el puerto serial 1 para comunicarse con el AIC.
2. El timer 0 se utilizó como base de tiempo.
3. El pin XF0 funciona como señal de reset para el AIC.

Para lograr una adecuada comunicación entre el AIC y el TMS320C30 se deben tomar en cuenta las siguientes características: el AIC es capaz de realizar el intercambio de información con otro dispositivo mediante un protocolo serial asíncrono (es decir, a una tasa variable de transmisión) y con una longitud de palabra de datos de 16 bits; además, el AIC maneja datos de tipo entero, mientras que el TMS320C30 es un dispositivo de punto flotante, por lo que debe realizarse la conversión adecuada de la información.

El temporizador, usado como base de tiempo, se configura a una frecuencia de $f_{MCLK}7.5MHz$, siendo la mitad de la señal interna $H1$ (es decir: 15MHz/2 = 7.5 MHz). El pin $XF0$ es programable modificando su estado en el registro IOF.

El AIC soporta dos tipos de transmisión:

1. *Transmisión primaria.* Cuando el dispositivo simplemente intercambia información desde el convertidor A/D o hacia el convertidor D/A, los datos de 14 bits deben estar acomodados en los bits menos significativos de la palabra de transmisión, asignando dos ceros a los bits menos significativos.
2. *Transmisión secundaria.* Este tipo de transmisión se inicia cuando se envían al AIC palabras con los dos bits menos significativos con valores de uno. A continuación del inicio de la transmisión secundaria, se envían palabras que configuran tanto la frecuencia de muestreo de entrada como la de salida, la frecuencia de corte de filtro *antialiasing*, así como la ventana de voltaje de los convertidores.

Tanto para Tx como para Rx deben configurarse dos parmetros, denominados A y B, que establecen las frecuencias de muestreo como sigue:

$$f_{SCF} = \frac{f_{MCLK}}{2A} \tag{2.1}$$

$$f_c = \frac{f_{MCLK}}{2AB} \tag{2.2}$$

Donde f_{SCF} es la frecuencia del filtro de capacitores conmutados de entrada (filtro pasobajas) y f_c es la frecuencia de muestreo. Para ello deben calcularse los números TA, TB para la transmisión y RA y RB para la recepción, que serán enviados posteriormente al inicio de una transmisión secundaria, como puede apreciarse en la figura 2.3.

Los números TA y RA se utilizan para realizar un ajuste fino de las frecuencias de muestreo, y su valor se resta al contenido original de TA y RA. A tiene un rango de valores entre 1 y 31, mientras que B, con un bit más de resolución, puede variar entre 1 y 63.

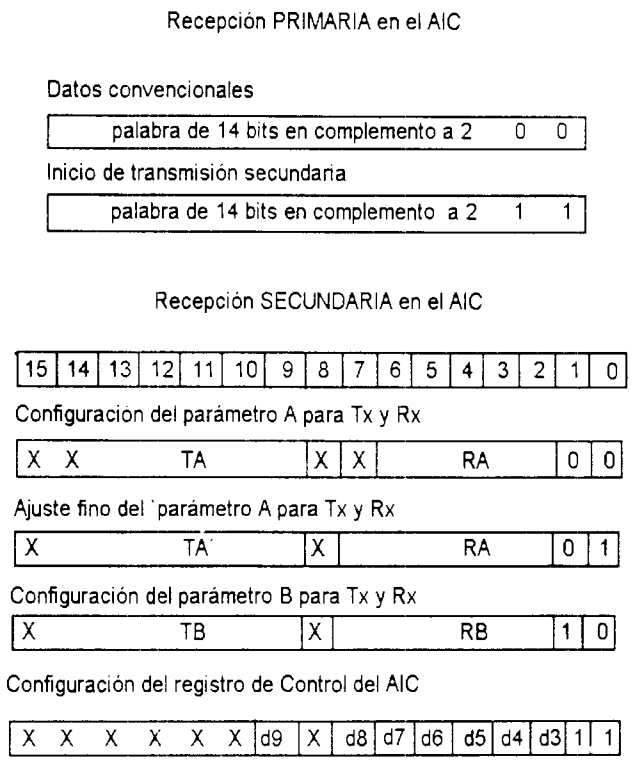

Figura 2.3: Formatos de transmisión de datos hacia el AIC

Además, existe un registro de control, accesible al colocar unos en los dos bits menos significativos de la palabra transmitida después del inicio de la transmisión secundaria. En dicha palabra se encuentran los bits $d2$ a $d9$ con los siguientes significados:

Bit	*Función*
d2	Habilita filtro paso-altas en A/D
d3	Habilita "loopback"
d4	Selecciona entrada analógica primaria(0) o secundaria (1)
d5	Tx y Rx síncrona
d7,d6	Selecciona entrada ± 1.5V (1,0) o entrada ± 3.0V (1,1)
d9	Habilita filtro corrector $(\text{sen } x)/x$ en D/A

Tabla 2.5: Parámetros del registro de control del TLC32044

A continuación se muestra la secuencia de inicialización del AIC para configurarlo a una tasa de muestreo de 8 kHz:

```
aicreset:
          ldi     2,iof               ; configura XF0 para que sea salida con 0,
                                      ; con lo que se pone en reset al AIC
          ldi     @t0_ctladdr,ar0     ; carga AR0 con la direccion del TIMER 0
                                      ; GLOBAL CONTROL REG
          ldi     1,r1                ; con 1, tclk0 ser (H)1/2
          sti     r1,*+ar0(8)         ; pone 1 en el TIMER 0
          ldi     @t0_ctlinit,r1      ; carga R1 con la configuracion del TIMER 0
          sti     r1,*ar0             ; pone la configuracion en TIMER GLOBAL
          ldi     @p0_addr,ar0        ; carga en AR0 la dir. del GLOBAL CONTROL
                                      ; REG del puerto serial 0
          ldi     111h,r1             ; carga en R1 la configuracion del p.s. 0
          sti     r1,*+ar0(2)         ; carga la configuracion en S.P. 0 Tx PORT
                                      ; CONTROL
          sti     r1,*+ar0(3)         ; carga la config. en S.P. 0 Rx PORT CONTROL
          ldi     @p0_global,r1       ; carga R1 con la config. para p.s. 0
          sti     r1,*ar0             ; escribe configuracion en el GLOBAL CONTROL
                                      ; REG del p.s. 0
          xor     r1,r1               ; borra el contenido de R1
          sti     r1,*+ar0(8)         ; pone un 0 en el reg. de Tx del p.s. 0
          rpts    99                  ; espera a que hayan ocurrido 50 ciclos
          nop                         ; del timer  para que se estabilice
          ldi     6,iof               ; pone a XF0 en 1, saca del reset al AIC
                                      ; y comienza a enviarle su configuracion
          call    wait_transmit_0     ; pelea a ver si se ha transmitido algo
          ldi     3,r1                ; guarda un 3 en R1
          sti     r1,*+ar0(8)         ; escribe el 3 en el S.P 0 Tx DATA
          call    wait_transmit_0     ; espera a que se haya transmitido
          ldi     1a34h,r1            ; carga la palabra de config. del reg de
                                      ; control del AIC
          sti     r1,*+ar0(8)         ; envia dicha palabra al AIC
          ldi     *+ar0(12),r1        ; lee lo que haya en el S.P. 0 Rx DATA
          call    wait_transmit_0     ; espera a que este listo el puerto
          ldi     3,r1                ; ahora, siguiendo el mismo protocolo
          sti     r1,*+ar0(8)         ; vuelve a mandar un 3
          call    wait_transmit_0     ; espera a que se vacie el puerto
          ldi     2a7h,r1             ; carga la config. de la frec. de muestreo
          sti     r1,*+ar0(8)         ; la envia
          ldi     *+ar0(12),r1        ; lee lo que haya en la recepcion
          xor     if,if               ; borra todas las banderas de interrupcion
          or      @enbl_sp0_r,ie      ; habilita el P.S. 0
          rets
wait_transmit_0:
          xor     if,if               ; borra todas las banderas de interrupcion
wloop:    tstb    10h,if              ; revisa la bandera de Tx efectuada
          bz      wloop
          rets
```

Instrucción	*Función*	*Ejemplo*
asm	Rastrear programa en ensamblador	asm
c	Rastreo en c (modo por default)	c
mix	Modo mixto	mix

Tabla 2.6: Instrucciones para cambiar de modo de programación

2.7 Software asociado: el depurador de código

El depurador de código es una poderosa herramienta de software que aprovecha la característica de emulación que posee la tarjeta. Mediante este es posible realizar una emulación en la cual, a diferencia de una simulación, se realiza la ejecución de las instrucciones realmente en el interior del microprocesador, a la vez que se obtienen las ventajas típicas de los simuladores, como son:

- Realizar la ejecución paso a paso, o utilizando *breakpoints*.
- Observar los cambios en diferentes variables a la vez.
- Poder modificar aleatoriamente el contenido de las variables durante la ejecución.
- Utilizar archivos como entrada o salida de datos.
- Definir "macros"con los comandos más usados.
- Realizar el rastreo de variables en un sistema de múltiples ventanas.

Al igual que el microprocesador al cual apoya, este programa está orientado hacia la programación en lenguaje C, por lo que sus instrucciones suelen tener similitudes principalmente en la sintaxis con la forma de expresar operaciones y direccionamientos en dicho lenguaje.

El depurador se invoca al ejecutar el comando **evm30.exe** y por default se inicializa con la información contenida en **evminit.cmd**, si se desea se puede editar este archivo, o bien, invocar el programa con otro archivo de comandos mediante la sentencia

evm30 -t archivo.cmd

2.7.1 Comandos del depurador de código

A continuación se resumen las instrucciones más usuales dentro del ambiente del depurador de código.
Sólo se permite asociar a un puerto un archivo de entrada y otro de salida. Un archivo puede tener asociados múltiples puertos.

Tecla	*Función*
$< F2 >$	Último comando
$< F3 >$	Extiende DISSASEMBLY y CPU ocultando FILE y CALLS
$< F4 >$	Cerrar la ventana actual de una estructura
$< F5 >$	Equivale a RUN
$< F6 >$	Brinca entre ventanas
$< F8 >$	Ejecuta paso a paso
$< F9 >$	Entrar a una subestructura (ver manipular datos)
$< F10 >$	Siguiente instrucción
$< TAB >$	Recorre el buffer de comandos
$< ESC >$	Salir

Tabla 2.7: Funciones de algunas teclas

Instrucción	*Función*	*Ejemplo*
zoom	Maximiza la ventana actual	zoom
move	Mueve la ventana actual	move
win	Cambia de ventana	win CPU, win FIL, win CAL win MEM, win COM, win DIS
size	Cambia dimensiones de la ventana	size

Tabla 2.8: Instrucciones para manejo de ventanas

Instrucción	*Función*	*Ejemplo*
system	Llamadas al DOS en modo SHELL	system, system "dir/w "[,0][,1] con 1 espera un ENTER con 0 regresa de inmediato
exit	Salir del modo SHELL	exit
take	Toma un archivo de comandos (.cmd)	take arch[.cmd][,0] el 0 inhibe el eco de los comandos
cd/chdir	Cambia el directorio de trabajo	cd user
cls	Borra la ventana de la ventana COM	cls
alias	Definir un macro	alias macro1,"comando1;comando2"
	Ejecutar el macro	macro1
	Mostrar macros existentes	alias
unalias	Cancelar el macro	unalias macro1
dir	Listar el directorio actual	dir
use	Nombrar directorios de fuentes adicionales	use otrodir
reset	Resetear el sistema (la EVM)	reset
restart	Continuar ejecución del programa	restart
quit	Salir del depurador	quit

Tabla 2.9: Instrucciones para ejecutar tareas del sistema

Instrucción	*Función*	*Ejemplo*
load	Cargar archivo ejecutable (.out)	load c:file[.out]
file	Cargar un archivo ASCII en FILE	file myfile.ext
func	Mover el apuntador de la ventana FILE a una función específica	función
dasm	Desplegar código a partir de una función específica	dasm función
	Desplegar código a partir de una función con fuente en C o ASM	dasm 0x100200
addr	Desplegar código a partir de una función con fuente en C	addr[función][direcc]
reload	Cargar archivo .out sin su tabla de símbolos	reload file1[.out]
sload	Cargar sólo la tabla de símbolos de un archivo .out	sload file1[.out]
	Modificar una instrucción en lenguaje ensamblador	patch 0x100200[,instrucción]
calls	Reabrir la ventana CALLS	calls

Tabla 2.10: Instrucciones para cargar y desplegar programas

Instrucción	*Función*	*Ejemplo*
wa	Abrir ventanas para rastrear variables	wa variable[,etiqueta] (aparece la ventana watch)
	Observar el contenido de una localidad	wa *0x100300
	Observar una variable	wa variable [,etiqueta][o,x,i,etc.]
	Observar como entero	wa i
	Observar como float	wa f
	Observar como double	wa d
	Observar con etiqueta	wa i,NEW i
wd	Borrar una variable de la ventana	wd N (N es el renglón)
wr	Borrar toda la ventana WATCH	wr
setf	Cambiar la forma de desplegar tipo de datos	setf int, x (los enteros se verán como hexa)
	Reestablecer el despliegue normal	setf*
	Listar los tipos de datos y su modo de despliegue actual	setf
disp	mostrar en la ventana un dato:	
	Mostrar una estructura	disp estructura
	Mostrar en detalle un elemento de la estructura	hacer <click> en el elemento
	Cerrar la ventana actual	presionar $< F4 >$
	Entrar a una "subestructura"	presionar $< F9 >$
	Conmutar entre "subestructuras" "hijas"	$< CTRL >< PgUp >$ $< CTRL >< PgDn >$
?	Evaluar y desplegar el resultado de una expresión	
	Evaluar una loc. de memoria	?*0x100200
	Evaluar loc. de memoria, forzando el despliegue como un tipo de datos	disp *(float *)0x100200
	Evaluar el contenido de una variable	? variable[c,f,o,etc.]
mem	Desplegar a partir de una localidad en la ventana MEMORY	mem 0x100200, pc [*,c,d,e,f,x,o,p,u]
	Saber el valor de un símbolo	mem &símbolo
eval	Evaluar una expresión	eval –R3
whatis	Averiguar el tipo de una variable o una función	whatis variable, whatis función

Tabla 2.11: Instrucciones para desplegar y manipular datos y variables

Instrucción	*Función*	*Ejemplo*
ba	Marcar un punto de prueba	ba [dirección,función o etiqueta]
br	Desmarcar todos los puntos	br
bd	Desmarcar un punto	bd 0x100200
bl	Listar los puntos marcados	bl
clk	Examinar el contenido de CLK, que cuenta ciclos de reloj entre breakpoints	? clk

Tabla 2.12: Instrucciones para utilizar puntos de prueba (breakpoints)

Instrucción	*Función*	*Ejemplo*
sconfig	Cambiar archivo de configuración de la pantalla	sconfig archivo[.clr]
prompt	Cambiar el prompt de la línea de comandos	prompt miprompt
ssave	Salvar en archivo la configuración	ssave archivo.ext
scolor	Modificar atributos de ventana	scolor menu-bar, yellow

Tabla 2.13: Instrucciones para personalizar la pantalla

Instrucción	*Función*	*Ejemplo*
ma	Dar de alta un bloque de memoria	ma 0x100200,0x0400,RAM/ROM
mc	Conectar el bloque a un archivo de entrada	mc 0x100200,filein.ext,READ
mc	Conectar el bloque a un archivo de salida	mc 0x100200,fileout.ext,WRITE
md	Dar de baja un bloque de memoria	md 0x100300
mi	Desconectar un puerto de memoria	mi 0x100200,READ
LDI	Leer de un puerto en ensamblador	LDI@0x100200,R0
ml	Desplegar configuración actual de memoria	ml
map	Habilitar/deshabilitar el mapeo en memoria	map ON/OFF
mr	Reinicializar todo el mapa de memoria	mr
fill	Llenar un bloque de memoria con un valor	fill 0x100200,0x100,0xffffffff
ms	Salvar un segmento de código como un programa .coff	ms dir-de-inicio, longitud, archivo.ext

Tabla 2.14: Instrucciones para manejo de memoria

Instrucción	*Función*	*Ejemplo*
step	Recorrer N sentencias en ensamblador	step N
cstep	Recorrer N sentencias en lenguaje C	cstep N
next	Recorrer N sentencias en ensamblador pero sin entrar a las subrutinas	next N
cnext	Recorrer N sentencias en C. pero sin entrar a las subrrutinas	cnext N
run	Ejecutar un programa	run
runf	Ejecutar un programa sin la intervención del puerto emulador (la EVM ejecuta libremente)	runf
halt	"Conecta" de nuevo el puerto de emulación a la EVM	halt
go	Ejecutar una subrutina	go subrrutina
runb	Ejecutar hasta el siguiente breakpoint contando ciclos de CPU	runb (benchmarking)
ret	Ejecutar el código de la función actual en C y regresar cuando termina	ret

Tabla 2.15: Instrucciones para depuración de programas

Capítulo 3

Realización de filtros digitales con TMS320C30

Para la realización de filtros digitales con TMS320C30 necesitamos algunas instrucciones que nos permitan multiplicar, sumar dos números, retardar la señal, etc. En este capítulo se explican algunas de ellas y al final se presenta un programa completo en ensamblador del filtro FIR.

3.1 Instrucciones del TMS320C30

3.1.1 MPYF3-Multiplicación en punto flotante

Sintaxis

Directo:	MPYF3	$src2, src1, dst$
Indirecto:	MPYF3	$*+AR2(IR0), src1, dst$

Ejecución

$$src1 \times src2 --> dst$$

El contenido del registro src1 se multiplica por el contenido de src2 y el resultado se guarda en dst.

Ejemplo 1

MPYF3 $R0, R7, R1$

Antes del comando:	Después del comando:
R0=057B400000h=6.281250e+01	R0=057B400000h=6.281250e+01
R7=0733C00000h=1.79750e+02	R7=0733C00000h=1.79750e+02
R1=0	R0=0D306A3000h=1.12905469e+04

Ejemplo 2

MPYF3 $*+AR2(IR0), R7, R2$

Antes del comando:	Después del comando:
AR2=809800h	AR2=809800h
IR0=12AH	IR0=12AH
R7=057B400000h=6.281250e+01	R7=057B400000h=6.281250e+01
R2=0h	R0=0D09E4A000h=8.82515625e+03
Los datos en 80992Ah=1.4050e+02	Los datos en 80992Ah=1.4050e+02

3.1.2 Instrucciones en paralelo MPYF3||ADDF3

En ensamblador existen instrucciones en paralelo que se realizan en un ciclo de reloj. El símbolo para las instrucciones que se ejecutan en paralelo es ||. Se pueden ejecutar estas instrucciones para multiplicación y guardar el resultado en un registro, o multiplicar y sumar en un ciclo de reloj.

Sintaxis

Directo: MPYF3 $srcA, srcB, dst1$
|| ADDF3 $srcC, srcD, dst2$

Indirecto: MPYF3 $*AR2(1)++, *--AR1(IR0), R0$
|| ADDF3 $R5, R7, R3$

Ejecución

$srcA \times srcB -->dst1$
|| $srcC + srcD -->dst2$

El contenido del registro srcA se multiplica por el contenido de srcB y el resultado se guarda en dst1. Al mismo tiempo se suma el contenido de srcC con el de srcD y el resultado se guarda en el registro dst2.

Ejemplo 3

MPYF3 *AR5++(1),*-(IR0),R0
|| ADDF3 R5,R7,R3

Antes del comando:	Después del comando:
AR5=8098C5h	AR5=8098C6h
AR1=8098A8h	AR1=8098A4h
IR0=4h	IR0=4h
R0=0h	R0=0467180000h=2.88867188e+01
R5=0733C00000h=1.79750e+02	R5=0733C00000h=1.79750e+02
R7=070C800000h=1.4050e+02	R7=070C800000h=1.4050e+02
R3=0h	R3=0820200000h=3.20250e+02
Los datos en 8098C5h=1.2750e+02	Los datos en 8098C5h=1.2750e+01
Los datos en 8098A4h=2.2500e+00	Los datos en 8098A4h=2.2500e+00

3.1.3 Multiplicar y cargar en paralelo MPYF3||STF

Sintaxis

$$\begin{array}{rll} \text{Directo:} & \text{MPYF3} & src2, src1, dst \\ & \| \ \text{STF} & src3, dst2 \\ \\ \text{Indirecto:} & \text{MPYF3} & *---AR2(1), R7, R0 \\ & \| \ \text{STF} & *AR0----(IR0) \end{array}$$

Ejecución

$$src1 \times src2 -- > dst1$$
$$\| \ src3 -- > dst2$$

El contenido del registro src1 se multiplica por el contenido src2 y el resultado se guarda en dst1. Al mismo tiempo el contenido src3 se guarda en el registro dst2.

Ejemplo 4

```
      MPYF3  * AR2(1),R7,R0
||    STF    R3,*AR0- -(IR0)
```

Antes del comando:	Después del comando:
AR2=80982Bh	AR2=80982Bh
R7=057B400000h=6.281250e+01	R7=057B400000h=6.281250e+01
R0=0h	R0=0D09E4A000h=8.82515625e+03
R3=086B280000h=4.7031250e+02	R3=086B280000h=4.7031250e+02
AR0=809860h	AR0=809858h
IR0=8h	IR0=8h
Los datos en 80982Ah=1.4050e+02	Los datos en 80982Ah=1.4050e+02
Los datos en 809860h=0h	Los datos en 809860h=4.703125e+02

3.1.4 Multiplicar y restar en paralelo MPYF3||SUBF3

Sintaxis

$$\begin{array}{rll} \text{Directo:} & \text{MPYF3} & srcA, srcB, dst1 \\ & || \ \text{SUBF3} & srcC, srcD, dst2 \end{array}$$

$$\begin{array}{rll} \text{Indirecto:} & \text{MPYF3} & R5, *++AR7(IR1), R0 \\ & || \ \text{SUBF3} & R7, *AR3----(1), R2 \end{array}$$

Ejecución

$$srcA \times srcB -->dst1$$
$$||\ srcD - srcC -->dst2$$

El contenido del registro srcA se multiplica por el contenido del registro srcB y el resultado se guarda en dst1. Al mismo tiempo el contenido del registro srcC se sustrae del contenido del registro srcD y el resultado se guarda en el registro dst2.

Ejemplo 5

$$\begin{array}{rll} & \text{MPYF3} & *--AR7(IR1), R5, R0 \\ || & \text{SUBF3} & R7, *AR3--(1), R2 \end{array}$$

Antes del comando:	Después del comando:
R5=034C000000h=1.2750e+01	R5=034C000000h=1.2750e+01
AR7=809904h	AR7=80090Ch
IR1=8h	IR1=8h-01
R0=0h	R0=0467180000h=2.88867188e+01
R7=0733C00000h=1.79750e+02	R7=0733C00000h=1.79750e+02
AR3=8098B2h	AR3=8098B1h
R2=0h	R2=05E3000000h=-3.9250e+01
Los datos en 80990Ch=2.50e+00	Los datos en 80990Ch=2.250e+00
datos en 8098B2h=1.4050e+02	Los datos en 8098B2h=1.4050e+02

3.1.5 Cargar en paralelo LDF||LDF

Sintaxis

$$\begin{array}{rll} \text{Directo:} & \text{LDF} & src2, dst2 \\ & || \ \text{LDF} & src1, dst1 \end{array}$$

$$\begin{array}{rll} \text{Indirecto:} & \text{LDF} & *--AR1(IR0), R7 \\ & || \ \text{LDF} & *AR7++(1), R3 \end{array}$$

Ejecución

$$src2 - - > dst2$$
$$|| \ src1 - - > dst1$$

El contenido del registro src2 se guarda en dst2 y al mismo tiempo el contenido de scr1 se guarda en el dst1.

Ejemplo 6

$$\begin{array}{ll} & \text{LDF} \quad *--AR1(IR0), R7 \\ || & \text{LDF} \quad *AR7++(1), R3 \end{array}$$

Antes del comando:	Después del comando:
AR1=80985Fh	AR1=809857h
IR0=8h	IR0=8h
R7=0h	R7=070C800000h=1.4050e+02
AR7=80988Ah	AR7=80988Bh
R3=0h	R3=6.281250e+01
Los datos en 809857h=1.4050e+02	Los datos en 809857h=1.4050e+02
Los datos en 80988Ah=6.281250e+01	Los datos en 80988Ah=6.281250e+01

3.1.6 Instrucciones para entrada y salida

Entrada del número

La muestra en la entrada puede ser obtenida del convertidor analógico digital (ADC), mediante las siguientes instrucciones:

$$\begin{array}{ll} \text{LDI} & @IN\_ADDR, AR2 \\ \text{FLOAT} & *AR2, R3 \end{array}$$

La dirección de la entrada (IN_ADDR) está cargada en el registro AR2. El número tipo entero obtenido desde ADC se convierte en el número flotante y éste es guardado en el registro extendido R3.

Salida del número

El número puede mandarse al convertidor digital-analógico (DAC) con las instrucciones:

```
LDI  @OUT_ADDR,AR3
FIX  R0,R1
STI  R1,*AR3
```

La dirección de salida está cargada en el registro AR3. El valor R0 tipo punto flotante es convertido en número entero y es cargado en R1. Este número se guarda en la dirección de salida que está determinada mediante el registro AR3.

3.1.7 Instrucción de repetición

El comando RPTS permite repetir la instrucción que sigue y también el bloque de las instrucciones que podemos repetir con la instrucción RPTB. Utilizando la instrucción RPTB, la dirección inicial se guarda en el registro de repetición RS (*Repeat Start register*). La última dirección que se debe repetir se guarda en un registro especial RE (*Repeat End address register*). En el registro especial RC (*Repeat Counter register*) se guarda el número de repeticiones que se desean.

Ejemplo 6

```
                LDI     9,RC
                RPTB    END_ADDR
                CALL    FILTER
                FIX     R0
END_ADDR        STI     R0,*AR3
```

El contador de la repetición está cargado con el número 9, eso significa que el bloque de las instrucciones se repetirá 10 veces, con las cuales se llama al subprograma FILTER. La dirección donde empieza el bloque se guarda en el registro RS y la última dirección se carga en el registro RE. El contenido de R0 que calcula la subrutina FILTER se convierte en un número tipo entero, el cual se guarda después en la dirección que señala el registro AR3.

Ejemplo 7

```
        LDF     0,R0
        LDI     29,AR2
        RPTS    AR2
        MPYF    *AR0++,*AR1++,R0
||      ADDF    R0,R2,R2
        ADDF    R0,R2
```

En este ejemplo se ejecutan en paralelo las instrucciones MPYF y ADDF. A consecuencia del trabajo en paralelo, el microcontrolador TMS320C25 realiza 33 millones de operaciones por segundo. El símbolo paralelo || junto con la instrucción ADDF indican que se realizan al mismo tiempo las instrucciones MPYF y ADDF.

Primeramente, se carga al registro R0 con cero y el número tipo entero (LDI) se carga al registro auxiliar AR2. En el registro AR2 se almacena el número 29. Entonces 29+1 veces se repiten las instrucciones MPYF y ADDF. Los números en los registros AR1 y AR2 se incrementan después de la multiplicación de los números guardados en las direcciones que señalan los registros auxiliares AR1 y AR0, y el resultado se carga al registro R0. Al mismo tiempo se suman los números guardados en los registros R0 y R2, y el resultado se guarda en el registro R2. Este cálculo se repite 30 veces hasta que el contenido en el registro auxiliar AR2 es igual a cero y se cumple la última instrucción. En esta instrucción se suma el contenido de R0 y R2, y el resultado se guarda en el registro R2. La última instrucción se cumple sólo una vez y se puede escribir también en la forma ADDF R0, R2, R2.

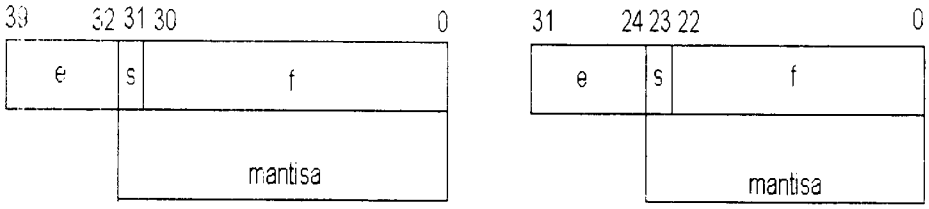

Figura 3.1: Formatos de los números con punto flotante: a) Precisión simple, b) Precisión extendida

3.1.8 Multiplicación con *buffer* circular

La multiplicación con *buffer* circular se utiliza muy a menudo si queremos, por ejemplo, calcular la convolución, correlación o realizar los filtros con la respuesta finita. El símbolo para buffer circular es %.

Ejemplo 8

```
LENGTH  .SET    30
        LDI     LENGTH,BK
        LDF     0,R0
        RPTS    LENGTH-1
        MPYF    *AR0++%,*AR1++%,R0
    ||  ADDF    R0,R2,R2
        ADDF    R0,R2,R2
```

En el programa, primeramente, el número 30 se guarda en la variable LENGTH. Esta variable señala la longitud del *buffer* circular. Después, la longitud del *buffer* se guarda en un registro especial BK. En el registro R0 se guarda cero. Las instrucciones MPYF y ADDF se repiten treinta veces LENGTH-1=29. Las direcciones guardadas en los registros auxiliares AR0 y AR1 se incrementan siempre en uno, hasta alcanzar el tope del *buffer* circular. En este ejemplo utilizamos dos *buffer* circulares para AR0 y AR1 que se marcan con %. Por ejemplo guardamos en el registro auxiliar AR0 la dirección de la variable X y en el registro auxiliar AR1 la dirección de variable Y. Después de 30 repeticiones tenemos en el registro auxiliar AR0 la dirección X+29 y en registro auxiliar AR1 la dirección Y+29.

Después de terminar la multiplicación en paralelo con la adición en los registros AR0 y AR1, se guarda la dirección X y Y.

3.2 Representación de los números en TMS320C30

En los registros de TMS320C30 podemos guardar los números de punto flotante con la precisión simple o extendida. Los registros para los números de punto flotante con precisión simple y precisión extendida se presentan en la figura 3.1.
El número que nos da el convertidor ADC es tipo entero y lo convertimos con la instrucción FLOAT en el formato con punto flotante. Antes de mandar el número al convertidor DAC es necesario convertirlo de punto flotante a tipo entero. La conversión desde la precisión

extendida a la precisión simple se realiza automáticamente. El número expresado en la forma extendida se puede guardar con 40 bits en los registros R0-R7. El resultado después de la multiplicación está expresado con la precisión extendida. El formato de número con precisión extendida está representado con un exponente (E) de 8 bits (bits 32-39), un bit de signo, (S, bit 31) y la parte fraccionaria (F, bits 0-30). En la figura 3.1 se muestran los formatos para los números expresados con la precisión simple y extendida. El número para el formato con punto flotante está en el rango $=-2^{128}$ hasta 2^{128}.

Donde el número N está representado en la forma:

$$N = \begin{cases} 01.F \times 2^E, & si\ el\ bit\ S = 0 \\ \\ 10.F \times 2^E, & si\ el\ bit\ S = 1 \end{cases} \tag{3.1}$$

Ejemplo 9

Convertir el número 07F38000h expresado con 32 bits a número decimal:

```
N=07F38000h = 0000 0111 1111 0011 1000 0000 0000 0000b
              ----E---- S-------------F-------------
```

El bit S=1, por lo que, mediante la ecuación 3.1, podemos escribir

```
N=10.111 0011 1000 0000 ...exp+07
N=10 111 0011. 1000 0000 ...exp+00
```

El bit más significativo de F es uno. Por lo que el número es negativo. Se debe complementar a dos el número N, es decir, sumar al complemento de N un bit.

```
N= 0 1 0 0 0 1 1 0 0, 0  1  1  1 ...
                                1
---------------------------------
N= 0 1 0 0 0 1 1 0 0, 1  0  0  0
  (8 7 6 5 4 3 2 1 0 -1 -2 -3 -4)
```

$$N_d = 2^{-1} + 2^2 + 2^3 + 2^7 = -140.5exp + 00 = -1.405e^{02}$$

Ejemplo 10

Convertir el número FC200000 hexadecimal expresado con 32 bits a número decimal:

```
N=FC200000h = 1111 1100 0010 0000 0000 0000 0000 0000b
              ----E---- S--------------F-------------
```

El bit S=0 y por eso, mediante la ecuación (3.1), podemos escribir

```
N=01.010 exp-04
```

El exponente es negativo porque el bit más significativo es uno. Entonces, el exponente E se calcula si los ceros son convertidos en uno y los unos en ceros, y sumamos con uno. Se tiene

```
exp = 0000 0011
                1
     ----------
exp = 0000 0100 =4
```

Entonces obtenemos

$$N = .0001\ \ 0100\ \ 0000... = 2^{-4} + 2^{-6} = \frac{5}{64} = 0.0781$$

Ejemplo 11

Convertir el número hexadecimal 070C8000 expresado con 32 bits a número decimal:

```
N=070C8000h = 0000 0111 0000 1100 1000 0000 0000 0000b
  E    F        ---E------S--------------F-------------
```

El bit más significativo de E y F es cero, por lo tanto el número y el exponente son positivos. El bit S=0, por lo que, mediante la ecuación (3.1), podemos escribir

```
N=0 1. 0 0 0 1 1 0 0  1 0 0 0 exp+07
N=0 1  0 0 0 1 1 0 0. 1 exp+00
   (7  6 5 4 3 2 1 0  -1)
```

Calculando el número N en la forma decimal, obtenemos

$$N_d = 2^{-1} - 2^2 + 2^3 + 2^7 = 140.5\ e^{+00} = 1.405\ e^{+02}$$

Ejemplo 12

Convertir el número hexadecimal 0733C000 expresado con 32 bits a número decimal:

```
N=0733C000h = 0000 0111 0011 0011 1100 0000 0000 0000b
  E    F       ---E------S--------------F-------------
```

El bit más significativo de E y F es cero, por lo tanto el número y el exponente son positivos. El bit S=0, por lo que, mediante la ecuación (3.1), podemos escribir

```
N=0 1. 0 1 1 0 0 1 1  1 1 0 0 exp+07
N=0 1  0 1 1 0 0 1 1. 1 1 exp+00
   (7  6 5 4 3 2 1 0 -1 -2)
```

Calculando el número N en la forma decimal, obtenemos

$$N_d = 2^{-1} + 2^{-2} + 2^0 + 2^1 + 2^4 + 2^5 + 2^7 = 179.75\ e^{+00} = 1.7975\ e^{+02}$$

Ejemplo 13

Convertir el número hexadecimal 05E30000 expresado con 32 bits a número decimal:

```
N=05E30000h = 0000 0101 1110 0011 0000 0000 0000 0000b
  E    F       ---E------S--------------F-------------
```

El bit más significativo de E es cero, por lo tanto, el exponente es positivo. El bit más significativo de F es uno, por lo que el número N es negativo. El bit S=0, por lo que, mediante la ecuación (3.1), podemos escribir

```
N=1 0. 1 1 0 0 0  1  1  exp+05
N=1 0  1 1 0 0 0. 1  1  exp+00
( 6 5  4 3 2 1 0 -1 -2)
```

Para obtener el número negativo calculamos el complemento del número N y sumamos con uno. Calculando obtenemos:

```
N=0 1 0 0 1 1 1 . 0 0
                  1
----------------------
N=0 1 0 0 1 1 1 . 0 1
 (  5 4 3 2 1 0  -1 -2)
```

Y el número N en la forma decimal tiene la forma

$$N_d = 2^{-2} + 2^0 + 2^1 + 2^2 + 2^5 = -39.25\ e^{+00}$$

3.3 Programas en ensamblador

En esta parte presentamos los programas para multiplicación de dos series de números y un programa para multiplicar una matriz con un vector.

3.3.1 Multiplicación de dos series de números

El programa en ensamblador para TMS320C30 se compone de un programa principal que se llama MULT.ASM y la subrutina MULTSUB.ASM.

```
     /* MULT.ASM - MULTIPLICACION DE DOS SERIES */

            .TITLE "MULT.ASM"          ;El titulo del programa
            .GLOBAL BEGIN,MULT         ;Los simbolos REF/DEF
            .DATA                      ;Se ensambla a la seccion de datos
H_ADDR      .WORD  HN                  ;La primera direccion del vector HN
X_ADDR      .WORD  XN                  ;La primera direccion del vector YN
IO_OUT      .WORD  804001H
HN          .FLOAT  1,2,3,4            ;Los numeros del vector HN
XN          .FLOAT  2,3,4,5            ;Los numeros del vector XN
            .TEXT                      ;Se ensambla a la seccion de texto
BEGIN       LDP        H_ADDR          ;Se inicializa la pagina de datos
            LDI   @H_ADDR,AR0          ;La direccion de HN en AR0
            LDI   @X_ADDR,AR1          ;La direccion de XN en AR1
            LDI   @IO_OUT,AR2          ;AR2 se configura como la salida
            LDI   3,RC                 ;El contador de repeticion RC=3
            CALL  MULT                 ;Salto a subprograma MULT
            STI   R2,*AR2              ;El resultado a la salida IO_OUT
WAIT        BR    WAIT                 ;Espera
```

Cada serie HN y XN tiene cuatro números. La serie HN tiene los números 1, 2, 3, 4 y la serie XN tiene los números 2, 3, 4, 5. El programa principal MULT.ASM llama una subrutina MULT en el programa MULTSUB.ASM.

```
          /* MULTISUB.ASM - SUBRUTINA PARA MULTIPLICACION /*

            .TITLE    "MULTISUB.ASM"     ;El titulo
            .GLOBAL    MULT              ;El simbolo REF/DEF
            .TEXT                        ;Ensambla a la seccion de texto
MULT        LDF       0,R0               ;R0=0
            LDF       0,R2               ;R2=0
            RPTS      RC                 ;Se ejecuta instruccion 3 veces
            MPYF      *AR0++,*AR1++,R0   ;(AR0)*(AR1)->R0
         ||  ADDF     R0,R2              ;El resultado se suma con R2
```

```
        ADDF    R0,R2           ;La ultima suma se guarda al R2
        RETS                    ;Se regresa desde el subprograma
       .END                     ;Fin del programa
```

Se necesita escribir el programa MULT.CMD para enlazador (linker) con el propósito de enlazar los programas MULT.ASM, MULTSUB.ASM, así como configurar la memoria y las direcciones de entrada y de salida.

```
/* MULT.CMD EL PROGRAMA PARA ENLAZADOR */
/* CON PROPOSITO DE ENLAZAR LOS PROGRAMAS */

        -E  BEGIN               /*Se especifica el inicio  */
        MULT.OBJ                /*El programa principal MULT*/
        MULTSUB.OBJ             /*La subrutina MULTSUB      */
        -O MULT.OUT             /*Instruccion para ligador  */
MEMORY
{
VECS:   org=0           len=0x40
SRAM:   org=0x40        len=0x3FC0      /*BUS de (16K)     */
RAM:    org=0x809800    len=0x800       /*RAM interna de 2K*/
IO:     org=0x804000    len=0x2000      /*IOSTRB de 8K     */
}
SECTIONS
{
 .data:   {} >  SRAM            /*Seccion de datos en SRAM    */
 .text:   {} >  SRAM            /*Seccion del texto en SRAM   */
 .cinit:  {} >  SRAM            /*Inicializacion de las tablas */
 .stack:  {} >  RAM             /*Sistema del STACK       */
 .bss:    {} >  RAM             /*Seccion de BSS in RAM   */
 vecs:    {} >  VECS            /*Los vectores de RESET e INTERRUPT /*
IN_PORT      804000h   : {} > IO   /*La direccion de entrada */
OUT_PORT     804002h   : {} > IO   /*La direccion de salida   */
XN_BUFFER    ALIGN(64) : {} > RAM  /*El buffer circular en RAM */
}
```

Ahora bien, si vamos a trabajar con el simulador es necesario crear los módulos con extensión .obj y .out, entonces en nuestro caso son MULT.OBJ, MULTSUB.OBJ y MULT.OUT.

Para ensamblar es necesario teclear ASM30 MULT.ASM y ASM30 MULTSUB.ASM. En el archivo se crean los módulos MULT.OBJ y MULTSUB.OBJ. En este momento podemos llamar al enlazador con el propósito de obtener el módulo MULT.OBJ que ya podemos cargar en el simulador o si trabajamos con las señales reales, en el módulo de evaluación EVM. Para obtener el módulo MULT.OUT es necesario teclear LNK30 MULT.CMD.

Si tecleamos SIM3x MULT.OUT cargamos en el simulador MULT.OUT. En una ventana también podemos cargar el programa en ensamblador si tecleamos ALT+L después FILE

y al final MULT.ASM. En la pantalla se muestran las ventanas que están en la figura 3.2. Podemos correr paso a paso el programa, si se teclea F8 o también teclear la instrucción STEP 5000 o F5.

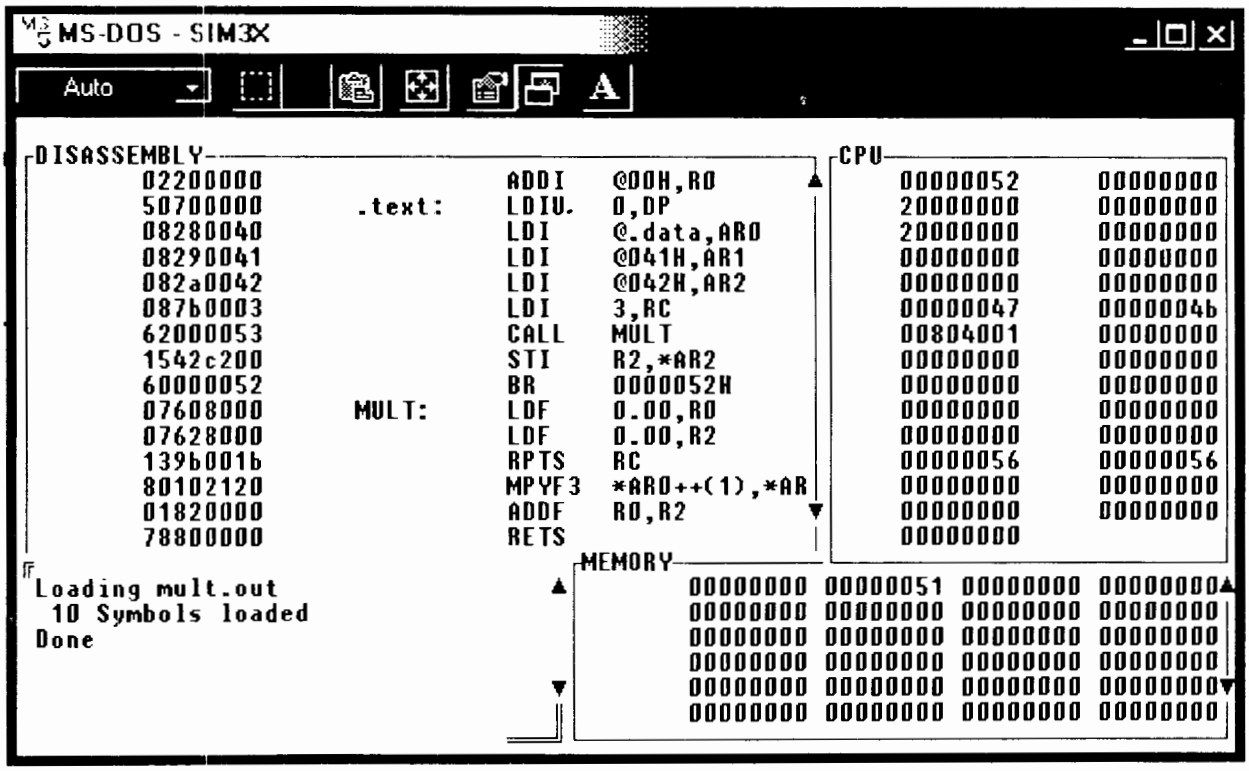

Figura 3.2: Pantalla del simulador SIM3x

3.3.2 Multiplicación de las matrices

Con cualquier editor de texto se puede escribir un programa para la multiplicación de una matriz con un vector.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 14 \\ 32 \\ 50 \end{bmatrix}$$

El programa en ensamblador se muestra a continuación.

```
;MATRIX.ASM-MULTIPLICA LA MATRIZ CON UN VECTOR EN ENSAMBLADOR
            .GLOBAL BEGIN
            .DATA                       ;Se ensambla en el seccion de datos
A           .FLOAT 1,2,3,4,5,6,7,8,9    ;Valores del matriz  A
B           .FLOAT 1,2,3                ;Valores del matriz  B
A_ADDR      .WORD A                     ;Direccion de los valores A
B_ADDR      .WORD B                     ;Direccion de los valores B
```

```
IO_OUT      .WORD 804001H              ;Direccion del puerto de salida
            .TEXT                      ;Ensamblar a la seccion de texto
BEGIN       LDI   @A_ADDR,AR0          ;AR0=Direccion de matriz A
            LDI   @B_ADDR,AR1          ;AR1=Direccion del vector B
            LDI   @IO_OUT,AR2          ;AR2=El puerto de salida
            LDI   3,R4                 ;3 -->R4
LOOPI       LDF   0,R0                 ;0 --> R0
            LDI   2,AR4                ;2 --> AR4
LOOPJ       MPYF  *AR0++,*AR1++,R1     ;A[I,J]*B[J] --> R1
            ADDF  R1,R0                ;En R0 se guarda el resultado
            DB    AR4,LOOPJ            ;Decrementa AR4 y si AR4<0 salto
            FIX   R0                   ;Se convierte R0 al numero entero
            STI   R0,*AR2              ;El resultado al IO_OUT
            LDI   @B_ADDR,AR1          ;AR1 --> B_ADDR
            SUBI  1,R4                 ;Se decrementa R4
            BNZ   LOOPI                ;Salto si R4<>0
WAIT        BR    WAIT                 ;Espera
```

En los registros auxiliares AR0, AR1 y AR2 se cargan las direcciones de las variables A, B y la dirección de puerto de salida. R4 se utiliza como contador de repetición. Tenemos R4=3, eso significa que multiplicamos tres números de la primera fila A con tres números en la columna de B y el resultado lo sumamos en R0. Las direcciones de la memoria de AR0 y AR1 se incrementan. El asterisco nos indica que se utiliza direccionamiento indirecto. La instrucción SUBI 1,R4 decrementa la variable R4. En caso de que R4=0, se salta a la etiqueta LOOPI. En el registro auxiliar AR1 se guarda la dirección donde principia el arreglo A. El resultado se guarda en el archivo OUTPUT.DAT. En el archivo OUTPUT.DAT deben aparecer los números 0000000e 00000020 00000032. Si a estos números los convertimos en decimales obtenemos 14, 32, 50. El archivo de comandos MATRIX.CMD o MULT.CMD lo podemos cambiar y utilizarlo para otros programas.

```
/*MATRIX.CMD              Archivo de los comandos              */
-E BEGIN                 /*Se especifica el inicio            */
MATRIX.OBJ               /*Se crea archivo MATRIX.OBJ         */
-O MATRIX.OUT            /*Instruccion para crear MATRIX.OUT  */
```

3.3.3 Programa con el *buffer* circular

En este ejemplo se realiza un filtro con la respuesta finita FIR. El programa FIRFIL.ASM se presenta a continuación y el programa de comandos en el archivo FIRFIL.CMD.

```
; NOMBRE DE ARCHIVO: FIRFIL.ASM
;
; A continuacion se observa
; como son organizados los datos en la memoria
```

```
; y como se cambia la memoria donde son guardadas las
; muestras de la senal x(n). Las dos ultimas columnas
; muestran como funciona el (buffering) circular en el
; registro  AR1.
;
;     la respuesta      las muestras de      las muestras de
;      al impulso         la senal       la senal despues buffering
;       +--------+       +------------+  +------------+
;       | h(N-1) |       | x[n-(N-1)] |  |    x(n)    |
;       +--------+       +------------+  +------------+.
;       | h(N-2) |       | x[n-(N-2)] |    x[n-(N-1)] |
;       +--------+       +------------+  -------------+
;           :                  :              :
;       +--------+       +------------+  +------------+
;       | h(1)   |       |   x(n-1)   |  |   x(n-2)   |
;       +--------+       +------------+  +------------+
;       | h(0)   |       |    x(n)    |  |   x(n-1)   +
;       +--------+       +------------+  +------------+
;
;       Si la subrutina filter es llamada por  primera vez,
;       la direccion de h(N-1) se guarda en el registro
;       auxiliar AR0,  y la direccion de x[n-(N-1)] en el
;       registro AR1.
;
        .global _main,fir
        .page
;
; Primeramente necesitamos definir el lugar para I/O en la memoria.
; Utilizamos el LINKER para definir las direcciones  de las secciones
; "in_port" y "out_port" en el mapa de la memoria.
;
in      .usect  "in_port",1       ;A/D la direccion de "input port"
out     .usect  "out_port",1      ;D/A la direccion  de "output port"
;
        .data
in_addr  .word  in                ;guardar 24-bit A/D direccion
out_addr .word  out               ;guardar 24-bit D/A direccion

x_n_addr .word  x_n+length-1      ;la direccion de x(n)
h_n_addr .word  h_n               ;la direccion de h(N-1)

scaler   .float  6.62             ;scaler for D/A output
;
                                  ; Los coeficientes para el filtro FIR.
                                  ; Podemos calcular en este ejemplo
                                  ; cualquier tamano de filtro hasta
                                  ; N=1024.
```

```
h_n     .float   2.2579136E-03     ; h(41)
        .float   4.1378370E-04     ; h(40)
        .float  -6.3593970E-03     ; h(39)
        .float  -4.3735630E-03     ; h(38)
        .float   9.0539033E-03     ; h(37)
        .float   1.0372631E-02     ; h(36)
        .float  -6.4953700E-03     ; h(35)
        .float  -1.1626530E-02     ; h(34)
        .float   7.7737306E-04     ; h(33)
        .float   6.4344249E-04     ; h(32)
        .float  -2.8973380E-03     ; h(31)
        .float   2.1137551E-02     ; h(30)
        .float   2.6027328E-02     ; h(29)
        .float  -3.8419060E-02     ; h(28)
        .float  -7.2032840E-02     ; h(27)
        .float   2.9972621E-02     ; h(26)
        .float   1.2341397E-01     ; h(25)
        .float   1.3980616E-02     ; h(24)
        .float  -1.5107940E-01     ; h(23)
        .float  -7.9517490E-02     ; h(22)
        .float   1.3472794E-01     ; h(21)
        .float   1.3472794E-01     ; h(20)
        .float  -7.9517490E-02     ; h(19)
        .float  -1.5107940E-01     ; h(18)
        .float   1.3980616E-02     ; h(17)
        .float   1.2341397E-01     ; h(16)
        .float   2.9972621E-02     ; h(15)
        .float  -7.2032840E-02     ; h(14)
        .float  -3.8419060E-02     ; h(13)
        .float   2.6027328E-02     ; h(12)
        .float   2.1137551E-02     ; h(11)
        .float  -2.8973380E-03     ; h(10)
        .float   6.4344249E-04     ; h(9)
        .float   7.7737306E-04     ; h(8)
        .float  -1.1626530E-02     ; h(7)
        .float  -6.4953700E-03     ; h(6)
        .float   1.0372631E-02     ; h(5)
        .float   9.0539033E-03     ; h(4)
        .float  -4.3735630E-03     ; h(3)
        .float  -6.3593970E-03     ; h(2)
        .float   4.1378370E-04     ; h(1)
h_0     .float   2.2579136E-03     ; h(0)
;
length  .set     (h_0-h_n)+1       ; length = 42 (02Ah)
                                   ; Al final es necesario reservar
                                   ; en la memoria RAM
```

```
                                    ; el lugar para los datos de las
                                    ; muestras de entrada  x(n) hasta
;
        .bss    offset,3
x_n     .usect "x_n_buff",length
;
        .page
        .text
;
_main   LDP     in_addr
        LDI     @in_addr,AR5    ;A/D direccion -> AR5
        LDI     @out_addr,AR6   ;D/A direccion -> AR6
        LDI     length,BK       ;el tamano (longitud) del filtro -> BK
        LDI     @x_n_addr,AR1   ;x(n) direccion -> AR1
        LDF     0.0,R0          ;0.0 -> R0
        RPTS    length-1
        STF     R0,*AR1--(1)%   ;R0 -> AR1 (direccion en AR1 se decrementa)
;
loop    FLOAT   *AR5,R3         ;se lee nueva muestra de x(n)
        STF     R3,*AR1++(1)%   ;R3 se guarda en el bufer circular
                                ;AR1 es ahora en x(n-(N-1))
        LDI     length-2,RC     ;length-2 se guarda en contador de repeticion
        LDI     @h_n_addr,AR0   ;la direccion de h(n) -> AR0
        CALL    fir             ;se calcula el valor de salida
        BUD     loop
        MPYF    @scaler,R0
        FIX     R0,R1           ;R1 es la forma entera para la salida
        STI     R1,*AR6
        BR      loop            ;se salta a la etiqueta loop
;
; La subrutina FIR
; Las ecuaciones que se calculan
; y(n) = h(0)*x(n) + h(1)*x(n-1) +...+ h(N-1)*x(n-(N-1))
;
; LOS REGISTROS UTILIZADOS EN EL PROGRAMA:
;
;       carga   AR0     addr of h(N-1)
;       carga   AR1     addr of x(N-1)
;       carga   RC      longitud del filtro - 2 (N-2)
;       carga   BK      longitud del filtro (N)
;       call    fir
;
; REGISTROS UTILIZADOS COMO ENTRADA: AR0, AR1, RC, BK
; REGISTROS QUE SE MODIFICAN : R0, R2, AR0, AR1, RC
; REGISTRO QUE CONTIENE EL RESULTADO: R0
; LOS CICLOS DE EJECUCION : 11 + (N-1)
;
```

```
        .global fir
        .text
;
fir     mpyf3   *AR0++(1),*AR1++(1)%,R0
;                               ; R0 = h(N-1)*x(n-(N-1))
        ldf     0.0,R2          ; se inicializa R2
        rpts    RC              ; se inicializa el contador de la repeticion
        mpyf3   *AR0++(1),*AR1++(1)%,R0
    ||  addf3   R0,R2,R2        ; la instruccion paralela
                                ; multiplica y suma
                                ; h(N-1-i)*x(n-(N-1-i))
        addf    R0,R2,R0        ; se suma el ultimo producto
        retsu

/*************************************************************************/
/* FILFIR.CMD -  EL ARCHIVO DE COMANDOS PARA EL ENLAZADOR                  */
/*                                                                       */
/* Las instrucciones: lnk30 <obj archivo...>  -o <out archivo            */
/*                    -m <map archivo>  FILFIR.cmd                       */
/*                                                                       */
/*    Descripcin: Este archivo trae las instrucciones para ligador y    */
/*                 compilador de TMS320C30 C                             */
/*************************************************************************/
-c                                      /* LINK UTILIZANDO C            */
-i c:\c30tools                          /* DIRECTORIO DE LIBRERIA       */
-l rts.lib                                                              */
-l bus.lib                              /* UTILITARIO DE BUS LIBRERIA   */
-l serial.lib                           /* SERIAL PUERTO LIBRERIA       */
-l timers.lib                                                           */
-l dma.lib                              /* UTILITARIO DE DMA LIBRERIA   */
                                                                        */
vectors.obj                             /* LOS VECTORES DE INTERRUPCION */
ints.obj                                /* LA RUTINA C DE INTERRUPCION  */

/* MAPA DE LA MEMORIA */

MEMORY
{
   VECS:    org = 0         len = 0x40      /* EL VECTOR DE INTERRUPCIONES*/
   ROM:     org = 0x40      len = 0x3fc0    /* C30 EVM 16K RAM            */
   RAM0:    org = 0x809800 len = 0x400      /* EL BLOQUE DE RAM 0         */
   RAM1:    org = 0x809c00 len = 0x400      /* EL BLOQUE DE RAM 1         */

   IOM:     org = 0800000h len = 02000h     /* -MSTRB I/O                 */
   IO:      org = 0804000h len = 02000h     /* -IOSTRB I/O                */

}
```

```
/* COLOCACIJN DE LAS SECCIONES O BLOQUES EN LA MEMORIA */

SECTIONS
{
   vectors: {} > VECS            /* EL VECTOR DEL INTERRUPCION        */
   .text:   {} > ROM             /* EL CODE                           */
   .cinit:  {} > ROM             /* LAS TABLAS DE INICIALIZACION      */
   .data:   {} > ROM             /* DATOS INICIALIZADOS SOLO DE .ASM  */
   .stack:  {} > RAM0            /* LA SISTEMA DE STACK               */
   .bss:    {} > RAM1
   x_n_buff  align(64) : {} > RAM1   /* circ.buffer en RAM0           */
   in_port   0804000h  : {} > IO     /* A/D "input port" direccion    */
   out_port  0804001h  : {} > IO     /* D/A "output port" direccion   */
}
```

Capítulo 4

Prácticas con EVM y Simulador de C30

4.1 Solo cargar y enviar

Ejemplo 1

Escribir los siguientes programas en el directorio $C:\backslash EVM30\backslash CARGENVI>$. Conectar la entrada del módulo de evaluación con el generador de señales y la salida con el analizador de espectro o con el osciloscopio. El programa sólo carga la señal del generador y sin procesarlo lo envia al osciloscopio. Para correr el módulo de evaluación es necesario primero compilar los programas para obtener los módulos con la extención .obj. El ligador hace los módulos con la extensión .obj el módulo ejecutable .out, que se carga al microcontrolador de TMS320C30. Los comandos para obtener el módulo ejecutable para el microcontrolador son los siguientes:

```
asm30.exe   inievm.asm
asm30.exe   readsend.asm
asm30.exe   vectors.asm
asm30.exe   enviar.asm
lnk30.exe   readsend.cmd
evm30.exe   readsend.out
```

Después de estos comandos, si los programas están correctos, en la plantilla del analizador de espectro se obtiene la gráfica que se muestra en la figura 4.1. La gráfica es la respuesta de la carta de módulo de evaluación EVM. La carta del módulo de evaluación incluye un filtro paso bajas para evitar el translape de los espectros (*antialiasing*), y un filtro paso altas para corregir la compensación de los amplificadores.

```
*****************************************
**********INIT.CMD***********************
*****************************************
;Mapa de la memoria del EVM TMS320C30
;
reset
MR
```

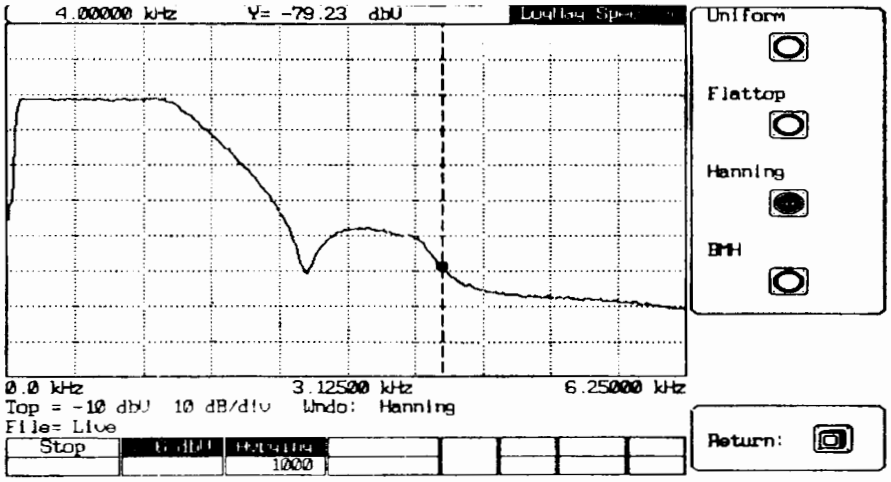

Figura 4.1: La respuesta de la carta del módulo de evaluación

```
;
MA 0x000000,0x004000,RAM      ; SRAM
MA 0x804000,0x001000,RAM      ; COMPORT
MA 0x809800,0x000400,RAM      ; RAM0
MA 0x809c00,0x000400,RAM      ; RAM1
;
;
;
MA 0x808000,0x000010,RAM      ; DMA
MA 0x808020,0x000010,RAM      ; TIMER0
MA 0x808030,0x000010,RAM      ; TIMER1
MA 0x808040,0x000010,RAM      ; SPORT0
MA 0x808050,0x000010,RAM      ; SPORT1
MA 0x808060,0x000001,RAM      ; XBUSCTL
MA 0x808064,0x000001,RAM      ; PBUSCTL
;
MAP ON
DASM PC
echo EVM ESTA INICIALIZADO

*************************************************
**************INIEVM.ASM*************************
*************************************************
            .global _main,inievm
timer0      .word 808020h
SPGC0       .word 0e970300h

            .text
inievm
            LDI     1h,R6
            LDI     @timer0,AR4
```

```
        STI     R6,*+AR4(8)
        LDI     2C1h,R6
        STI     R6,*AR4
        LDI     2h,IOF
        LDI     111h,R6
        STI     R6,*+AR3(2)
        STI     R6,*+AR3(3)
        LDI     @SPGC0,R6
        STI     R6,*AR3
        LDI     0,R6
        STI     R6,*+AR3(8)
        LDI     6h,IOF
        LDI     0,IF
        OR      10h,IE
        LDI     3h,R5
        OR      2000h,ST
        IDLE
        LDI     1A34H,R5      ;se ajusta la frecuencia de muestreo
        IDLE
        LDI     3h,R5
        IDLE
        LDI     2A7h,R5       ;se quita DP y HP de la carta de EVM
        IDLE
        RETS
        .end

;*************************************************
;*****************READSEND.ASM*****************
;*************************************************
;ASM30 readsend.asm
;LNK30 readsend.cmd
;EVM30 readsend.out
;*************************************************
;AR3      - el puert serial
;R2,R5,R6 - en estos registros se cambian los valores
;R5       - registro donde se almacenan las muestras de
;           entrada y de salida
;*************************************************

        .global inievm, serialp, _main

serialp .word   808040h       ;808040h = la direccion del puerto serial

_main:  LDI     @serialp,AR3  ;AR3=808040h-la direccion del puerto
        CALL    inievm        ;inicializacion de  EVM.
        AND     0h,ST         ;no se permite inerrupcion.
        LDI     010H,IE       ;se permite interrupcion de transmision
```

```
                                ;del puerto serial num. 0
        LDF     0.0,R6
        FIX     R6,R5           ;0--->R5
Loop:   LSH     2,R5            ;se corre contenido de R5, 2 bits a la
                                ;izquierda
        IDLE                    ;el procesador espera para interrupcion
        AND     0h,ST           ;no se permite interrupcion


        BRD     Loop            ;se salta a la etiqueta LOOP despues de
                                ;ejecutar tres siguientes instrucciones

        FLOAT   R5,R2
        NOP

        FIX     R2,R5             ;La conversion de punto flotante a
                                  ;punto fijo


*******************************************************
*****************READSEND.CMD**************************
*******************************************************

inievm.obj
vectors.obj
enviar.obj
readsend.obj

-e _main
-o readsend.out
-m readsend.map

MEMORY
{
    VECS  : org = 0        , len = 0x40
    ROM   : org = 0x40     , len = 0x3fc0
    RAM   : org = 0x809800, len = 0x800
}
SECTIONS
{
vecs:   load = 000000h
.text : {} > ROM
.cinit: {} > ROM
.stack: {} > RAM
.bss  : {} > RAM
}
```

```
*************************************************
***************VECTORS.ASM***********************
*************************************************

        .global RESET,INT0,INT1,INT2,INT3,_main
        .global XINT0,RINT0,XINT1,RINT1,TINT0,TINT1,DINT
        .global _c_int05,_c_int99

;Vectores de interrupcion:
        .sect   "vecs"          ;esta seccion se escribe a la direccion 0
RESET   .word  _main            ;vector RESET
INT0    .word   _c_int99        ;vector INT0
INT1    .word   _c_int99        ;vector INT1
INT2    .word   _c_int99        ;vector INT2
INT3    .word   _c_int99        ;vector INT3
XINT0   .word   _c_int05        ;vector de puerto serial numero cero XMT
RINT0   .word   _c_int99        ;vector de puerto serial numero cero  RCV
XINT1   .word   _c_int99        ;vector de puerto serial numero uno   XMT
RINT1   .word   _c_int99        ;vector de puerto serial numero uno   RCV
TINT0   .word   _c_int99        ;vector de timer numero 0
TINT1   .word   _c_int99        ;vector de timer numero 1
DINT    .word   _c_int99        ;vector DMA
        .space  20              ;espacio reservado
        .space  32              ;espacio de vector TRAP
        .end

*************************************************
*************ENVIAR.ASM**********************
*************************************************

        .global _c_int05,_c_int99,serialp
        .text

_c_int05:                       ;XINT0  interrupcion del puerto serial
                                ;para transmision
        LDI     @serialp,AR3    ;en AR3 es la direccion del puerto serial
        STI     R5,*+AR3(8)     ;se transmite la palabra de R5 a puerto
                                ;serial
        LDI     *+AR3(12),R5    ;se lee la palabra de puerto serial

;Dos bits menos significativos que se reciben de AIC no traen la
;informacion sobre la muestra desde  A/D.
;Estos bits son siempre igual a cero. Para obtener los valores
;correctos de la muestra de entrada es necesario quitar dos bits
```

```
;menos significativos. Esto hacen los instrucciones siguientes.

        LSH      16,R5      ;corrimiento del <R5> a la izguierda de 16 bits
        ASH     -18,R5      ;corrimiento del <R5> a derecho de 18 bits

        RETI                ;regreso de interupcion.

;Si se hace la interrupcion del puerto de transmision el programa salta a un
;lazo indefinido. Si se hace la interrupción, el programa salta a un lazo in-
;definido. En el registro R7 se alternan en cada paso los valores 0h y fffh.

_c_int99:                          ;se idican todos los interrupciones no deseados
CHYBA:      LDI       C,R7
            LDI       1,R7
            BR        CHYBA
            .end
```

4.2 Filtro FIR de segundo orden

Exeplo 2

Para el módulo de evaluación escribir el programa para el filtro digital con la respuesta finita a impulso de segundo orden, que se muestra en la figura 4.2. La frecuencia del muestreo es $f_m = 8013$ Hz.

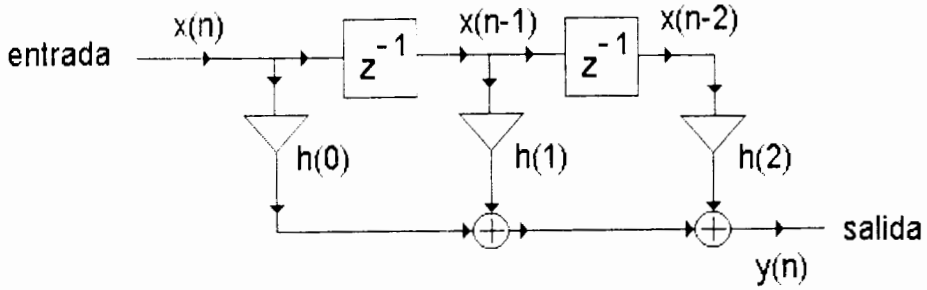

Figura 4.2: Filtro FIR de segundo orden orden

La ecuación que se programa, se obtiene del circuito que se muestra en la figura 4.2, y es la siguiente:

$$y(n) = h(0) * x(n) + h(1) * x(n-1) + h(2) * x(n-2)$$

4.2.1 Simulación del filtro FIR con el simulador

En el directorio $C:\backslash SIM30\backslash FIR2ORD >$ se escribe el programa *fir2ord.asm*. Ensamblador ASM30.EXE convierte el programa *fir2ord.asm* en el módulo de códigos con extención .obj. El módulo *fir2ord.obj* se obtiene tecleando

asm30.exe fir2ord.asm

El ligador une las programas con la extención .obj en un módulo ejecutable para simulador con la extención .out. El programa *siminit.cmd* debe ser en el mismo archivo y es utilizado automaticamente en el proceso al llamar *lnk30.exe*. El ligador se invoca mediante el comando

lnk30.exe fir2ord.obj

La simulación del filtro empieza si se teclea

sim30.exe a.out

Para correr el programa ejecutable, de la ventana de comandos, se teclea run 5000 y enter o F5 y después de algunos segundos teclear Esc. El programa calcula la respuesta a un impulso unitario. Es necesario escribir impulso unitario en el archivo *input.dat* Este archivo de entrada *input.dat* fue determinado en el programa *siminit.dat*. El resultado en la forma hexadecimal se guarda en el archivo *output.dat* como se determinó en el programa *siminit.cmd*. Si se necesita el resultado en la forma decimal se utiliza el comando

hex2dec.exe output.dat outputde.dat

```
*************************************************
*****************FIR2ORD.ASM*********************
*************************************************
;          FILTRO FIR DE SEGUNDO ORDEN
;************************************************
;************************************************
;       el programa se asemblea con:
;       ASM30 fir2ord.asm
;
;       para el ligador se utiliza:
;       LNK30 fir2ord.obj
;
;       el programa se simula mediante:
;       SIM3    a.out
;
;************************************************************
;
;   EQUATION  y(n) = h(0)*x(n) + h(1)*x(n-1) + h(2)*x(n-2)
;
;************************************************************

;    ARGUMENTOS   |       FUNCIONES
; ------------------------------------------
;       R0        |       x(n)
;       R1        |       x(n-1)
;       R2        |       x(n-2)
```

```
;       h0          |       h(0)
;       h1          |       h(1)
;       h2          |       h(2)
;
;       AR1         |   la direccion del puerto de entrada
;       AR2         |   la direccion del puerto de salida
;
;       R0,R1,R2,R3,R6,R7 - registros usados
;
************************************************************

        .data

inaddr  .word   804000h     ;inaddr  - direccion de entrada
outaddr .word   804001h     ;outaddr - direccion de salida

h0      .float  0.0462      ;set h(0)
h1      .float  0.9076      ;set h(1)
h2      .float  0.0462      ;set h(2)
        .text
        ldi     @inaddr,AR1
        ldi     @outaddr,AR2
        LDF     0.0,R1      ;inicializa R1
        LDF     0.0,R2      ;R2
        LDF     0.0,R7      ;R7

Loop:   LDI     *AR1,R6     ;las muestras de entrada x(n)--->R0
        FLOAT   R6,R0       ;la muestra x(n)) se convierte a
                            ;punto flotante ---> R0
        MPYF    @h2,R2      ;h(2)*x(n-2)--->R2
        ADDF    R2,R7       ;h(2)*x(n-2)+0--->R7
        LDF     R1,R2       ;x(n-1)--->x(n-2)
        MPYF    @h1,R1      ;h(1)*x(n-1)--->R1
        ADDF    R1,R7       ;h(1)*x(n-1)+h(2)*x(n-2)
        LDF     R0,R1       ;x(n)--->x(n-1)
        MPYF    @h0,R0      ;h(0)*x(n)--->R0
        ADDF    R0,R7       ;h(0)*x(n)+h(1)*x(n-1)+h(2)*x(n-2)=y(n)
        FIX     R7,R3       ;n\'umero de punto flotante se convierte
                            ;a n\'umero de punto fijo ---> R3
        STI     R3,*AR2     ;y(n)--->salida
        LDF     0.0,R7      ;0-->R7
        BR      Loop        ;salto a la etiqueta Loop


****************************************
**************SIMINIT.CMD***************
****************************************
```

```
;*************************************
;       El archivo para el ligador
;*************************************

ma 0x0,0x1000,ram
ma 0x808000,0x10,ram            ;memoria 0x808010 a 0x80801F reserveda
ma 0x808020,0x400,ram
ma 0x809800,0x800,ram

ma 0x804000,0x1,ioport          ;direccion 0x804000,0x1 puerto de entrada
ma 0x804001,0x1,ioport          ;direccion 0x804001,0x1 puerto de salida

mc 0x804000,input.dat,read      ;se especifica el archivo de entrada
                                ;input.dat, puerto 0x804000, donde se
                                ;guardan las muestras de entrada

mc 0x804001,output.dat,write    ;se especifica el archivo de salida
                                ;output.dat, puerto 0x804001, donde
                                ;se guardan las muestras de resultado
map on

*****************************************
**************INPUT.DAT******************
*****************************************
0x00800000     ; Impulso unitario
0x00000000     ; Es necesario escribir por los
0x00000000     ; menos cien ceros
0x00000000
0x00000000
0x00000000
.........
.........
.........
```

4.2.2 Simulación del filtro mediante MATLAB

Mediante el MATLAB calcule los valores del filtro FIR de segundo orden para la frecuencia del corte ω_1=0.5. Calcule la respuesta del filtro FIR a un impulso. El archivo *input.dat* tiene impulso unitario de tamaño en el punto flotante 8388608. En el archivo *output.dat* se encuentra la respuesta a impulso. Si se dividen los valores en el archivo entre el número 8388608 se obtienen los números fraccionarios. Grafique la respuesta a impulso y compárela con la gráfica obtenida mediante el simulador.

```
clear
************************************************
*********Filtroro FIR de segundo orden**********
************************************************
```

```
wn=0.5
b=fir1(2,wn)    %la frecuencia del corte wn debe ser
                %entre 0 < wn < 1.0, with 1.0
                %la frecuencia del muestreo es 2.0

N=128
step=1/N;
x=0:step:((N/2-1))/N;
h=freqz(b,1,N/2);
H=20*log10(abs(h));
figure(1);
subplot(2,1,1);
plot(x,H,'r'),title('Matlab'),
ylabel('-->  A(f/fv) [dB]'),
xlabel('-->  f/fv [-]'),
grid;

*****************************************
***************Simulador*****************
*****************************************

load 'outputde.dat' -ascii;  %carga la respuesta del filtro

ir_u=outputde(1:N);          %selecciona N muestras
                             %(de primero a entero)

ji=8388608;                  %el tamano del impulso unitario
i_r=ir_u/ji;                 %se normaliza la respuesta del impulso
hs = fft(i_r);
hs=hs(1:N/2);
hs=abs(hs);
Hs=20*log10(hs);

figure(1);set(1,'name','La respuesta en la frecuencia')
subplot(2,1,2);
plot(x,Hs),
title('Simulador'),
ylabel('-->  A(f/fv) [dB]'),
xlabel('-->  f/fv [-]'),
grid,;

*****************************************
**************Comparacion****************
*****************************************

figure(2);set(1,'name','Comparacion')
plot(x,Hs,x,H,'+r'),
```

```
title('+++++ Matlab      --- Simulador'),
ylabel('-->  A(f/fv) [dB]'),
xlabel('-->  f/fv [-]'),
grid,;
```


4.3 Filtro de onda de tercer orden

Ejemplo 3:

Para el módulo de evaluación escribir el programa para el filtro digital de onda de tercer orden, que se muestra en la figura 4.3. El filtro es de Cauer, el rizo en el paso de banda es 1 dB, la frecuencia del corte es 1.422 kHz. En la frecuencia 2.584 kHz la atenuación es 24 dB. La frecuencia del muestreo es $f_m = 8013$ Hz.

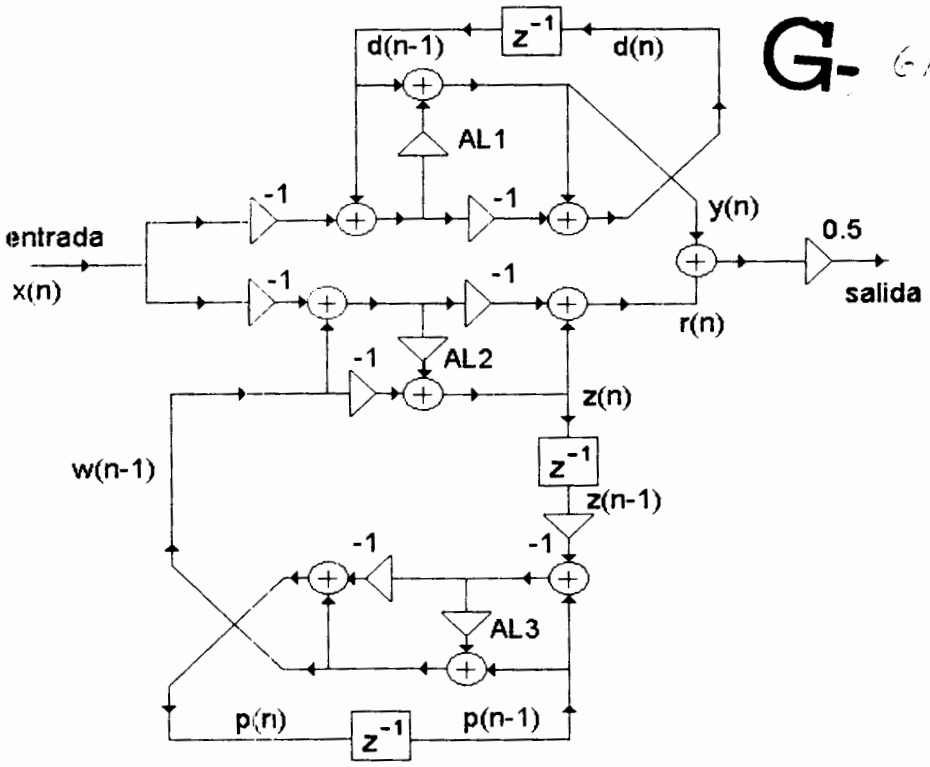

Figura 4.3: El filtro paso bajas de onda de tercer orden

Mediante el MATLAB se obtienen los polos y ceros de la función de transferencia H(s):
Polos:

$$a_0 = 0.399896 \qquad a_1 \pm jb_1 = 0.297568 \pm j0.711237$$

Ceros:

$$z_0 = -1 \qquad z_1 = \pm 0.840788$$

De la estructura en la figura 4.3 se pueden escribir las ecuaciones siguientes:

$$d(n) = AL1d(n-1) + (1-AL1)x(n) \qquad y(n) = (1+AL1)d(n-1) - AL1x(n)$$
$$z(n) = (AL2-1)w(n) - AL2x(n) \qquad r(n) = (AL2-2)w(n) + (1-AL2)x(n)$$
$$w(n) = (a+AL3)p(n-1) - AL3z(n-1) \qquad p(n) = AL3p(n-1) + (1-AL3)z(n-1)$$

Los coeficientes de los multiplicadores AL1, AL2 y AL3 son:

$$AL1 = 0.399897 \qquad AL2 = 0.405585 \qquad AL3 = 0.373285$$

En el archivo

```
C:\SIM30\WDF3ORD>
```

escriba los programas *wdf3ord.asm, vectors.asm, inievm.asm, wdf3ord.cmd, init.cmd y transmis.asm* y compilelos con el programa ejecutable asm30.exe. Despúes de la compilación se obtienen los módulos objetos *wdf3ord.obj, vectors.obj, inievm.obj y transmis.obj.* El ligador crea para el módulo de evaluación de todos los archivos objetos, el archivo ejecutable wdf3ord.out, pero sólo en este caso, si hay en el mismo archivo el programa init.cmd. Si tenemos en el archivo el modulo ejecutable wdf3ord.out podemos conectar la entrada del microcontrolador con un generador de funciones y la salida del micro con un analizador de espectro. Con el comando

evm30.exe *wdf3ord.out*

se carga el programa ejecutable al micro y si se tecla F5 se obtiene la gráfica en la pantalla del analizador de espectro que se muestra en la figura 4.4

```
************************************************
***************INIEVM.ASM***********************
************************************************
        .global _main,inievm
timer0  .word       808020h
SPGC0   .word       0e970300h

        .text
inievm
        LDI         1h,R6
        LDI         @timer0,AR4
        STI         R6,*+AR4(8)
        LDI         2C1h,R6
        STI         R6,*AR4
        LDI         2h,IOF
        LDI         111h,R6
        STI         R6,*+AR3(2)
        STI         R6,*+AR3(3)
        LDI         @SPGC0,R6
        STI         R6,*AR3
        LDI         0,R6
```

```
STI         R6,*+AR3(8)
LDI         6h,IOF
LDI         0,IF
OR          10h,IE
LDI         3h,R5
OR          2000h,ST
IDLE
LDI         1A34H,R5
IDLE
LDI         3h,R5
IDLE
LDI         2A7h,R5
IDLE
RETS
.end
```

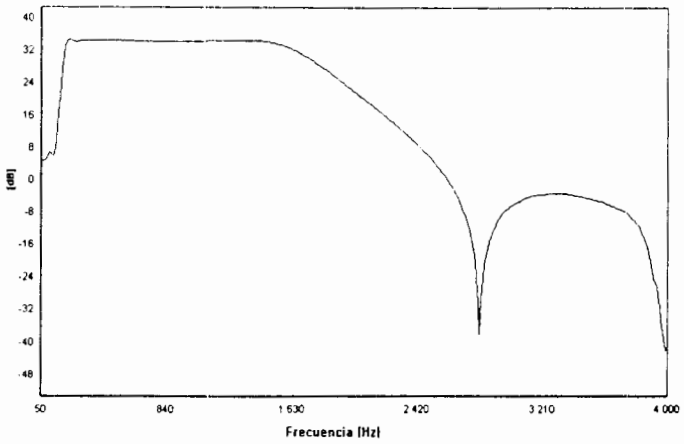

Figura 4.4: El módulo de la función de transferencia obtenida mediante el analizador del espectro.

```
**********************************************************
**********************WDF3ORD.CMD*************************
**********************************************************
wdf3ord.obj
inievm.obj
vectors.obj
transmis.obj

-e _main
-o wdf3ord.out
-m wdf3ord.map
```

```
MEMORY
{
    VECS  : org = 0        , len = 0x40
    ROM   : org = 0x40     , len = 0x3fc0
    RAM   : org = 0x809800, len = 0x800
}

SECTIONS
{
.text : {} > ROM
.cinit: {} > ROM
.stack: {} > RAM
.bss  : {} > RAM

}
```

```
************************************************
*****************INIT CMD***********************
************************************************

;El mapa de la memoria del modulo de evaluacion de TMS320C30
;
reset
MR
;
MA 0x000000,0x004000,RAM        ;SRAM
MA 0x804000,0x001000,RAM        ;COMPORT
MA 0x809800,0x000400,RAM        ;RAM0
MA 0x809c00,0x000400,RAM        ;RAM1
;
;El mapa de la memoria de la unidad periferica.
;
MA 0x808000,0x000010,RAM        ;DMA
MA 0x808020,0x000010,RAM        ;TIMER0
MA 0x808030,0x000010,RAM        ;TIMER1
MA 0x808040,0x000010,RAM        ;SPORT0
MA 0x808050,0x000010,RAM        ;SPORT1
MA 0x808060,0x000001,RAM        ;XBUSCTL
MA 0x808064,0x000001,RAM        ;PBUSCTL
;
MAP ON
DASM PC
echo EVM esta inicializado
```

```
*****************************************************
***************TRANSMIS.ASM**************************
*****************************************************

        .global _c_int05,_c_int99,serialp
        .text

_c_int05:                       ;XINT0  interrupcion del puerto serial
                                ;para transmision
        LDI     @serialp,AR3    ;en AR3 es la direccion del puerto serial
        STI     R5,*+AR3(8)     ;se transmite la palabra de R5 a puerto
        LDI     *+AR3(12),R5    ;se lee la palabra de puerto serial

;Dos bits menos significativos que se reciben de AIC no traen la
;informacion sobre la muestra desde  A/D.
;Estos bits son siempre equal a cero. Para obtener los valores
;correctos de la muestra de entrada es necesario quitar dos bits
;menos significativos. Esto hacen los instrucciones siguientes.

        LSH      16,R5          ;corrimiento del <R5> a la izquierda de 16 bits
        ASH     -18,R5          ;corrimiento del <R5> a derecho de 18 bits
        RETI                    ;regreso de interupcion.

;Si se hace la interrupcion del puerto de transmision el programa salta a un
;lazo indefinido. Si se hace la interrupcion el programa salta a un lazo in-
;definido. En el registro R7 se alternan en cada paso los valores 0h y fffh.

_c_int99:                       ;se idican todas los interrupciones no deseadas
ERROR:    LDI     0,R7
          LDI     1,R7
          BR      ERROR
          .end

*************************************************
***************VECTORS.ASM***********************
*************************************************

        .global RESET,INT0,INT1,INT2,INT3,_main
        .global XINT0,RINT0,XINT1,RINT1,TINT0,TINT1,DINT
        .global _c_int05,_c_int99

;Vectores de interrupcion:
        .sect   "vecs"          ;esta seccion se escribe a la direccion 0
RESET   .word  _main            ;vector RESET
INT0    .word   _c_int99        ;vector INT0
INT1    .word   _c_int99        ;vector INT1
INT2    .word   _c_int99        ;vector INT2
```

```
INT3      .word   _c_int99      ;vector INT3
XINT0     .word   _c_int05      ;vector de puerto serial numero 0 XMT
RINT0     .word   _c_int99      ;vector de puerto serial numero 0  RCV
XINT1     .word   _c_int99      ;vector de puerto serial numero 1   XMT
RINT1     .word   _c_int99      ;vector de puerto serial numero 1   RCV
TINT0     .word   _c_int99      ;vector de timer numero 0
TINT1     .word   _c_int99      ;vector de timer numero 1
DINT      .word   _c_int99      ;vector DMA
          .space  20            ;espacio reservado
          .space  32            ;espacio de vector TRAP
          .end

*************************************************************************
**************************WDF30RD.ASM************************************
*************************************************************************

          .global _main,inievm,serialp
x         .float    0
d         .float    0
y         .float    0
z         .float    0
p         .float    0
d_addr    .word     d
z_addr    .word     z
p_addr    .word     p

A1c       .float    0.399897
          .float    1.399897
          .float    0.600103
A2c       .float    0.405535
          .float   -0.594415
          .float   -1.594415
          .float    0.594415
A3c       .float    0.373285
          .float    1.373285
          .float    0.626715

A1        .word     A1c
A2        .word     A2c
A3        .word     A3c

serialp   .word     808040h
_main     LDI       @serialp,AR3
          LDI       2,IR0
          LDI       3,IR1
          CALL      inievm
          AND       0h,ST
```

```
        LDF     0,R6
        LDI     010H,IE
        FIX     R6,R5
loop:   LSH     2,R5
        IDLE
        AND     0h,ST
        FLOAT   R5,R7
        STF     R7,@x
        LDI     @d_addr,AR1
        LDI     @A1,AR3
        CALL    GAMA_2
        STF     R6,@y
        LDF     @z,R7
        LDI     @p_addr,AR1
        LDI     @A3,AR3
        CALL    GAMA_2
        LDF     R6,R5
        LDF     @x,R7
        LDI     @A2,AR3
        LDI     @z_addr,AR1
        CALL    GAMA_5
        LDF     @y,R7
        BRD     loop
        ADDF    R7,R6
        MPYF    0.5,R6
        FIX     R6,R5

GAMA_2: MPYF3   *+AR3,*AR1,R6
        MPYF3   *AR3,R7,R0
        SUBF     R0,R6
        MPYF3   *AR3,*AR1,R0
        MPYF3   *+AR3(IR0),R7,R1
        ADDF    R1,R0
        STF     R0,*AR1
        RETS

GAMA_5: MPYF3   *+AR3,R5,R0
        MPYF3   *AR3,R7,R1
        SUBF    R1,R0
        STF     R0,*AR1
        MPYF3   *+AR3(IR0),R5,R0
        MPYF3   *+AR3(IR1),R7,R6
        ADDF    R0,R6
        RETS
```

4.3.1 Comparación de los resultados del SIM30 y EVM30

Ejemplo 4:

En el directorio $C:\backslash SIM30\backslash WDF3ORD>$ se encuentran los archivos *simini.cmd, input.dat, output.dat, hex2dec.exe, sim30.exe y wdf3ord.out.* Simulador de C30 se activa tecleando

sim3.exe wdf3ord.out.

Si se teclea **F5** o **run** se hace la simulación del programa para el filtro digital de onda de tercer orden. Tecleando el botón **Esc** se interrupe la simulación. Con **quit** se sale del simulador. Se obtuvieron los datos en la forma hexadecimal de TI en el archivo *output.dat.* Para convertirlas en la forma decimal se utiliza el programa HEX2DEC.EXE, tecleando

hec2dec.exe output.dat outputde.dat.

El archivo *output.dat* contiene los datos de la respuesta a impulso normalizada mediante el coeficiente $ji = 8388608$

4.3.2 La respuesta al impulso obtenida mediante MATLAB

Otro tarea es analizar los resultados mediante el programa MATLAB. En el archivo *wdf30rd.m* escriba el programa siguiente:

```
>>N=128;
>>step=1/N;
>>x=0:step:((N/2-1))/N;
>>load 'outputde.dat' -ascii;   %se carga la respuesta del filtro a un
                                %impulso
>>ir_u=outputde(1:N);           %se seleccionan N muestras de la respuesta
>>ji=8388608;
>>i_r=ir_u/ji;
>>hs=fft(i,r);                  %se calcula la amplitud de la respuesta
>>hs=hs(1:N/2);
>>hs=abs(hs);
>>Hs=20*log10(hs);
>>figure(1);set(1,'nombre','Amplitud de la respuesta')
>>plot(x,Hs),
>>title('Simulador'),
>>ylabel(-->A(f/fv) [dB]'),
>>xlabel('-->f/fv [-]'),
grid,;
```

El resultado obtenido mediante MATLAB se muestra en la figura 4.5.

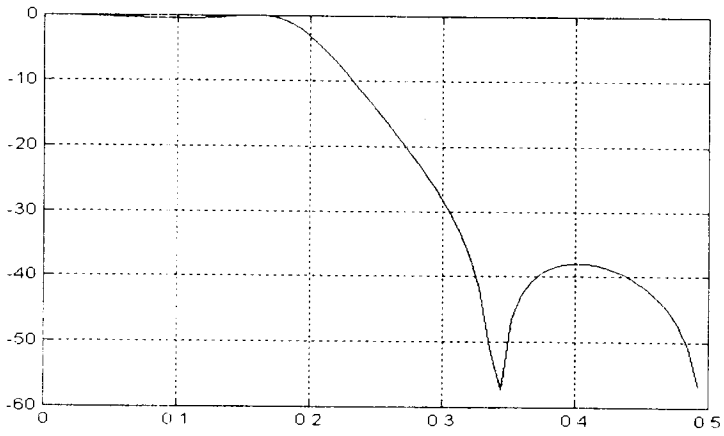

Figura 4.5: La amplitud de la respuesta al impulso

4.4 Filtro de onda de sexto orden

Ejeplo 5:

Para el módulo de evaluación, escribir el programa para el filtro digital de onda de sexto orden, que se muestra en la figura 4.6. El filtro es de Butterworth. El rizo en el paso de banda es de 3 dB, la frecuencia del corte normalizada es $\omega = 1$ rad/sec. La frecuencia del muestreo es $f_m = 8013$ Hz.

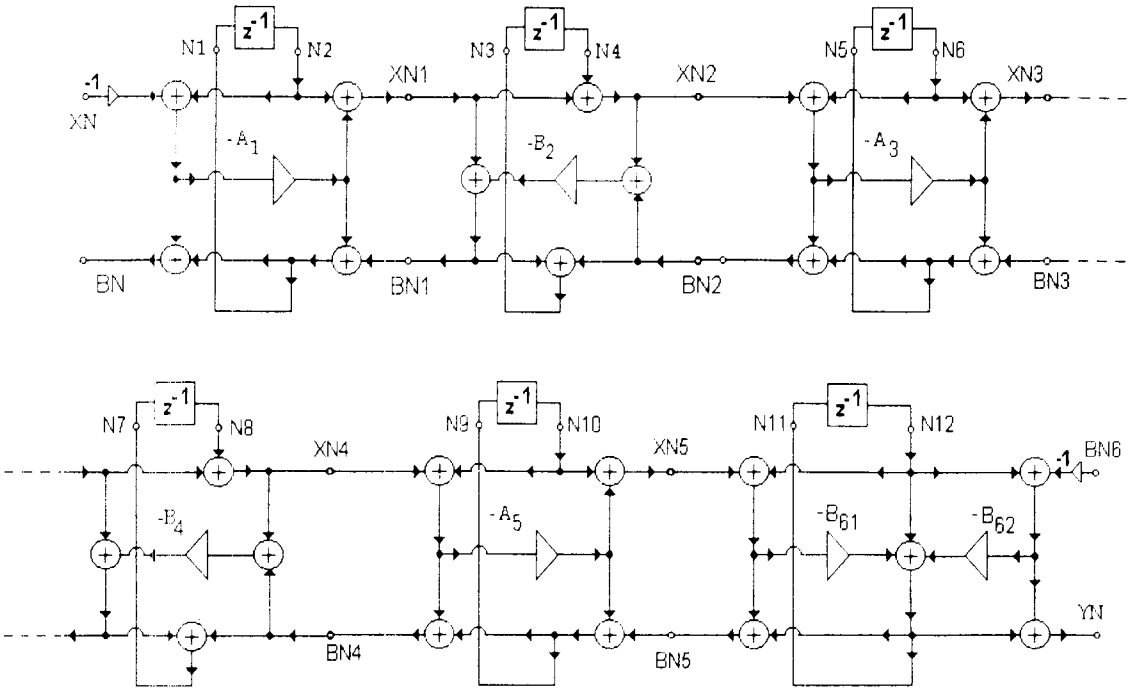

Figura 4.6: El filtro paso bajas de onda de sexto orden

Los coeficientes del filtro pasa bajas de sexto orden son:

$$\begin{array}{ll} A_1 = 0.658935 & B_2 = 0.317843 \\ A_3 = 0.199797 & B_4 = 0.176550 \\ A_5 = 0.231596 & \\ B_{61} = 0.512134 & B_{62} = 0.487865 \end{array}$$

Los ecuaciones obtenidos del circuito en la figura 4.6 para el simulalador y el módulo de evaluación de C30 son las siguientes:

$$\begin{array}{l} XN1 = A1.XN + N2 - A1.N2 \\ XN2 = XN1 + N4 \\ XN3 = -A3.XN2 + N6 - A3.N6 \\ XN4 = XN3 + N8 \\ XN5 = -A5.XN4 + N10 - A5.N10 \\ \hline BN5 = XN5 - B61.XN5 - B61.N12 \\ BN4 = XN4 - A5.XN4 - A5.N10 + N10 + BN5 \\ BN3 = XN3 - B4.XN4 - B4.BN4 \\ BN2 = XN2 - A3.XN2 - A3.N6 + N6 + BN3 \\ BN1 = XN1 - B2.XN2 - B2.BN2 \\ \hline N1 = -A1.Xn - A1.N2 + BN1 \\ N3 = BN1 + BN2 \\ N5 = -A3.XN2 - A3.N6 + BN3 \\ N7 = BN3 + BN4 \\ N9 = -A5.XN4 - A5.N10 + BN5 \\ N11 = BN5 - B62.N2 + N12 \\ \hline YN = -B62.XN5 - B62.N12 \end{array}$$

En el directorio $C:\backslash EVM30\backslash WDF6ORD>$ escriba los archivos *inievm.asm,init.cmd, transmis.asm, vectors.asm, wdf6ord.asm y wdf6ord.cmd.* Para obtener los módulos de objeto de cada archivo tecle los comandos siguiente:

asm30.exe	**wdf6ord.asm**
asm30.exe	**vectors.asm**
asm30.exe	**inievm.asm**
asm30.exe	**transmis.asm**

Estos comandos crearon en el archivo $C:\backslash EVM30\backslash WDF6ORD>$ los módulos *inievm.obj, transmis.obj, vectors.obj y wdf6ord.obj.* Mediante el ligador se obtiene el módulo ejecutable *wdf6ord.out*, después de teclear

lnk30.exe wdf6ord.cmd

Este módulo ejecutable se carga al módulo de evaluación mediante el comando

evm30.exe wdf6ord.out

El *debuger* llama a la plantalla de la ventana depurador que nos permite con F5 correr el programa ejecutable. El programa se puede correr tambien con el comando **run 5000** enter. Si el programa corre con F5 es necesario, en la ventana de comandos, escribir después de algunos segundos quit. Antes es necesario conectar la entrada del módulo de evaluación con generador de señales que produce el ruido blanco. La salida del módulo de evaluación se conecta con el analizador de espectro. Si no hay errores en los programas, debe salir en la pantalla del analizador de espectro la gráfica que se muestra en la figura 4.7

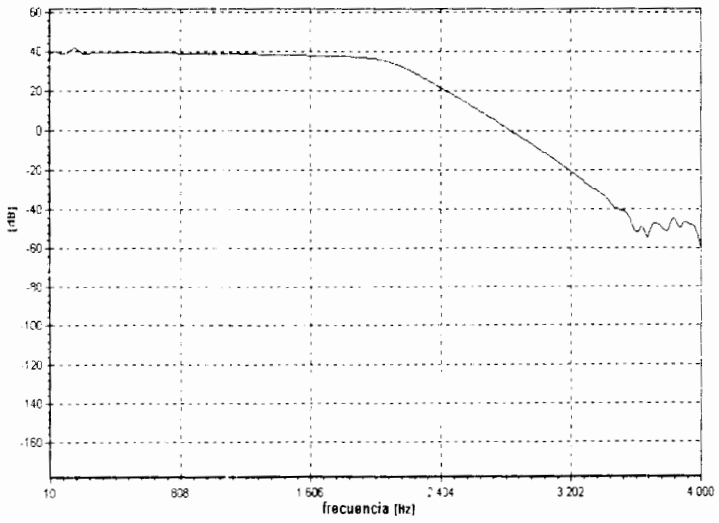

Figura 4.7: El módulo de la función de transferencia obtenida mediante el analisador del espectro

```
*************************************************
*************INIEVM.ASM**************************
*************************************************
        .global _main,inievm
timer0  .word   808020h
SPGC0   .word   0e970300h
        .text
;

inievm
        LDI     1h,R6
        LDI     @timer0,AR4
        STI     R6,*+AR4(8)
        LDI     2C1h,R6
        STI     R6,*AR4
        LDI     2h,IOF
        LDI     111h,R6
        STI     R6,*+AR3(2)
        STI     R6,*+AR3(3)
        LDI     @SPGC0,R6
```

```
        STI     R6,*AR3
        LDI     0,R6
        STI     R6,*+AR3(8)
        LDI     6h,IOF
        LDI     0,IF
        OR      10h,IE
        OR      200h,ST
        LDI     3h,R5
;       OR      2000h,ST
        IDLE
        LDI     1A34H,R5
        IDLE
        LDI     3h,R5
        IDLE
        LDI     1A7H,R5             ;Paso altas esta desactivado
        IDLE
        RETS


************************************************
*********************INIT.CMD***********************
************************************************
;Mapa de la memoria del EVM TMS320C30
;
reset
MR
;
MA 0x000000,0x004000,RAM ; SRAM
MA 0x304000,0x001000,RAM ; COMPORT
MA 0x809800,0x000400,RAM ; RAM0
MA 0x809c00,0x000400,RAM ; RAM1
;
;El mapeo de la memoria de periferica.
;
MA 0x808000,0x000010,RAM ; DMA
MA 0x808020,0x000010,RAM ; TIMER0
MA 0x808030,0x000010,RAM ; TIMER1
MA 0x808040,0x000010,RAM ; SPORT0
MA 0x808050,0x000010,RAM ; SPORT1
MA 0x808060,0x000001,RAM ; XBUSCTL
MA 0x808064,0x000001,RAM ; PBUSCTL
;
MAP ON
DASM PC
echo EVM ESTA INICIALIZADO

************************************************
```

```
****************TRANSMIS.ASM************************
*****************************************************
        .global _c_int05,_c_int99,serialp
        .text

_c_int05:                       ;XINT0  interrupcion del puerto serial
                                ;para transmision
        LDI     @serialp,AR3    ;en AR3 es la direccion del puerto serial
        STI     R5,*+AR3(8)     ;se transmite la palabra de R5 a puerto
        LDI     *+AR3(12),R5    ;se lee la palabra de puerto serial
;Dos bits menos significativos que se reciben de AIC no traen la
;informacion sobre la muestra desde  A/D.
;Estos bits son siempre equal a cero. Para obtener los valores
;correctos de la muestra de entrada es necesario quitar dos bits
;menos significativos. Esto hacen los instrucciones siguientes.

        LSH      16,R5      ;corrimiento del <R5> a la izquierda de 16 bits
        ASH     -18,R5      ;corrimiento del <R5> a derecho de 18 bits
        RETI                ;regreso de interupcion.

;Si se hace la interrupcion del puerto de transmision el programa salta a un
;lazo indefinido. Si se hace la interrupcion el programa salta a un lazo
;indefinido. En el registro R7 se alternan en cada paso los valores 0h y
fffh.

_c_int99:                       ;se indican todas los interrupciones no deseadas
CHYBA:  LDI     0,R7
        LDI     1,R7
        BR      CHYBA
        .end

*****************************************************
*****************VECTOR.ASM**************************
*****************************************************

        .global RESET,INT0,INT1,INT2,INT3,_main
        .global XINT0,RINT0,XINT1,RINT1,TINT0,TINT1,DINT
        .global _c_int05,_c_int99

;Vectores de interrupcion:
        .sect   "vecs"      ;esta seccion se escribe a la direccion 0
RESET   .word   _main       ;vector RESET
INT0    .word   _c_int99    ;vector INT0
INT1    .word   _c_int99    ;vector INT1
INT2    .word   _c_int99    ;vector INT2
INT3    .word   _c_int99    ;vector INT3
XINT0   .word   _c_int05    ;vector de puerto serial numero cero XMT
```

```
RINT0     .word   _c_int99   ;vector de puerto serial numero cero  RCV
XINT1     .word   _c_int99   ;vector de puerto serial numero uno   XMT
RINT1     .word   _c_int99   ;vector de puerto serial numero uno   RCV
TINT0     .word   _c_int99   ;vector de timer numero 0
TINT1     .word   _c_int99   ;vector de timer numero 1
DINT      .word   _c_int99   ;vector DMA
          .space  20         ;espacio reservado
          .space  32         ;espacio de vector TRAP
          .end

*********************************************
***************WDF6ORDS.ASM*****************
*********************************************
*     El Filtro de onda orden n=6
*
*      MODULO DE EVALUCION
*********************************************
;
;assembler:
;ASM30 wdf5ordc.asm
;linker:
;LNK30 wdf5ordc.cmd
;El modulo de Evaluacion EVM:
;EVM30 wdf5ordc.out

          .global inievm,x,serialp,_main
;La memoria contiene los variables

N2        .float     0
N4        .float     0
N6        .float     0
N8        .float     0
N10       .float     0
N12       .float     0

N2_addr   .word      N2
N4_addr   .word      N4
N6_addr   .word      N6
N8_addr   .word      N8
N10_addr  .word      N10
N12_addr  .word      N12

BN1_temp  .float     0
BN3_temp  .float     0
BN5_temp  .float     0
XN1_temp  .float     0
XN5_temp  .float     0
```

```
BN1_temp_addr   .word   BN1_temp
BN3_temp_addr   .word   BN3_temp
BN5_temp_addr   .word   BN5_temp
XN1_temp_addr   .word   XN1_temp
XN5_temp_addr   .word   XN5_temp


;Coeficientes del filtro
;La seccion de la memoria contiene los coeficientes

A1      .float    0.658935        ;A1
B2      .float    0.317843        ;B2
A3      .float    0.1997968       ;A3
B4      .float    0.17655         ;B4
A5      .float    0.231596        ;A5
B61     .float    0.512134        ;B61
B62     .float    0.487865        ;B62
A1p     .word     A1
B2p     .word     B2
A3p     .word     A3
B4p     .word     B4
A5p     .word     A5
B61p    .word     B61
B62p    .word     B62

serialp .word   808040h         ; la direccion del puerto serial

_main   LDI     @serialp,AR3    ;AR3=808040h - serial port address
        CALL    inievm          ;se llama la subrutina de inicializacion
        AND     0h,ST           ;interrupcion mascarable
        LDF     0,R6
        LDI     010H,IE         ;posibilita la interrupcion de transmisio
        FIX     R6,R5           ;0--->R5
Loop:   LSH     2,R5            ;el contenido de R5 se corre de dos bits
                                ;a la izquierda
        IDLE                    ;el procesador espera a la interrupcion
        AND     0h,ST           ;interrupcion no posible
        FLOAT   R5,R0           ;el numero obtenido del puerto de entrada
                                ;se convierte del formato entero a numero
                                ;flotante y se carga a la variable R0

        LDI     @A1p,AR3
        LDI     N2_addr,AR1

;XN1
        MPYF3   *AR3,*AR1,R6          ;A1*N2 ---> R6
```

```
        MPYF3   *AR3,R0,R4          ;A1*XN ---> R4
        SUBF3   R6,*AR1,R6          ;N2-A1*N2 ---> R6
        ADDF    R4,R6               ;XN1=A1*XN+N2-A1*N2 ---> R6

;XN2
        LDI     @N4_addr,AR1
        ADDF3   *AR1,R6,R1          ;XN2=N4 + XN1 ---> R1

;XN3
        LDI     @N6_addr,AR1
        LDI     @A3p,AR3
        MPYF3   *AR3,R1,R2          ;A3*XN2 ---> R2
        SUBF3   R2,*AR1,R3          ;N6-A3*XN2 ---> R3
        MPYF3   *AR3,*AR1,R4        ;A3*N6 ---> R4
        SUBF    R4,R3               ;XN3=N6-A3*XN2-A3*N6 ---> R3

;XN4
        LDI     @N8_addr,AR1
        ADDF3   R3,*AR1,R2          ;XN4=XN3+N8 ---> R2

;XN5
        LDI     @N10_addr,AR1
        LDI     @A5p,AR3
        MPYF3   *AR3,R2,R4          ;A5*XN4 ---> R4
        SUBF3   R4,*AR1,R4          ;N10-A5*XN4 ---> R4
        MPYF3   *AR3,*AR1,R5        ;A5*N10 ---> R5
        SUBF    R5,R4               ;XN5=N10-A5*XN4-A5*N10 ---> R4
        STF     R4,@XN5_temp        ;XN5 ---> XN5_temp

;BN5
        LDI     @N12_addr,AR1
        LDI     @B61p,AR3
        MPYF3   *AR3,*AR1,R5        ;B61*N12 ---> R5
        MPYF3   *AR3,R4,R7          ;B61*XN5 ---> R7
        SUBF    R7,R4               ;XN5-B61*XN5 ---> R4
        SUBF3   R5,R4,R7            ;BN5=XN5-B61*XN5-B61*N12 ---> R7
        STF     R7,@BN5_temp        ;BN5 ---> BN5_temp to R7 free

;BN4
        LDI     @N10_addr,AR1
        LDI     @A5p,AR3
        MPYF3   *AR3,R2,R4          ;A5*XN4 ---> R4
        SUBF    R4,*AR1,R5          ;N10-A5*XN4 ---> R5
        ADDF    R2,R5               ;XN4+N10-A5*XN4 ---> R5
        MPYF3   *AR3,*AR1,R4        ;A5*N10 ---> R4
        SUBF    R4,R5               ;XN4+N10-A5*XN4-A5*N10 ---> R5
        ADDF    R7,R5               ;BN4=XN4+N10-A5*XN4-A5*N10+BN5 ---> R5
```

```
;BN3
        LDI     @B4p,AR3
        MPYF3   *AR3,R2,R4              ;B4*XN4 ---> R4
        MPYF3   *AR3,R5,R7              ;B4*BN4 ---> R7
        SUBF3   R7,R3,R7                ;XN3-B4*BN4 ---> R7
        SUBF    R4,R7                   ;BN3=XN3-B4*BN4-B4*XN4 ---> R7

;BN2
        LDI     @N6_addr,AR1
        LDI     @A3p,AR3
        ADDF3   *AR1,R1,R4              ;N6+XN2 ---> R4
        ADDF    R7,R4                   ;BN3+N6+XN2 ---> R4
;       STF     R7,@R7                  ;BN3 ---> BN3_temp

;       LDI     @BN3_temp_addr,AR5      ;BN3 ~ AR5 a R7
        STF     R7,@BN3_temp            ;BN3 ---> BN3_temp to R7 free
        MPYF3   *AR3,R1,R7              ;A3*XN2 ---> R7
        SUBF    R7,R4                   ;BN3+N6+XN2-A3*XN2 ---> R4
        MPYF3   *AR3,*AR1,R7            ;A3*N6 ---> R7
        SUBF    R7,R4                   ;BN2=BN3+N6+XN2-A3*XN2-A3*N6 --> R4

;BN1
        LDI     @B2p,AR3
        MPYF3   *AR3,R1,R7              ;B2*XN2 ---> R7
        SUBF3   R7,R6,R7                ;XN1-B2*XN2 ---> R7
        STF     R6,@XN1_temp            ;XN1 ---> XN1_temp to R6 free
        MPYF3   *AR3,R4,R6              ;B2*BN2 ---> R6
        SUBF    R6,R7                   ;BN1=XN1-B2*XN2-B2*BN2 ---> R7

;N1
        LDI     @A1p,AR3
        LDI     @N2_addr,AR1
        STF     R6,@XN1_temp            ;XN1 ---> XN1_temp to R6 free
        STF     R7,@BN1_temp            ;BN1 ---> BN1_temp to R7 free
        MPYF3   *AR3,R0,R6              ;A1*XN ---> R6
        MPYF3   *AR3,*AR1,R7            ;A1*N2 ---> R7
        SUBF    R7,R6                   ;A1*XN-A1*N2 ---> R6
        LDI     @BN1_temp_addr,AR6
        ADDF    *AR6,R6                 ;N1=BN1+A1*XN-A1*N2 ---> R6

;N3
       ADDF3    *AR6,R4,R7              ;N3=BN1+BN2 ---> R7

;XN, XN1, y XN3 no se utilizan en el programa siguiente
;XN=R0
;XN1=XN1_temp
```

```
;XN3=R3

;N5
        LDI     @A3p,AR3
        LDI     @N6_addr,AR1
        MPYF3   *AR3,R1,R0              ;A3*XN2 ---> R0
        LDI     @BN3_temp_addr,AR5
        SUBF3   R0,*AR5,R0              ;BN3-A3*XN2 ---> R0
        MPYF3   *AR3,*AR1,R3            ;A3*N6 ---> R3
        SUBF    R3,R0                   ;N5=BN3-A3*XN2-A3*N6 ---> R0

;N7
        ADDF    *AR5,R5                 ;N7=BN3+BN4 ---> R5

;XN2,BN1,BN2,BN3,BN4 no se utilizan en el programa siguiente
;XN2=R1
;BN1=BN1_temp
;BN2=R4
;BN3=BN3_temp
;BN4=sobrescrita

;N9
        LDI     @A5p,AR3
        LDI     @N10_addr,AR1
        MPYF3   *AR3,R2,R3              ;A5*XN4 ---> R3
        LDI     @BN5_temp_addr,AR6
        SUBF3   R3,*AR6,R3              ;BN5-A5*XN4 ---> R3
        MPYF3   *AR3,*AR1,R1            ;A5*N10 ---> R1
        SUBF    R1,R3                   ;N9=BN5-A5*XN4-A5*N10 ---> R3

;N11
        LDI     @B62p,AR3
        LDI     @N12_addr,AR1
        MPYF3   *AR3,*AR1,R1            ;B62*N12 ---> R1
        SUBF3   R1,*AR6,R1              ;BN5-B62*N12 ---> R1
        ADDF    *AR1,R1                 ;N11=N12+BN5-B62*N12 ---> R1


;Actualizacion del parte Nx
;N1=R6
;N3=R7
;N5=R0
        STF     R6,@N2                  ;N1 ---> N2
        STF     R7,@N4                  ;N3 ---> N4
        STF     R0,@N6                  ;N5 ---> N6

;YN
```

```
;       LDI     @B62p,AR3
;       LDI     @N12_addr,AR1
        LDI     @XN5_temp_addr,AR6
        MPYF3   *AR3,*AR6,R0            ;B62*XN5 ---> R0
        LDF     0,R6
        SUBF3   R0,R6,R0                ;0-B62*XN5 ---> R0
        MPYF3   *AR3,*AR1,R6            ;B62*N12 ---> R6
        SUBF3   R6,R0,R6                ;0-B62*XN5-B62*N12 ---> R6
;       ADDF3   *AR1,R3,R1              ;YN=N10+N9 ---> R1

;Actualizacion del resto de Nx
;N1=R6
;N3=R7
;N5=R0
;N7=R5
;N9=R3
        STF     R5,@N8                  ;N7 ---> N8
        BRD     Loop                    ;salto a la etiqueta Loop. Tres
                                        ;instrucciones que siguen se
                                        ;ejecutan
        STF     R3,@N10                 ;N9 ---> N10
        STF     R1,@N12                 ;N11 ---> N12
        FIX     R6,R5                   ;YN ---> R5


*****************************************************
*******************WDF6ORDC.CMD**********************
*****************************************************

wdf6ordc.obj
inievm.obj
vectors.obj
transmis.obj
-e _main
-o wdf6ordc.out
-m wdf6ordc.map

MEMORY
{
    VECS  : org = 0       , len = 0x40
    ROM   : org = 0x40    , len = 0x3fc0
    RAM   : org = 0x809800, len = 0x800
}
SECTIONS
{
        .text :  {}  >  ROM
        .cinit:  {}  >  ROM
```

```
        .stack:  {}  >  RAM
        .bss   :  {}  >  RAM
}
```

4.5 Filtro de estado

Ejemplo 6:

Calcular mediante el MATLAB el filtro de estado·paso banda de 4 orden para los especificaciones siguientes: Aproximación ellíptica, frecuencia del corte f_1=0.5 f_{-1}=0.6, atenuación en la banda de paso a_{max}=0.05 dB y atenuación en la banda de rechazo a_{min}=20 dB. Simular el filtro mediante el simulador dr TMS320C30 y realizarlo mediante el módulo de evaluación de C30.

4.5.1 Diseño y análisis del filtro mediante MATLAB

Escriba el siguiente programa en el archivo $C:\backslash SIM30\backslash ESTADO>$. y mediante el matlab calcule las matrices de estado **A**, **B**, **C** y **D**. La estructura del filtro para el orden N=4 se muestra en la figura 4.8. De esta estructura se puede dibujar la estructura de estado para cualquier orden. Las ecuaciones siguientes realizan el filtro de estado de orden 4.

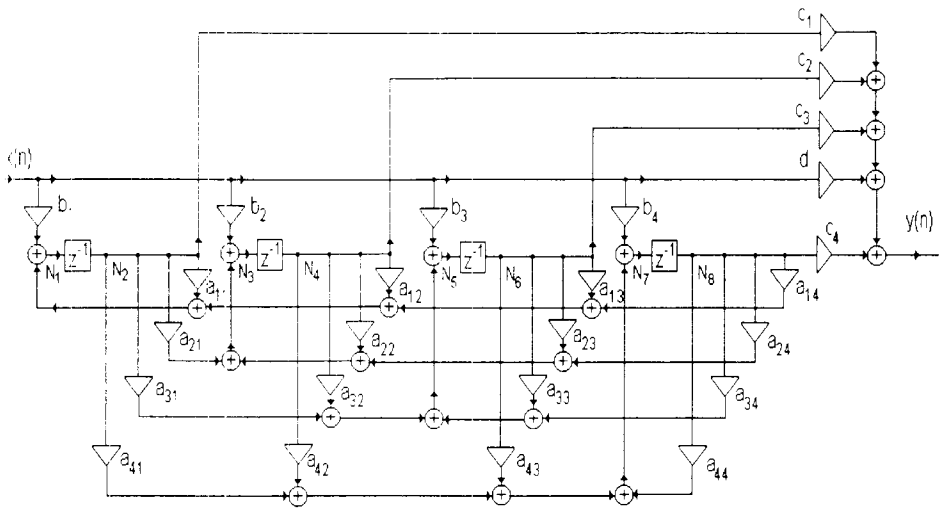

Figura 4.8: El filtro de estado de orden n=4

$$YN = D.XN + N2.C1 + N4.C2 + N6.C3 + N8.C4$$
$$N1 = B1.XN + N2.A11 - N4.A12 + N6.A13 + N8.A14$$
$$N3 = B2.XN + N2.A21 - N4.A22 + N6.A23 + N8.A24$$
$$N5 = B3.XN + N2.A31 + N4.A32 + N6.A33 + N8.A34$$
$$N7 = B4.XN + N2.A41 + N4.A42 + N6.A43 + N8.A44$$

```
clear
tic %time start
```

```
N=2
Amax=0.05
Amin=20
f1=0.5
f2=0.6
wn=[f1 f2]
[a,b,c,d]=ellip(N,Amax,Amin,wn)

d
c*b
number=200%15%30
cc=[d c*b];
for i=1:number

   cc=[cc c*a^i*b];

end

[h,w]=freqz(cc,1,100);
toc %time end
figure(1)
plot(w,20*log10(abs(h)))

figure(2)
plot(w,20*log10(abs(h))),
grid
```

Las matrices de estado obtenidos mediante matlab son las siguientes:

```
a =
   -0.5805   -0.2955    0.4195   -0.2955
    0.2955   -0.2082    0.2955    0.7918
   -0.4195    0.2955    0.5805    0.2955
   -0.2955   -0.7918   -0.2955    0.2082
b =
    0.3855
    0.2716
   -0.3855
   -0.2716
c =
    0.1620    0.5713    0.1620    0.5713
d =
    0.2489
```

El resultado obtenido mediante MATLAB se muestra en la figura 4.9

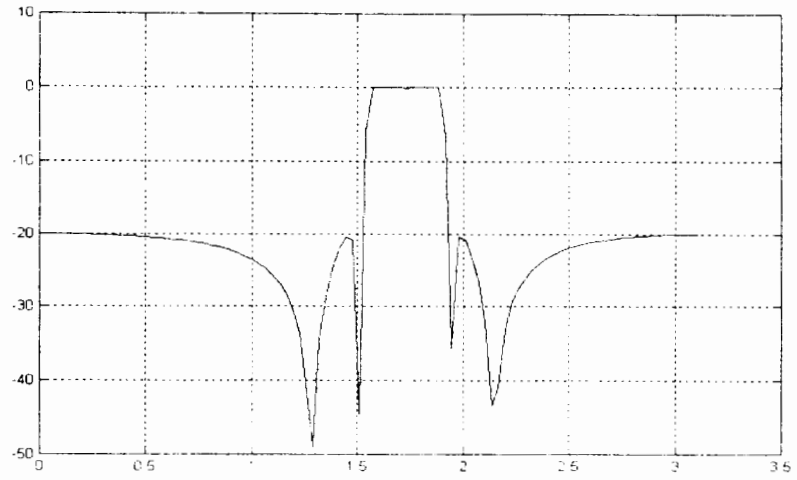

Figura 4.9: Atenuación del filtro paso banda obtenida mediante MATLAB

4.5.2 Análisis del filtro de estado mediante el simulador

Escriba el siguiente programa en el archivo $C:\backslash SIM30\backslash ESTADO>$, compilelo para obtener el módulo estado.obj. Mediante el ligador se obtiene el módulo estado.out. Mediante el comando

sim30.exe estado.out

el resultado, respuesta al impulso, se escribe en el archivo output.dat, en la forma hexadecimal. Para calcular el espectro, es necesario convertir los números de hexadecimal a decimal mediante el comando

hex2dec.exe output.dat outputde.dat

Los datos en la forma decimal se encuentran en el archivo *outputdc.dat*. Mediante MATLAB se obtiene el espectro que se muestra en la figura 4.10. Comparando el espectro calculado mediante MATLAB, Figura 4.9 con el espectro obtenido mediante el simulador de TMS320C30 figura 4.10, se ve que se obtiene el mismo resultado, esto significa que el programa en ensamblador es correcto y se puede implementar el filtro con el módulo de evaluación. El programa para el simulador TMS320C30 es el siguiente.

```
            .global     _main, RESET

            .data
inaddr      .word 804000h           ;inaddr-input addres
outaddr     .word 804001h           ;outaddr-output addres

*****************************************************
* esta seccion contiene los coeficientes del filtro*
*****************************************************
;
;La matriz A
;
```

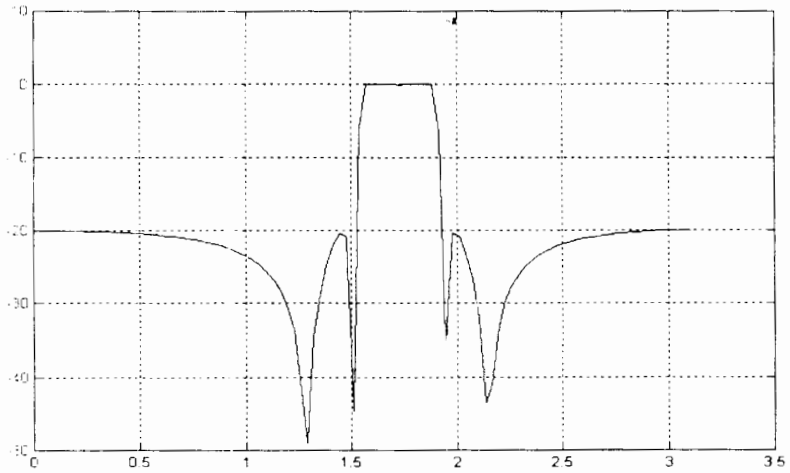

Figura 4.10: Espectro del filtro paso banda obtenido mediante el simulador

```
A11 .float    -0.5855
A12     .float    -0.2955
A13     .float     0.4195
A14     .float    -0.2955


A21     .float     0.2955
A22     .float    -0.2082
A23     .float     0.2955
A24     .float     0.7918

A31     .float    -0.4195
A32     .float     0.2955
A33     .float     0.5805
A34     .float     0.2955


A41     .float    -0.2955
A42     .float    -0.7918
A43     .float    -0.2955
A44     .float     0.2082



A11_addr    .word     A11
;
;La matriz B
;

B1      .float     0.3855
B2      .float     0.2716
```

```
B3      .float    -0.3855
B4      .float    -0.2716

B1_addr .word B1

;
;La matriz  C
;

C1      .float      0.1620
C2      .float      0.5713
C3      .float      0.1620
C4      .float      0.5713


;LDI @C1_addr,AR7
C1_addr .word   C1

;
;El coeficiente  D
;

D       .float     0.2489

        .text

;
;La matriz N

N1      .float      0
N3      .float      0
N5      .float      0
N7      .float      0


N1_addr .word       N1

N2      .float      0
N4      .float      0
N6      .float      0
N8      .float      0


N2_addr  .word   N2

_main   ldi      @inaddr,AR4
        ldi      @outaddr,AR3
```

```
Loop    LDI     *AR4,R5
        FLOAT   R5,R0
;*******************************
        LDI     @A11_addr,AR1   ;apuntador de la direccion de Axx
        LDI     @B1_addr,AR5    ;apuntador de la direccion de Bx
        LDI     @N1_addr,AR6    ;apuntador de nuevo Nx
        LDI     @N2_addr,AR2    ;apuntador de retardados Nx
        LDI     @C1_addr,AR7    ;apuntador de la direccion de Cx
        LDF     8,R1            ;contador de repeticion

Equat:
        LDI     7,RC
        RPTB    LoopNA
        MPYF3   *AR1++,*AR2++,R2        ;A11*N2--->R2
LoopNA  ADDF    R2,R6

        MPYF3   *AR5++,R0,R4            ;R4=B1*XN
        ADDF    R4,R6                   ;N1=R6
        STF     R6,*AR6++               ;@N1
        LDF     0,R6
        SUBF    1,R1
        BNZ     Equat
;YN
        LDF     @D,R7
        MPYF3   R7,R0,R6                ;D*XN ---> R6
        LDI     7,RC
        RPTB    LoopNC
        MPYF3   *AR7++,*AR2++,R2        ;C*N2--->R2
LoopNC ADDF    R2,R6
                                        ;YN=R6

;Actualizacion
        LDI     7,RC
        RPTB    Actual
        LDF     *--AR6,R7               ;*AR6--,R7
Actual  STF     R7,*--AR2               ;R7,*AR2--
        FIX     R6,R                    ;R6 se convierte a punto fijo
                                        ;y el resultado se guarda a R5
        STI     R5,*AR3
        BR      Loop            ;el retrazo, salto a la etiqueta Loop
                                ;tres instrucciones siguientes se ejecutan
```

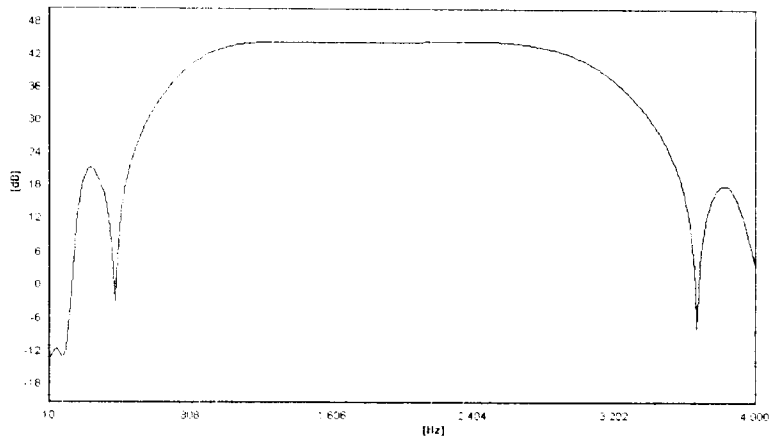

Figura 4.11: Atenuación de paso banda de orden 4, obtenida en la pantalla de analizador de espectro

4.5.3 Implantación del filtro de estado con el módulo de evaluación EVM

En el directorio $C:\backslash EVM\backslash ESTADO>$, escribir los programas siguientes:

ssevm30.asm *ssevmco.asm* *transmis.asm*
inievm.asm *vectors.asm* *init.cmd*
ssfevm.cmd

Mediante los comandos siguientes crear el módulo .out.

```
asm30.exe  ssevm30.asm
asm30.exe  ssevmco.asm
asm30.exe  transmis.asm
asm30.exe  vectors.asm
asm30.exe  inievm.asm
lnk30.exe  ssevm30.cmd
evm30.exe  ssevm30.out
```

Si el programa está correcto y en la entrada de la tarjeta se conecta el generador que produce el ruido blanco en el analizador del espectro, conectado a la salida del módulo de evaluación, se obtiene la gráfica que se muestra en la figura 4.11

```
;*************************************************
;*********************SSEVM30.ASM*****************
;*************************************************
;assembler:
;ASM30 SSFEVM30.asm
;ASM30 SSFEVMco.asm
;linker:
;LNK30 SSFEVM30.cmd
;instruccion para el EVM:
;EVM30 SSFEVM30.out
```

```
;************************************
; Filtro de estado paso banda  (SSF)
;
;N=4 Amax=0.05 Amin=20 f1=0.5 f2=0.6
;
;************************************

; MATLAB
; wn=[f1 f2]
; [a,b,c,d]=ellip(N,Amax,Amin,wn)
;************************************

        .global _main,inievm,serialp,A11_addr,B1_addr,C1_addr,D
        .data
order_1 .set  7       ;el orden del filtro menos uno
order   .float  8       ;el orden del filtro


;*********************************
; Coeficientes del filtro
; en el archivo SSFcoef.asm
;*********************************

        .text
;Matrix

N1      .float    0
N3      .float    0
N5      .float    0
N7      .float    0

N1_addr .word   N1

N2      .float    0
N4      .float    0
N6      .float    0
N8      .float    0

N2_addr .word   N2

serialp .word   808040h          ;la direccion del puerto serial
_main   LDI     @serialp,AR3     ;AR3=808040h - direccion del
        CALL    inievm           ;rutina de inicializacion de EVM
        AND     0h,ST            ;interrupcion protejida (mascarable)
        LDF     0,R6
        LDI     010H,IE          ;se habilita la interrupcion del
                                 ;puerto serial numero cero
```

```
        FIX     R6,R5               ;0(entero)--->R5
Loop:   LSH     2,R5                ;R5 se corre dos bits a lado
                                    ;izguierdo
        IDLE                        ;se espera a interrupcion protejida
        AND     0h,ST               ;interupciones no permitidas
        FLOAT   R5,R0               ;la muestra obtenida se convierte de
                                    ;forma entera a la forma flotante y
                                    ;se almacena a variable R0

;Loop   LDI     *AR4,R5
;       FLOAT   R5,R0
        LDI     @A11_addr,AR1       ;el apuntador de la direccion de Axx
        LDI     @B1_addr,AR5        ;el apuntador de la direccion de Bx
        LDI     @N1_addr,AR6        ;el apuntador de nuevo Nx
        LDI     @N2_addr,AR2        ;el apuntador de Nx retardadas
        LDI     @C1_addr,AR7        ;el apuntador de la direccion de
                                    ;los coefficientes  Cx
        LDF     @order,R1           ;contador de repeticiones R1=8
        LDF     0,R6
Equat:
        LDI     order_1,RC          ;7,RC
        RPTB    LoopNA
        MPYF3   *AR1++,*AR2++,R2    ;A11*N2--->R2
        ADDI    1,R3
LoopNA ADDF    R2,R6
        MPYF3   *AR5++,R0,R4        ;R4=B1*XN
        ADDF    R4,R6               ;N1=R6
        STF     R6,*AR6++           ;@N1
        LDF     0,R6
;YN
        SUBF    1,R1
        LDI     @N2_addr,AR2        ;El contador apunta N2, N4, ... N16
        BNZ     Equat
        LDF     @D,R7
        MPYF3   R7,R0,R6                ;D*XN ---> R6
        LDI     order_1,RC              ;7,RC
        RPTB    LoopNC
        MPYF3   *AR7++,*AR2++,R2        ;C*N2--->R2
LoopNC  ADDF     R2,R6                  ;YN=R6

;
;Se actualiza N par N2, N4, ..., N16
;
        LDI     order_1,RC              ;7,RC
        RPTB    Actual
        LDF     *--AR6,R7
Actual  STF     R7,*--AR2
```

```
        NOP
        BRD     Loop            ;el retrazo - salto a la etiqueta Loop.
                                ;tres instrucciones siguientes se ejecutan
        FIX     R6,R5           ;R6 se convierte en numero entero y el
                                ;resultado se almacena a R5
        NOP
NOP
        .global _main,inievm,serialp,A11_addr,B1_addr,C1_addr,D
        .data

*************************************************
***************SSEVMCO.ASM***********************
*********LA SECCION TIENE SILO COEFICIENTES********
;
;Matriz A
A11     .float     -0.5855
A12     .float     -0.2955
A13     .float      0.4195
A14     .float     -0.2955
*
A21     .float      0.2955
A22     .float     -0.2082
A23     .float      0.2955
A24     .float      0.7918
*
A31     .float     -0.4195
A32     .float      0.2955
A33     .float      0.5805
A34     .float      0.2955
*
A41     .float     -0.2955
A42     .float     -0.7918
A43     .float     -0.2955
A44     .float      0.2082

A11_addr .word     A11
;
;Matriz B
;

B1      .float     0.3855
B2      .float     0.2716
B3      .float    -0.3855
B4      .float    -0.2716


B1_addr .word B1
```

```
;
;Matriz C
;
C1      .float      0.1620
C2      .float      0.5713
C3      .float      0.1620
C4      .float      0.5713

C1_addr .word   C1

;
;Matriz D
;

D       .float     0.2489

*****************************************
************TRANSMIS.ASM*****************
*****************************************


     .global _c_int05,_c_int99,serialp
     .text

_c_int05:                   ;XINT0 Interrupcion del puerto serial
     LDI    @serialp,AR3    ;AR3 contiene la direccion del puerto
                            ;serial numero cero
     STI    R5,*+AR3(8)     ;la muestra de salida es en la variable
                            ;R5 y se almacena al puerto serial numero
                            ;cero

     LDI    *+AR3(12),R5    ;la muestra de la entrada se obtiene
                            ;del puerto serial
;Dos bits mas significatiovs de la palabra obtenida de A/D no traen
;la informacion sobre la muestra de entrada. Estos bits tienen el
;valor cero. Para obtener el valor verdadero es necesario borrar dos
;bits mas significativos. Eso se hace con los instrucciones siguientes:

        LSH     16,R5       ;contenido de R5 se corre de 16 bits
        ASH     -18,R5      ;contenido de R5 se corre de 18 bits
        RETI



_c_int99:           ;indicacion de todas las interrupciones que no fueron
                    ;producidas con interrupcion del puerto serial  0
                    ;de transmision.
```

```
MISTAKE: LDI     0,R7
         LDI     1,R7
         BR      MISTAKE
         .end

*********************************
*********INIEVM.ASM**************
*********************************

         .global _main,inievm
timer0   .word   808020h
SPGC0    .word   0e970300h
         .text

inievm
         LDI     1h,R6
         LDI     @timer0,AR4
         STI     R6,*+AR4(8)
         LDI     2C1h,R6
         STI     R6,*AR4
         LDI     2h,IOF
         LDI     111h,R6
         STI     R6,*+AR3(2)
         STI     R6,*+AR3(3)
         LDI     @SPGC0,R6
         STI     R6,*AR3
         LDI     0,R6
         STI     R6,*+AR3(8)
         LDI     6h,IOF
         LDI     0,IF
         OR      10h,IE
         OR      200h,ST
         LDI     3h,R5
         OR      2000h,ST
         IDLE
         LDI     1A34H,R5
         IDLE
         LDI     3h,R5
         IDLE
         LDI     2A7H,R5
         IDLE
         RETS

*************************************************
*****************VECTORS.ASM*********************
*************************************************
```

```
        .global RESET,INT0,INT1,INT2,INT3,_main
        .global XINT0,RINT0,XINT1,RINT1,TINT0,TINT1,DINT
        .global _c_int05,_c_int99

;Vectores de interrupcion
        .sect   "vecs"      ;esta seccion se escribe a la direccion 0
                            ;y a las direcciones siguientes
RESET   .word  _main        ;vector RESET
INT0    .word   _c_int99    ;vector INT0
INT1    .word   _c_int99    ;vector INT1
INT2    .word   _c_int99    ;vector INT2
INT3    .word   _c_int99    ;vector INT3
XINT0   .word   _c_int05    ;vector de puerto serial numero cero XMT
RINT0   .word   _c_int99    ;vector de puerto serial numero cero  RCV
XINT1   .word   _c_int99    ;vector de puerto serial numero uno   XMT
RINT1   .word   _c_int99    ;vector de puerto serial numero uno   RCV
TINT0   .word   _c_int99    ;timer numero 0
TINT1   .word   _c_int99    ;timer numero 1
DINT    .word   _c_int99    ;vector DMA
        .space  20          ;espacio reservado
        .space  32          ;espacio de vector TRAP
        .end

************************************************
****************INIT.CMD************************
************************************************

;El mapa de la memoria de EVM (Evaluation Module) TMS320C30
;
reset
MR
;
MA 0x000000,0x004000,RAM ; SRAM
MA 0x804000,0x001000,RAM ; COMPORT
MA 0x809800,0x000400,RAM ; RAM0
MA 0x809c00,0x000400,RAM ; RAM1
;
;El
 mapa de la memoria de los unidos perifericos.
;
MA 0x808000,0x000010,RAM ; DMA
MA 0x808020,0x000010,RAM ; TIMER0
MA 0x808030,0x000010,RAM ; TIMER1
MA 0x808040,0x000010,RAM ; SPORT0
MA 0x808050,0x000010,RAM ; SPORT1
```

```
MA 0x808060,0x000001,RAM ; XBUSCTL
MA 0x808064,0x000001,RAM ; PBUSCTL
;
MAP ON
DASM PC
echo EVM ESTA INICIALIZADO

*************************************************
****************SSFEVM.CMD***********************
*************************************************

SSFEVM30.obj
SSFEVMco.obj
inievm.obj
vectors.obj
transmis.obj

-e _main
-o SSFEVM30.out
-m SSFEVM30.map

MEMORY
{
    VECS  : org = 0        , len = 0x40
    ROM   : org = 0x40     , len = 0x3fc0
    RAM   : org = 0x809800, len = 0x800
}

SECTIONS
{
.text : {} > ROM
.cinit: {} > ROM
.stack: {} > RAM
.bss  : {} > RAM

}
```

4.6 Generador de triángulo

Para el módulo de evaluación de C30 escribir el programa para generar la señal triangular. El programa compilar, crear el módulo *generado.out*, conectar la tarjeta con el osciloscopio y correr el programa con el módulo de evaluación.

```
*****************************************************
**********GENERADOR.ASM**********************
*****************************************************

        .global _main,inievm,serialp
        .data
outaddr .word 804001h       ;la direccion de outaddr (salida)

A0      .float    0
A1      .float    1         ;0.1
A2      .float    2         ;0.2
A3      .float    3         ;0.3
A4      .float    4         ;0.4
A5      .float    3         ;0.3
A6      .float    2         ;0.2
A7      .float    1         ;0.1

A0_addr   .word   A0
          .text

serialp   .word   808040h
_main     LDI     @serialp,AR3
          CALL    inievm
          AND     0h,ST
          LDF     0,R6
          LDI     010H,IE
          FIX     R6,R5

Start     LDI     @A0_addr,AR4
          LDF     8,R2
Loop:     LSH     2,R5
          IDLE
          AND     0h,ST
          FLOAT   R5,R7
          LDF     *AR4++,R0
          FIX     R0,R5
          SUBF    1,R2
          BZ      Start
          B       Loop

*****************************************************
**********INIEVM.ASM**********************
*****************************************************
        .global   _main,inievm
timer0  .word     808020h
SPGC0   .word     0e970300h
```

```
        .text
inievm
        LDI         1h,R6
        LDI         @timer0,AR4
        STI         R6,*+AR4(8)
        LDI         2C1h,R6
        STI         R6,*AR4
        LDI         2h,IOF
        LDI         111h,R6
        STI         R6,*+AR3(2)
        STI         R6,*+AR3(3)
        LDI         @SPGC0,R6
        STI         R6,*AR3
        LDI         0,R6
        STI         R6,*+AR3(8)
        LDI         6h,IOF
        LDI         0,IF
        OR          10h,IE
        LDI         3h,R5
        OR          2000h,ST
        IDLE
        LDI         1A34H,R5
        IDLE
        LDI         3h,R5
        IDLE
        LDI         2A7h,R5
        IDLE
        RETS
        .end

*********************************************************
******************TRANSMIS.ASM***************************
*********************************************************
        .global _c_int05,_c_int99,serialp
        .text

_c_int05:                       ;XINT0 interrupcion del puerto serial 0
                                ;para transmision
        LDI     @serialp,AR3    ;en AR3 es la direccion del puerto serial
        STI     R5,*+AR3(8)     ;la muestra de la salida de R5 se transmite
                                ;a puerto serial 0

       LDI     *+AR3(12),R5     ;la muestra en entrada  se recibe del
                                ;puerto serial
;Dos bits menos significativos que se reciben de AIC no traen la
;informacion sobre la muestra desde  A/D.
```

```
;Estos bits son siempre igual a cero. Para obtener los valores
;correctos de la muestra de entrada es necesario quitar dos bits
;menos significativos. Esto hacen los instrucciones siguientes.


        LSH     16,R5           ;se corre <R5> a la do izguierda
                                ;de 16 bits
        ASH     -18,R5          ;se corre <R5> a la derecha de 18 bits
        RETI                    ;regreso de interrupcion


;Si se hace la interrupcion del puerto de transmision el programa salta a un
;lazo indefinido. Si se hace la interrupcion el programa salta a un lazo in-
;definido. En el registro R7 se alternan en cada paso los valores 0h y fffh.


_c_int99:                       ;se indican todas las interrupciones,
                                ; que no son llamadas mediante la
                                ;interrupcion del puerto serial numero 0.
MISTAKE: LDI    0,R7
         LDI    1,R7
         BR     MISTAKE
         .end

**********************************************************
**********VECTORS.ASM*****************************
**********************************************************

         .global RESET,INT0,INT1,INT2,INT3,_main
         .global XINT0,RINT0,XINT1,RINT1,TINT0,TINT1,DINT
         .global _c_int05,_c_int99

;Vectores de interrupcion:
         .sect   "vecs"         ;el ligador coloca esta seccion desde la
                                ;direccion 0 y a las direcciones siguientes
RESET    .word  _main           ;vector RESET
INT0     .word   _c_int99       ;vector INT0
INT1     .word   _c_int99       ;vector INT1
INT2     .word   _c_int99       ;vector INT2
INT3     .word   _c_int99       ;vector INT3
XINT0    .word   _c_int05       ;vector de puerto serial numero cero XMT
RINT0    .word   _c_int99       ;vector de puerto serial numero cero  RCV
XINT1    .word   _c_int99       ;vector de puerto serial numero uno   XMT
RINT1    .word   _c_int99       ;vector de puerto serial numero uno   RCV
TINT0    .word   _c_int99       ;vector de timer numero 0
TINT1    .word   _c_int99       ;vector de timer numero 1
```

```
DINT      .word   _c_int99     ;vector DMA
          .space  20          ;espacio reservado
          .space  32          ;espacio de vector TRAP
          .end

*****************************************************
***************GENERADO.CMD**************************
*****************************************************

Generado.obj
inievm.obj
vectors.obj
transmis.obj

-e _main
-o Generado.out
-m Generado.map

MEMORY
{
    VECS  : org = 0       , len = 0x40
    ROM   : org = 0x40    , len = 0x3fc0
    RAM   : org = 0x809800, len = 0x800
}

*****************************************************
***************INIT.CMD**************************
*****************************************************

;EL mapa de la memoria del modulo de evaluacion de TMS320C30
reset
MR
;
MA 0x000000,0x004000,RAM          ; SRAM
MA 0x804000,0x001000,RAM          ; COMPORT
MA 0x809800,0x000400,RAM          ; RAM0
MA 0x809c00,0x000400,RAM          ; RAM1
;
;El mapa de la memoria periferica
;
MA 0x808000,0x000010,RAM          ; DMA
MA 0x808020,0x000010,RAM          ; TIMER0
MA 0x808030,0x000010,RAM          ; TIMER1
MA 0x808040,0x000010,RAM          ; SPORT0
MA 0x808050,0x000010,RAM          ; SPORT1
MA 0x808060,0x000001,RAM          ; XBUSCTL
MA 0x808064,0x000001,RAM          ; PBUSCTL
```

```
;
MAP ON
DASM PC
echo EVM ESTA INICIALIZADA
```

Índice analítico

Bibliografía

CHASSAING, R. and Horning, D.W.: *Digital Signal Processing with the TMS320C25.* Toronto, John Wiley and Sons, 1990.

_____.: *Digital Signal Processing with the TMS320C30.* Toronto, John Wiley and Sons, 1992.

LARRY, E.S.:*Manual de laboratorio de procesamiento digital de señales.* México, UNAM, Facultad de Ingeniería, 2000.

LARRY, E.S.:*Arquitecturas de DSP's, Familia TMS320 y El TMS320C50.* México, UNAM, Facultad de Ingeniería, 2000.

TEXAS INSTRUMENTS: *TMS320C25 Digital Signal Processor. Product Description.* Printed in USA, 1986.

_____.: *Second-Generation TMS320 User's Guide.* Printed in USA, 1987.

_____.: *TMS34020 User's Guide.* Printed in USA, 1987.

_____.: *TMS320C1x/TMS320C2x Assembly Language Tools User's Guide.* Printed in USA, 1987.

_____.: *TMS320C25 C Compiler Reference Guide.* Printed in USA, 1988.

_____.: *TMS320C5X Starter Kit Users Guide.* Printed in USA, 1995.

_____.: *TMS320 Family Development Suport-Reference Guide.* Printed in USA, 1994.

_____.: *TMS320C2X DSP Starter Kit Users Guide.* Printed in USA, 1993.

_____.: *TMS320C25 C Compiler Reference Guide.* Printed in USA, 1988.

Esta obra se terminó de imprimir
en marzo 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.