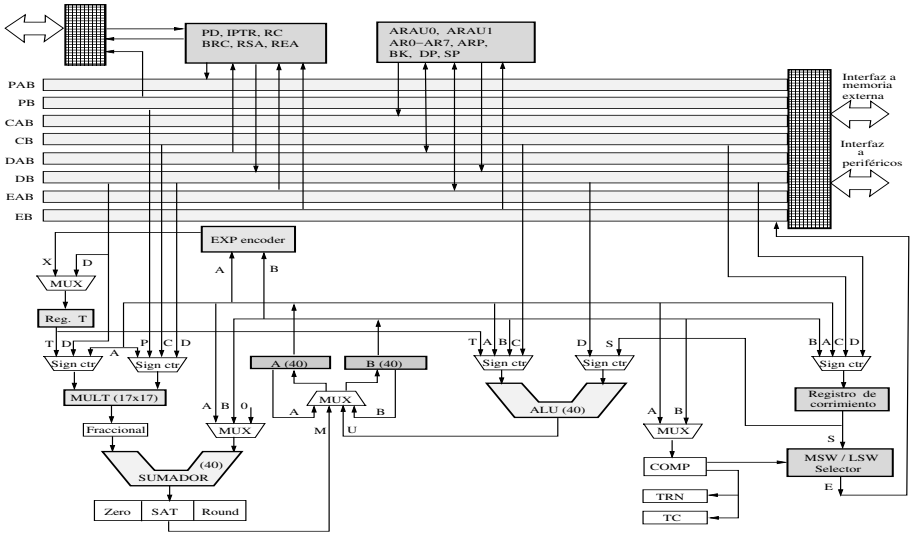
Facultad de Ingeniería

UNAM

Arquitecturas de DSPs, familias TMS320C54x y TMS320C54xx, y aplicaciones



M.I. Larry Escobar Salguero
Dr. Bohumil Psenika
Ing. Miguel Molero Armenta

Enero de 2005

El dibujo de la portada es la arquitectura del DSP TMS320C542

Índice general

Índice de figuras

Índice de cuadros

Introducción

En la actualidad los procesadores de señales (DSP) se han convertido en dispositivos electrónicos de alto desempeño para implementar soluciones a muchos problemas en áreas de ingeniería como medición, control de motores, filtrado, análisis de señales, análisis espectral, comunicaciones, procesamiento de voz, procesamiento de imágenes, etc.

En la Facultad de Ingeniería (FI), de la Universidad Nacional Autónoma de México (UNAM), se imparten las materias de Procesamiento Digital de Señales (PDS) en licenciatura, Aplicaciones con DSPs, Filtros Digitales en posgrado y otras materias que involucran la utilización de DSPs, con base a estos cursos se han elaborado las presentes notas con el objeto de que exista una guía básica para la experimentación y desarrollo de proyectos por parte de los alumnos que reciben estas materias o los interesados en este tipo de aplicaciones.

La explicación y utilización de las arquitecturas del DSP TMS320C54x (C54x) y el TMS320C54xx (C54xx), es un nuevo aporte que se hace con la intención de que las nuevas generaciones de ingenieros tengan un conocimiento más profundo del estado del arte de los DSPs, ya que en la actualidad la punta de lanza de los DSPs de Texas Instruments (TI) son las familias TMS320C5000 (C5000) y TM320C6000 (C6000).

Según nuestra experiencia, partiendo en forma ascendente desde el DSP TMS3520C50 (ver referencias [9], [10], [11]), se puede migrar al C54x y la familia C54xx que han agregado una gran potencialidad en cuanto a su CPU, periféricos y comunicación, pero la arquitectura base sigue siendo la del TMS320C50, por tanto en muchas ocasiones se hace referencia a esta última.

En el presente trabajo se supone que el lector tiene conocimientos teóricos de las bases del PDS y de preferencia que conozca algún DSP específicamente de las familias anteriores de TI, aunque no es una condición necesaria.

Este documento está conformado de las siguientes partes:

- En la *primera*, se presenta la arquitectura de los DSPs C54x y C54xx, las unidades fundamentales y los modos de direccionamiento.
- En la *segunda*, se describe el sistema pipeline de los DSPs C54x y C54xx, su operación y alguna problemática.
- En la *tercera*, el sistema de control e instrucciones relacionadas con el flujo de un programa.
- En la *cuarta*, se da una introducción al la tarjeta Starter Kit del TMS320C54x, se muestran programas elementales para que el alumno los ensamble, los corra en el ambiente del DSK observando la evolución de las variables, el estado del DSP y que a su vez aprenda a utilizar el ambiente. Estos programas empiezan con instrucciones elementales hasta instrucciones mas completas para llegar a elaborar programas más complicados que corren en tiempo real.
- En la *quinta*, se da una introducción a la tarjeta Starter Kit del TMS320C54xx y el ambiente integral code composer studio (CCS), además se tienen programas realizados tanto en lenguaje ensamblador y lenguaje C.
- En la *sexta*, se explican el manejo de los periféricos principales en conjunto con el uso de interrupciones y se muestran ejemplos.
- En la *séptima*, se presenta la forma de implementar filtros digitales tipo FIR e IIR en los DSPs C54x y C54xx, utilizando instrucciones específicas que optimizan el desempeño, a la vez se presenta un ejemplo completo de un filtro digital FIR pasobajas en tiempo real.
- Para finalizar en la *octava* se presenta una aplicación realizada en el TMS320C5402, específicamente un Sistema de Síntesis de voz en tiempo real.

Se agradece a la Facultad de Ingeniería de la UNAM y al departamento de publicaciones su apoyo a esta obra, así como las sugerencias hechas por profesores y algunos alumnos para ir depurando este material.

Los autores
Enero del 2005

Capítulo 1

Arquitectura del DSP TMS320C54x

El DSP TMS320C54x (C54x)[1] de Texas Instruments (TI) es un procesador digital de señales orientado a aplicaciones incrustadas de tiempo real en áreas tales como comunicaciones, instrumentación, control, filtrado, etc. Este DSP está basado en un hardware mejorado del TMS320C5x con algunos periféricos similares, una arquitectura tipo Harvard modificada, alto grado de paralelismo, bajo consumo de potencia que lo hace conveniente para telefonía celular, modos de direccionamiento versátiles y un conjunto de instrucciones que mejoran su desempeño. La familia TMS320C5400 (C5400) es similar al C54x en cuanto a la arquitectura y la programación, a excepción que el C5400 contiene otros periféricos internos más poderosos para comunicaciones.

En este material se da una breve introducción al DSP C54x, ya que consideramos que cuando una persona conoce la filosofía y herramientas de los DSPs de la familia TMS320 de Texas Instruments (TI) o que haya programado algunos de ellos, es capaz de migrar hacia otro DSP en muy poco tiempo.

1.1. Características generales

En esta sección se da una descripción general de la arquitectura de los DSPs C5x y C54xx, para entrar en detalle en las siguientes secciones:

- **Arquitectura y CPU**: Su arquitectura de buses separados para datos y programa tipo Harvard, permite el acceso simultáneo a instrucciones y datos proveyendo un alto

[1] La terminación "x" de esta familia de DSPs es un número que puede variar de 0 a 9 y sólo agrega cambios en el espacio de memoria, la velocidad y periféricos.

grado de paralelismo, el C54x puede ejecutar tres lecturas y una escritura en un ciclo simple:

- Arquitectura multibuses: uno para programa, tres para datos y cuatro para direcciones.
- Unidad ALU de 40 bits, con dos acumuladores independientes de 40 bits.
- Multiplicador en paralelo de 17x17 bits acoplado a un sumador dedicado (MAC).
- Unidad de comparación, selección y almacenamiento (CSSU) para codificación Viterbi.
- Codificador de exponente para calcular el exponente de un acumulador de 40 bits en un ciclo de instrucción.
- Dos generadores de dirección, es decir, dos unidades ARAU y ocho registros auxiliares mapeados (AR0-AR7).

- **Memoria**: 192 K-palabras[2] en el espacio de memoria
  - 64 K-palabras para memoria programa,
  - 64 K-palabras para datos
  - 64 K-palabras para puertos I/O.
  - El DSP TMS320C548 puede direccionar hasta 8 M-palabras (divididos en 128 páginas de 64 K), para esto contiene un registro extra que apunta a estas páginas y seis instrucciones extras para direccionar el espacio de programa extendido.
- **Instrucciones:**
  - Para repetición de una instrucción y repetición de bloques de instrucciones.
  - Para movimiento de bloques de datos y programa.
  - Con operandos de 32 bits.
  - Para lectura de dos o tres operandos simultáneos.
  - Aritméticas con almacenamiento y carga paralela.
  - Almacenamiento condicional.
  - Lenguaje de programación algebraico.

[2]Una palabra = 16 bits.

- **Periféricos internos:**
  - Estados de espera programables por software.
  - Generador de una malla de fase amarrada (PLL).
  - Control externo de bus para deshabilitación de buses externos.
  - Bus de datos con características de retención (Holder).
  - Temporizador programable.
  - Tres puertos seriales, que dependiendo de la versión pueden ser tipo host, serial o puerto multiplexado por división de tiempo (TDM).
- **Otras:**
  - Opera en modo de bajo consumo con instrucciones IDLE1, IDLE2 e IDLE3.
  - Emula el estándar 1149.1 de IEEE.
  - Velocidades: 25/20/12.5/10 ns en un ciclo de instrucción (40/50/66/80/100 MIPS).
  - Maneja seis niveles de pipeline: prebúsqueda, búsqueda, decodificación, acceso, lectura y ejecución.

1.1.1. Buses

La arquitectura del C54x está construida sobre ocho buses de 16 bits (figura 1.1):

- Buses de direcciones: PAB, CAB, DAB y EAB.
- Bus de programa (PB): transportan el código de instrucción y los operandos inmediatos de memoria programa. Este bus también está conectado al multiplicador para proveer una entrada desde memoria programa.
- Buses de datos: Interconectan varios elementos tales como el CPU, generador de direcciones de datos y programa, periféricos y memoria dato.
  - Los buses CB y DB transportan los operandos que son leídos de memoria dato. Estos buses permiten la búsqueda simultánea de dos operandos en un mismo ciclo de instrucción, o una palabra doble en un mismo ciclo.
  - El bus EB transporta los datos para ser escritos en memoria, permitiendo lecturas y escrituras para instrucciones en paralelo.

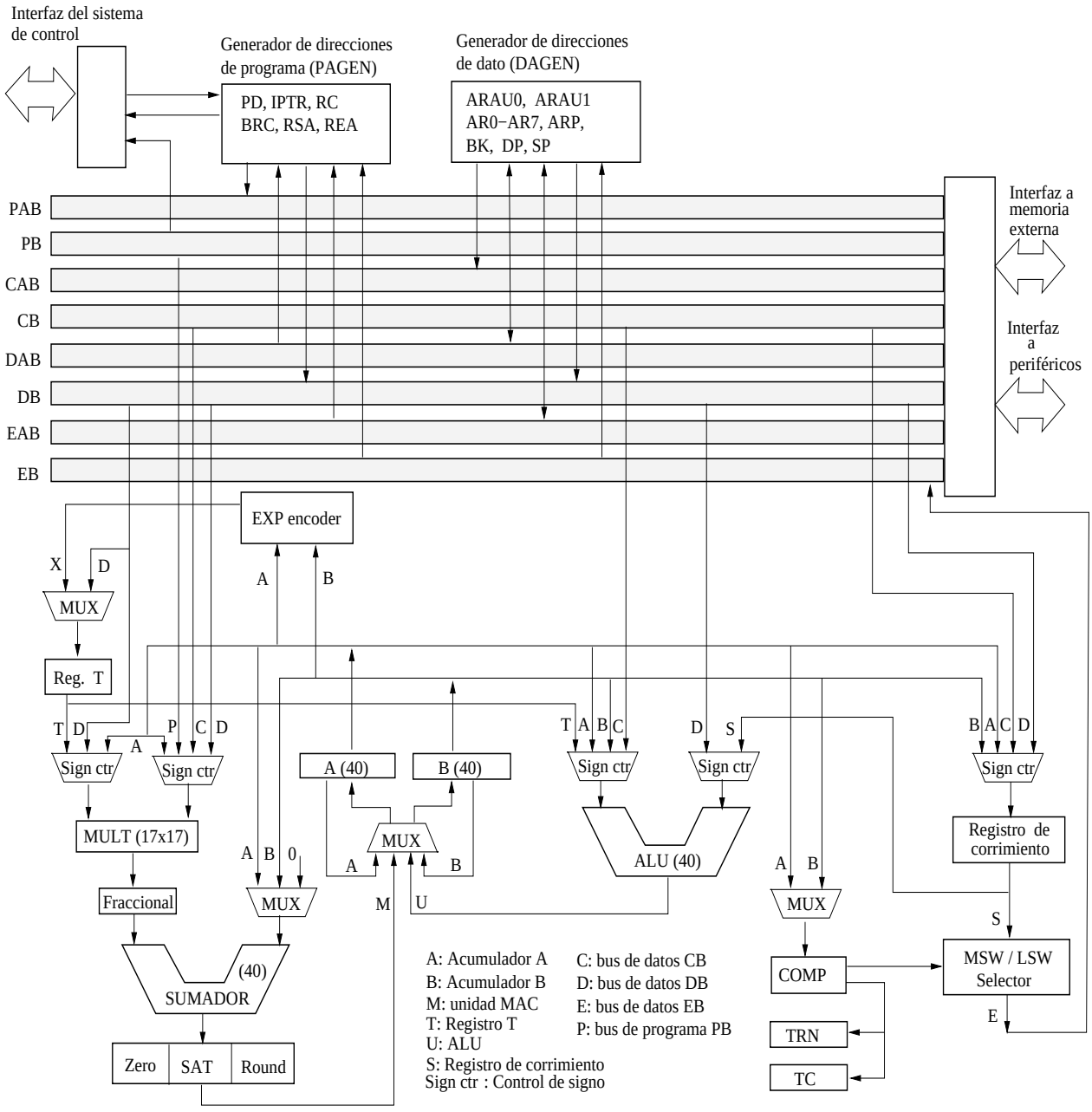


Figura 1.1: Arquitectura de los DSPs TMS320C54x y TMS320C54xx.

1.1.2. Organización de la memoria

La memoria del C54x está organizada en tres espacios: 64 K-palabras para memoria programa, 64 K-palabras para datos y 64 K-palabras para puertos I/O, estos espacios de memoria se pueden seleccionar por medio de las señales /PS, /DS e /IS respectivamente (pines externos). La memoria puede ser del tipo RAM o ROM, la memoria RAM puede ser SARAM (de simple acceso por ciclo de máquina) y DARAM (de doble acceso por ciclo de máquina). En el cuadro 1.1 se muestra la distribución del mapa de memoria para los DSPs TMS320C542/C543.

Memoria Programa			**Memoria Dato**	
OVLY = 1	0000h-007Fh	Reservado	0000h-005Fh	Registros Mapeados
	0080h-27FFh	DARAM Interna	0060h-007Fh	DARAM
OVLY = 0	0000h-27FFh	Externa	0080h-27FFh	DARAM Interna
	2800h-EFFFh	Externa	2800h	
MP/mc =0	F000h-F7FFh	Reservado		
	F800h-FF7Fh	ROM Interna		
	FF80h-FFFFh	Vectores de Interrupción		Externa
MP/mc =1	F000h-FF7Fh	Externo		
	FF80h-FFFFh	Vectores de Interrupción	FFFFh	

Cuadro 1.1: Mapa de memoria de los DSP C542 y C543.

La familia del DSP C54x contiene al menos una memoria ROM de 2 K-palabras mapeada en la dirección F800h a FFFFh que se distribuye:

- Un programa bootloader que levanta desde puerto serie, memoria externa, puertos I/O o interface de host.
- Una tabla de la Ley μ de 256 valores (localidades FC00-FCFFh).
- Una tabla de la Ley A de 256 valores (localidades FD00-FDFFh).
- Una tabla de la función seno de 256 valores para realizar la FFT (localidades FE00-FEFFh).

- Una tabla de vectores de interrupción, el vector de reset está mapeado en localidad FF80h.
- 26 registros mapeados de acceso en un ciclo (0000h a 001Eh). Dentro de éstos están los registros de estado ST0, ST1 y PMST. Las tres partes de los acumuladores A y B también son registros mapeados (AG, AH, AL, BG, BH y BL)
- Registros de control y datos de periféricos (0020h a 005Fh).

Al momento del reset la memoria RAM: DRAM y SARAM no están disponibles en el espacio de programa. La memoria RAM puede direccionarse en el espacio de programa si se pone en "uno" el bit OVLY del registro PMST:

Si OVLY = 0, la memoria RAM es mapeada sólo en el espacio de dato.
Si OVLY = 1, la memoria RAM es mapeada en ambos espacio, dato y programa.

Organización de la memoria ROM en bloques

La memoria ROM está subdividida y organizada en bloques para mejorar el desempeño. Esta organización de la memoria permite la búsqueda de una instrucción de un bloque de memoria ROM sin sacrificar el acceso de datos que provienen de otro bloque diferente de memoria ROM. Por ejemplo:

- en el DSP C541 la memoria ROM está mapeada en las direcciones 9000h-FFFFh y se divide en bloques de 4K palabras,
- en los DSPs C542 y C543 la memoria ROM está mapeada en las direcciones F7FFh-FFFFh y sólo tiene ese bloque único de 2K palabras,
- en los DSPs C545/C546 su memoria ROM está mapeada en las direcciones 4000h-FFFFh y se divide en bloque de 4K palabras y
- en el DSP C548 su memoria ROM está mapeada en las direcciones F7FFh-FFFFh como bloque único de 2K palabras.

Organización de la Memoria RAM interna

La *memoria ROM* está subdividida y organizada en bloques para mejorar el desempeño, esta organización permite la búsqueda de dos operandos de un bloque de memoria DARAM y la escritura en otro bloque DARAM en el mismo ciclo de instrucción.

- En el DSP C541 *la memoria DARAM* está mapeada en las direcciones 0000h-13FFh y se divide en cinco bloques de 1K palabras,
- en los DSPs C542 y C543 la memoria DARAM está mapeada en las direcciones 0000h-27FFh y con cinco bloques de 2K palabras
- en los DSPs C545 y C546 su memoria DARAM está mapeada en las direcciones 0000h-17FFh y se divide en un bloque de 2K palabras
- en el DSP C548 su memoria DARAM está mapeada en las direcciones 0000h-1FFFh con cuatro bloques de 2K palabras, además el C548 tiene memoria SARAM en 2000h-7FFFh dividida en tres bloques 8K palabras.

Para toda la familia C54x en la página cero 0000h-007Fh (DARAM), se encuentran:

- en localidades 0000h-1Fh los 26 registros mapeados del CPU (cuadro 1.2).
- en localidades 0020h-5Fh los registros mapeados de periféricos.
- en localidades 0060h-7Fh para almacenamiento de variables que ayudan a evitar la fragmentación de bloques largos de memoria RAM (DP=0).

1.1.3. Unidad Central de Proceso (CPU)

En esta unidad es donde se realizan la mayor cantidad de cálculos que realiza el DSP, esta unidad está compuesta de las siguientes partes:

- La unidad aritmético lógica (ALU) de 40 bits.
- Dos acumuladores (A y B) de 40 bits, éstos almacenan la salida de la ALU o del bloque multiplicador/sumador. Cada acumulador está dividido en tres partes:
  - Bits de guarda (bits 39-32, AG o BG).
  - Bits parte alta (bits 31-16, AH o BH).
  - Bits parte baja (bits 15-0, AL o BL).
- Registro para corrimientos de 0-31 bits a la izquierda (+) y de 0-16 bits a la derecha (-). Este registro junto con el codificador de exponente normalizan el acumulador en un ciclo de instrucción.

Dirección	Nombre	Descripción
0	IMR	Registro de máscara de interrupción
1	IFR	Registro de banderas de interrupción
2-5	—	Reservado
6	ST0	Registro de estado 0
7	ST1	Registro de estado 1
8	AL	Acumulador A parte baja (bits 15-0)
9	AH	Acumulador A parte alta (bits 31-16)
A	AG	Acumulador A bits de guarda (bits 39-32)
B	BL	Acumulador B parte baja (bits 15-0)
C	BH	Acumulador B parte alta (bits 31-16)
D	BG	Acumulador B bits de guarda (bits 39-32)
E	T	Registro temporal
F	TRN	Registro de transición
10	AR0	Registro auxiliar 0
11	AR1	Registro auxiliar 1
12	AR2	Registro auxiliar 2
13	AR3	Registro auxiliar 3
14	AR4	Registro auxiliar 4
15	AR5	Registro auxiliar 5
16	AR6	Registro auxiliar 6
17	AR7	Registro auxiliar 7
18	SP	Apuntador de pila
19	BK	Registro del tamaño del buffer circular
1A	BRC	Contador de repetición de bloque
1B	RSA	Dirección inicial de repetición de bloque
1C	REA	Dirección final de repetición de bloque
1D	PMST	Registro de estado, para modos del procesador
1E	XPC	Registro de extensión del contador de programa sólo en el C548
1E-1F	—	Reservado

Cuadro 1.2: Registros mapeados del DSP C54x.

- Multiplicador de 17x17 bits en complemento a dos.
- Sumador de 40 bits.
- Unidad de comparación, selección y almacenamiento (CSSU).
- Unidad generadora de dirección de datos.
- Unidad generadora de dirección de programa.
- Registros de estado y control.

1.1.4. Periféricos Internos

- Pines de propósito general (/BIO de entrada y XF de salida) para manejo de dispositivos externos.
- Generador de estados de espera por software.
- Banco de switches lógicos programables.
- Interfaz de puerto huésped (HPI) de 8 bits paralelos que provee una interfaz a un procesador huésped.
- Temporizador por hardware de 16 bits con cuatro bits preescaladores.
- Generador de reloj que consiste de un oscilador interno y un circuito PLL.
- Puertos serie: puerto serial síncrono, puerto serial buffereado y puerto TDM (depende de la versión "x" del C54x).

1.1.5. Modos de Direccionamiento

Son las formas de transferir los operandos para lectura, almacenamiento y operación en el DSP.

- Inmediato: una constante viene en el código de instrucción.
- Absoluto: la instrucción codifica una dirección fija.
- De Acumulador: utiliza el acumulador A parte baja (AL) para accesar una localidad en memoria programa como dato.

- Directo: similar al C5x, divide la memoria dato en páginas apuntadas por el apuntador de página (DP) o el apuntador de pila (SP).
- Direccionamiento indirecto: similar al C5x, utiliza registros ARi y dos unidades ARAU. En la familia C54x y C54xx no es necesario el registro apuntador de registros auxiliares (ARP), ya que el ARi correspondiente con modificador se utiliza directamente en la instrucción.
- Registros mapeados.
- De stack: agrega o remueve datos del sistema de stack.

1.2. La Unidad Central de Proceso

La podemos dividir en las siguientes unidades:

- Unidad Aritmética Lógica (ALU)
- Registro de corrimiento
- Unidad Multiplicador/Sumador (MAC)
- Codificador de Exponente
- Unidad de comparación, selección y almacenamiento

1.2.1. La Unidad Aritmética Lógica (ALU)

La unidad ALU efectúa operaciones aritméticas y lógicas a 40 bits casi siempre en un ciclo de instrucción, el resultado de las operaciones se escribe en los acumuladores A o B.

En la figura 1.2 se observa que la unidad ALU tiene dos operandos de entrada X e Y con diferentes fuentes:

- entradas X :
  - Del registro de corrimiento (operando de 16 o 32 bits de memoria o acumuladores respectivamente).
  - Dato de memoria del bus DB.

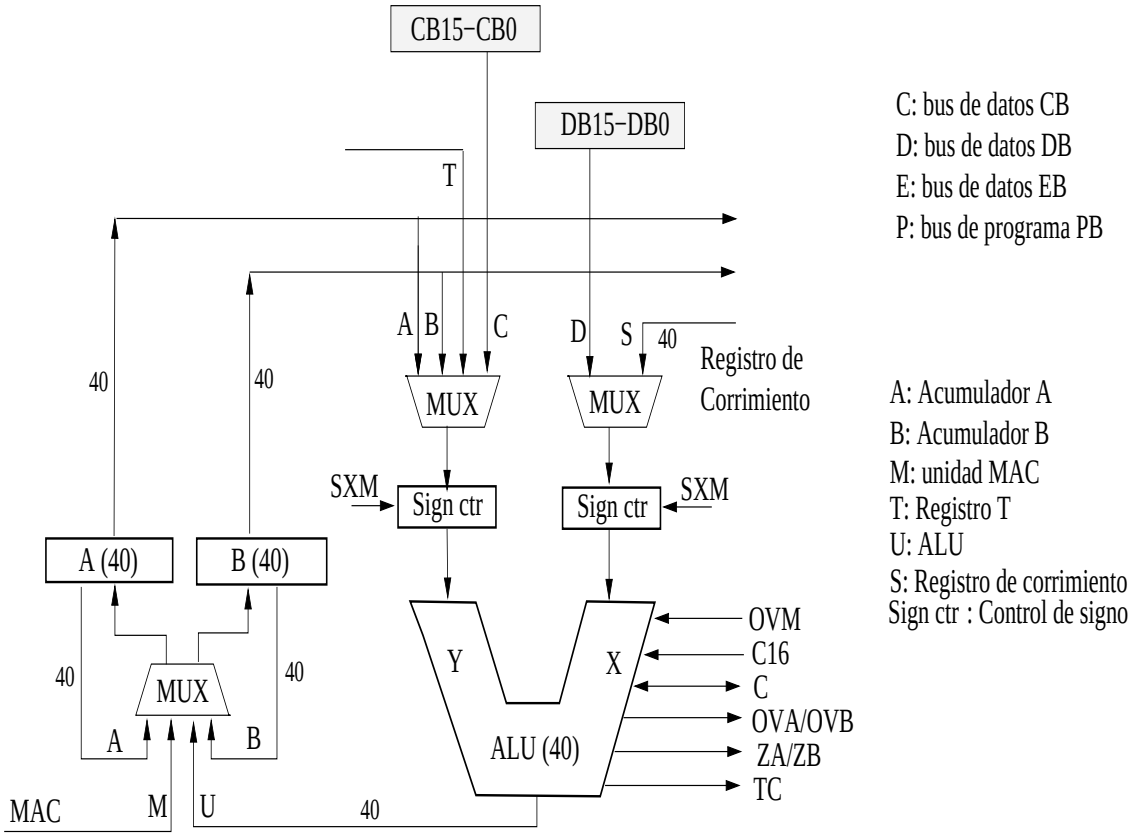


Figura 1.2: Unidad ALU del DSP TMS320C54x.

- entradas Y:
  - Valores de los acumuladores A o B.
  - Dato de memoria del bus CB.
  - Valor en el registro temporal T.

Como se observa en la figura 1.2, dependiendo de las instrucciones, la unidad ALU manipula las banderas de estado de entrada OVM, C16 y C, y también afecta las banderas de estado de salida C, OVA/OVB, ZA/ZB y TC.

1.2.2. Registro de corrimiento

El registro de corrimiento es utilizado para realizar operaciones tales como:

- Preescalar operandos de entrada de la memoria o un acumulador antes de operar en la unidad ALU.
- Efectuar corrimientos lógicos o aritméticos de los acumuladores A o B.
- Normalizar el acumulador para implementar operaciones en punto flotante.
- Post-escalamiento de un acumulador antes de almacenarlo en memoria.

Estas operaciones de corrimiento se observan en la figura 1.3.

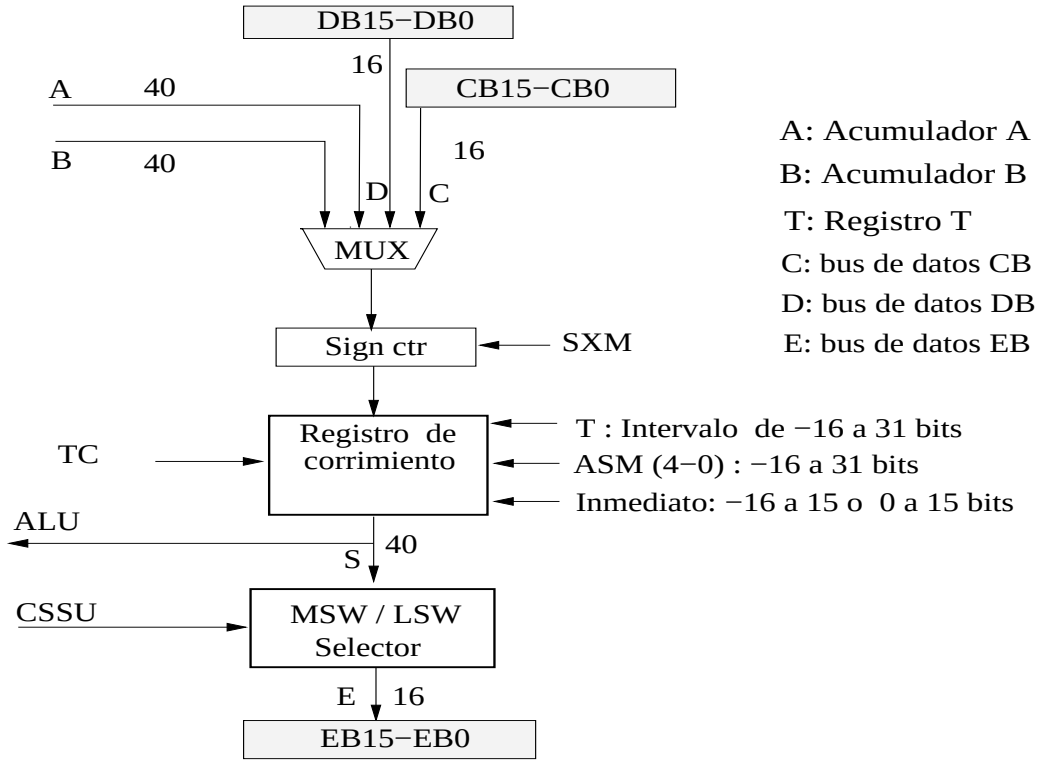


Figura 1.3: Registro de corrimiento del DSP TMS320C54x.

1.2.3. Unidad Multiplicador/Sumador

Esta unidad consta de un multiplicador en paralelo de 17x17 bits acoplado a un sumador dedicado, lo que conforma la unidad de multiplicación acumulación (MAC) de 40 bits (figura 1.4). El multiplicador puede efectuar multiplicaciones signadas, no signadas y signada/no signada con las limitaciones:

- En multiplicación con signo, cada operando de 16 bits en memoria es asumido como una palabra de 17 bits con extensión de signo.
- En multiplicación sin signo, se agrega un 0 al bit 16 (MSB) a cada operando de entrada (operandos de 17 bits).
- En multiplicación con signo/sin signo, uno de los operandos es de signo extendido y al otro se le agrega un 0 al bit 16 (MSB).

La salida del multiplicador se puede correr un bit a la izquierda para compensar el bit de signo extra generado por el multiplicador, esto se efectúa fijando el bit FRCT en "1" en el registro de estado ST1.

La unidad MAC contiene a la salida un detector de cero, un modo de redondeo y una lógica de saturación. El redondeo consiste en sumar 2^{15} al resultado (MAC) y limpiar los 16 bits bajos.

La fuente de datos del multiplicador viene seleccionado en las instrucciones como se ven en la figura 1.4, la fuente XM puede ser:

- El registro temporal T.
- Un dato de memoria que viene por el bus DB.
- El acumulador A parte alta, bits 32-16.

Para la fuente YM:

- Datos de memoria por el bus DB.
- Datos de memoria por el bus CB.
- Operando inmediato de memoria programa por el bus PB.
- El acumulador A parte alta, bits 32-16.

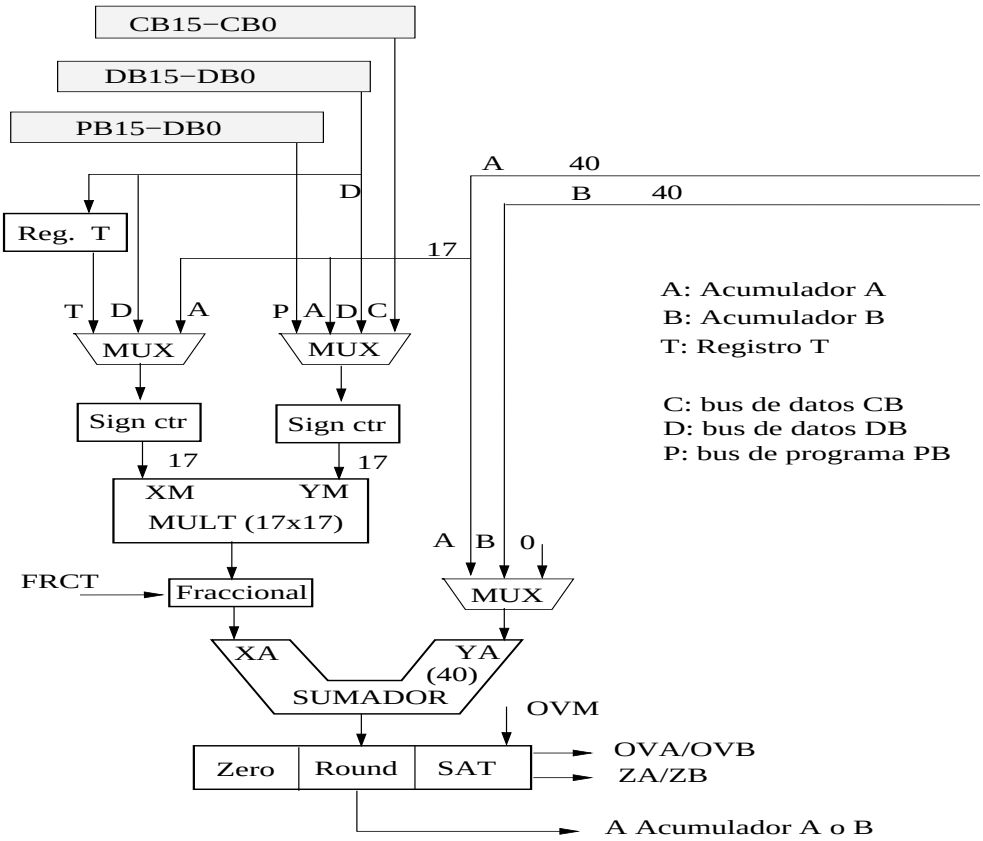


Figura 1.4: Unidad Multiplicador/Sumador del DSP TMS320C54x.

En instrucciones que utilizan al registro T como una entrada, la segunda entrada puede ser obtenida de memoria dato por el bus DB o del acumulador A.
En instrucciones que usan un operando simple de memoria dato, un operando es alimentado al multiplicador por el bus DB, y el segundo operando puede venir del registro T, o como un valor inmediato de memoria programa por PB o del acumulador A.
En instrucciones que usan direccionamiento doble de operandos de memoria dato, los buses DB y CB llevan los datos al multiplicador.

1.2.4. Unidad de comparación, selección y almacenamiento

La unidad de comparación, selección y almacenamiento (CSSU) es una unidad de aplicación específica de hardware dedicada a la operación suma/comparación/selección (ACS) del operador Viterbi. La unidad CSSU del C54x soporta varias formas del algoritmo Viterbi en la igualación y decodificación de canal (figura 1.5).
El operador Viterbi es ejecutado en la ALU, esta función consiste en la doble adición de 16 bits entre dos operandos a 32 bits, esta adición es completada en un ciclo de máquina, si la ALU está configurada para la suma dual fijando el bit C16 en el registro de estado ST1, las palabras de 32 bits se convierten en aritméticas de 16 bits. Para ejecutar este tipo de operación el C54x tiene instrucciones especiales que empiezan con la letra D (de doble).

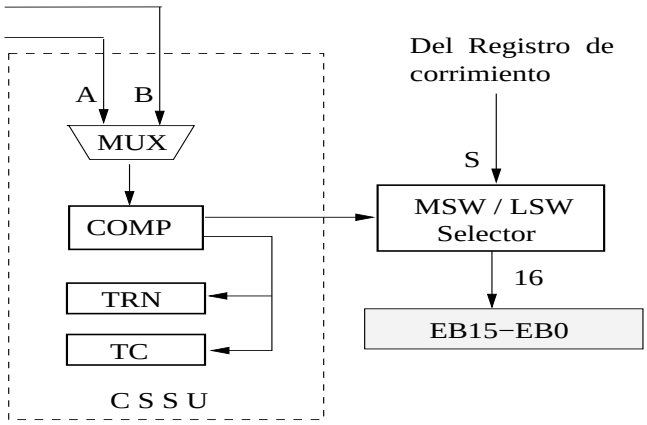


Figura 1.5: Unidad CSSU del DSP TMS320C54x.

1.2.5. Codificador de Exponente

Es una aplicación específica de hardware dedicada a ejecutar la instrucción EXP en un ciclo de instrucción. Esta instrucción extrae el valor del exponente del acumulador y le resta ocho, este valor es almacenado en el registro T en complemento a dos en un intervalo de -8 a +31. Este número corresponde a la cantidad de corrimientos (+ a la izquierda, - a derecha) requeridos en el acumulador (40 bits) para eliminar los bits menos significativos a excepción del bit de signo, es decir, que extrae el exponente para representar al ACC en formato q_i máximo para representar su mantisa (q_{31}). Las instrucciones EXP y NORM son utilizadas para codificar el exponente y normalizar el contenido del acumulador eficientemente. La instruc-

ción NORM extrae la mantisa del ACC, haciendo los corrimientos de acuerdo al exponente almacenado en el registro T.

1.3. Modos de direccionamiento

Los modos de direccionamiento utilizados por éstos DSPs le dan una gran potencialidad y versatilidad a la utilización de la arquitectura y la ejecución de las operaciones.

1.3.1. Inmediato

En este modo una constante específica es el operando de la instrucción, y esta constante se codifica en un campo del código de la instrucción. Para este modo el símbolo # precede a la constante o un símbolo. Una constante corta puede ser de 3,5,8 y 9 bits y una constante larga es 16 de bits, es decir, que en este último caso el código de instrucción puede ser de dos palabras.

Ejemplos:

```
LD   #100h,A     ; Carga la constante 100h al acumulador A
ADD  #20,A,A     ; Suma al acumulador A una constante, A=A+20
```

1.3.2. Absoluto

En este modo la instrucción codifica una dirección fija en el mapa de memoria. Existen cuatro posibilidades:

- Direccionamiento a memoria dato (dmad), direcciona un valor específico en memoria dato.
- Direccionamiento a memoria programa (pmad), direcciona un valor específico en memoria programa.
- Direccionamiento a puerto I/O, direcciona un valor específico en un puerto, utiliza instrucciones PORTR y PORTW
- *(lk), utiliza un valor para especificar una dirección en el espacio de dato. La sintaxis *(lk) utiliza un símbolo o un número para indicar una dirección en el espacio de dato. En este modo la longitud de la palabra de instrucción crece en un palabra, este direccionamiento no se puede utilizar con instrucciones RPT y RPTZ.

Ejemplo:

```
STL   B,#y  ; Almacena el acumulador B en la dirección absoluta de y
```

1.3.3. De Acumulador

Utiliza el acumulador A (parte baja) para accesar una localidad en memoria programa como dato. Específicamente se utiliza en dos instrucciones: READA Smem, que transfiere una palabra de memoria programa especificada por el ACC A parte baja a una localidad de memoria dato especificada por Smem (memoria dato de simple acceso) y WRITA Smem, que transfiere un word de memoria dato especificada por Smem a una localización de memoria programa especificada por el ACC A.

1.3.4. Direccionamiento directo

Este modo es similar al DSP C50, divide la memoria dato en páginas apuntadas por el apuntador de página (DP) o el apuntador de pila (SP). La instrucción contiene el offset o desplazamiento de 7 bits de la dirección del dato a transferir. Con este modo de direccionamiento se pueden accesar a 128 localidades en memoria RAM sin cambiar DP o SP.

La selección de DP o SP se hace:

- Si el bit de modo compilación (CPL) del registro ST1 es cero, se selecciona el apuntador de página DP, en este caso la memoria dato se divide en 512 páginas de 128 localidades cada una. Una dirección efectiva de 16 bits se forma de los 9 bits de la página con los 7 bits del offset.
- Si CPL=1, se selecciona el contenido del stack pointer (SP) como dirección de referencia. En este caso la dirección efectiva es la suma de SP más el offset, donde el SP apunta a una dirección en memoria, se pueden accesar a 128 localidades tomando como dirección base SP.

Ejemplo:

```
LD   #x,DP     ; Carga el apuntador de página DP con la página de x
LD   #200,A    ; A = 200
ADD  #120,A,B  ; B=A+120
STL  B,@x      ; almacena el acumulador B en la loc. de variable x
```

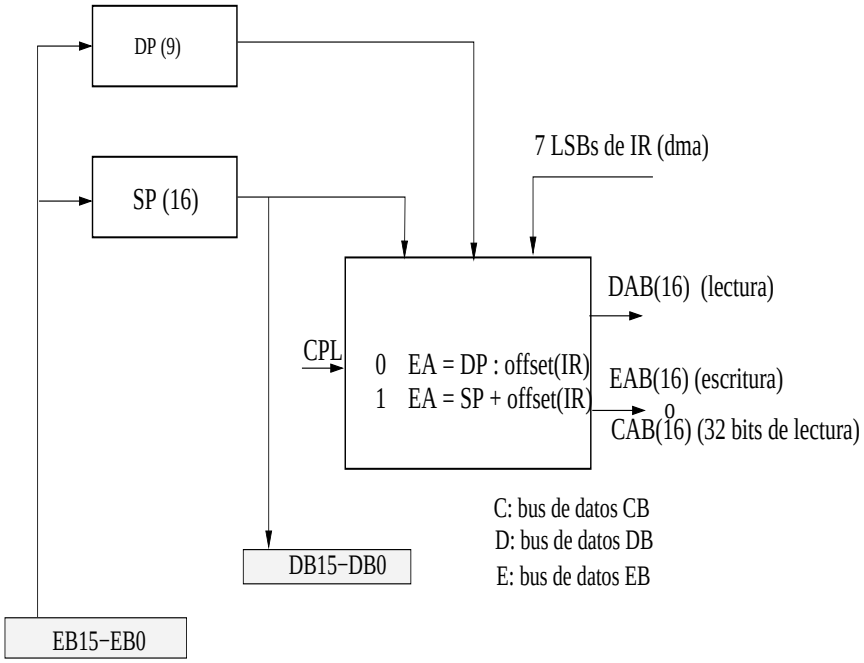


Figura 1.6: Direccionamiento directo del DSP TMS320C54x.

1.3.5. Direccionamiento indirecto

Es similar al DSP C50, cualquier dirección de 16 bits contenida en un registro auxiliar ARi (AR0-AR7) puede ser accesada. Este modo permite accesar dos datos en un ciclo de instrucción. Diferente al C50, en el C54x se indica directamente el ARi a utilizar y su modificador (ver cuadro 1.3).

Este modo utiliza los registros auxiliares ARi (AR0-AR7) y las unidades aritméticas de registros auxiliares ARAU0 y ARAU1, que efectúan aritmética no signada a 16 bits sobre los registros ARi (figura 1.3). Con los registros auxiliares pueden efectuarse las operaciones:

- Cargarse con un valor inmediato utilizando instrucción STM.
- Cargarse de memoria por el bus de datos.
- Modificarse en el mismo ciclo de instrucción a través de unidades ARAUs.
- Modificarse por instrucción MAR (modifica registro auxiliar).

- Utilizarse como contador de ciclo utilizando instrucción BANZ[D].

Ejemplo:

```
STM #x,AR1        ; AR1 se carga con la dirección de localidad x
STM #y,AR2        ; AR2 se carga con la dirección de localidad y
STM #z,AR3        ; AR3 se carga con la dirección de localidad z
LD  #0,A          ; A=0
LD  #0,B          ; B=0
ADD *AR1+,A       ; A=A+x, AR1=AR1+1
ADD *AR2,B        ; B=B+y
ADD #10,A,B       ; B=A+10= x + 10
STL B,*AR2+10     ; Almacena Acc. B en localidad y+10
ADD *AR2,*AR3,A   ; A=A+y+z = x + y + z
STL A,*AR1        ; Almacena Acc. A en localidad x+1
```

En modo de direccionamiento indirecto, el DSP C54x utiliza como desplazamiento el registro AR0, este registro también se carga con el valor de N/2 para direccionamiento en acarreo inverso.

Direccionamiento Circular

Este modo se puede decir que es un modificador del direccionamiento indirecto, en este caso el registro mapeado BK contiene el tamaño del buffer circular. Todo buffer circular de tamaño R debe de empezar en límites (fronteras) de N bits, es decir, los N bits LSB de la dirección base del buffer circular deben ser cero, donde N es un número entero menor que satisface $2^N > R$. El valor de R es cargado en el registro de tamaño de buffer circular BK.

Para el uso de buffer circular se toma en cuenta las siguientes reglas:

- Poner la dirección (parte baja) del buffer circular en una frontera 2^N, donde 2^N es mayor que el tamaño del buffer circular.
- Utilizar un paso menor o igual al buffer circular.
- La primera vez que se utiliza el buffer circular, el ARi a usar debe estar dentro del buffer.

Operando	Función	Descripción
*ARi	dir=ARi	ARi contiene una dirección de memoria dato
*ARi-	dir=ARi ARi=ARi-1	Post-decrementa ARi después de accesar el dato
*ARi+	dir=ARi ARi=ARi+1	Post-incrementa ARi después de accesar el dato
*+ARi	dir=ARi ARi=ARi+1	incrementa ARi antes de accesar dato
*ARi-0B	dir=ARi ARi=B(ARi-AR0)	después de accesar el dato, al ARi se le resta AR0 en acarreo inverso (para efectuar la FFT)
*ARi-0	dir=ARi ARi=ARi-AR0	después de accesar el dato, al ARi se le resta AR0
*ARi+0	dir=ARi ARi=ARi+AR0	después de accesar el dato, al ARi se le suma AR0
*ARi+0B	dir=ARi ARi=B(ARi+AR0)	después de accesar el dato, a ARi se le suma AR0 en acarreo inverso (para efectuar la FFT)
*ARi-%	dir=ARi ARi=circ(ARi-1)	Post-decrementa ARi después de accesar el dato en direccionamiento circular
*ARi-0%	dir=ARi ARi=circ(ARi-AR0)	Post-decrementa ARi en AR0 después de accesar el dato en direccionamiento circular
*ARi+%	dir=ARi ARi=circ(ARi+1)	Post-incrementa ARi después de accesar el dato en direccionamiento circular
*ARi+0%	dir=ARi ARi=circ(ARi+AR0)	Post-incrementa ARi en AR0 después de accesar el dato en direccionamiento circular
*ARi(lk)	ARi=ARi+lk dir=ARi	la dirección del dato es ARi más una constante de 16 bits, ARi no cambia
*+ARi(lk)	dir=ARi+lk ARi=ARi+lk	la dirección del dato es ARi más una constante de 16 bits, ARi se actualiza
*+ARi(lk) %	dir=circ(ARi+lk) ARi=circ(ARi+lk)	la dirección del dato es ARi más una constante de 16 bits en direc. circular, ARi se actualiza
*(lk)	dir=lk	Una constante de 16 bits es utilizada como dirección absoluta

Cuadro 1.3: Modificadores de direccionamiento indirecto.

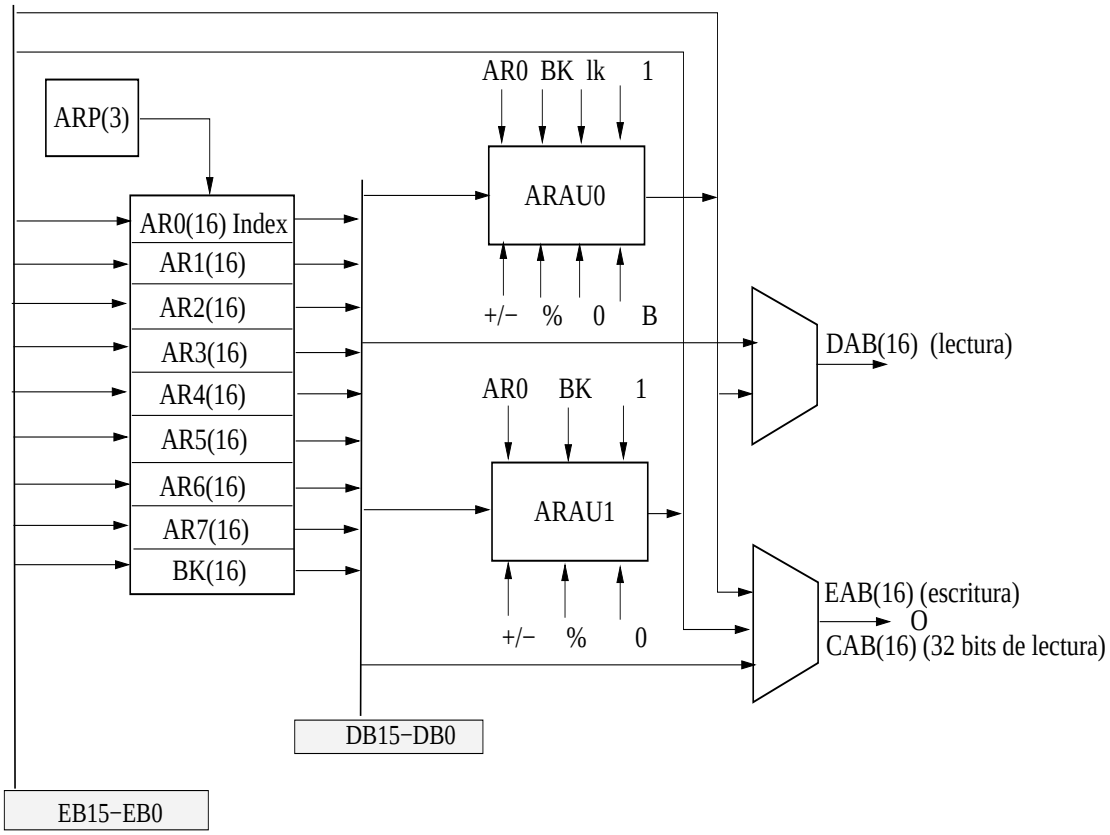


Figura 1.7: Direccionamiento indirecto del DSP TMS320C54x.

El algoritmo de direccionamiento de buffer circular es:

```
if ( 0 <= ARi + paso < BK )
     ARi = ARi + paso
elseif ( ARi + paso >= BK )
     ARi = ARi + paso - BK
elseif (ARi + paso < 0)
     ARi = ARi + paso + BK
end
```

1.3.6. Direccionamiento de Registros mapeados

Es similar al DSP C50, este modo de direccionamiento se utiliza para modificar los registros mapeados (página 0) sin modificar el contenido de DP o SP. Este modo utiliza los siete bits menos significativos de una dirección para seleccionar un dato en memoria. En direccionamiento indirecto utiliza únicamente los siete bits LSB del registro ARi para acceder a la localidad en memoria.

Ejemplo:

```
ADD  DRR,0,A      ; A=A+ Registro DRR
```

Resumen

En este capítulo se ha descrito las partes principales de las arquitecturas de DSPs C54x y C54xx, sus buses, sus unidades, el mapa de memoria y los modos de direccionamiento, es importante señalar que el conocimiento de estos conceptos aunados a la teoría son fundamentales para entender los demás capítulos de este trabajo y realizar aplicaciones con estas familias.

Capítulo 2

Pipeline

El pipeline es una técnica de procesamiento en paralelo que se utiliza ampliamente en los DSPs. Esta técnica consiste en solapar las operaciones de los buses durante el proceso de ejecución de las instrucciones, esto puede observarse como una cadena de producción de autos donde existen procesos de ensamblado que se realizan sobre el auto desde que entra hasta que sale ya fabricado. En una arquitectura digital que opere con pipeline, las instrucciones son transferidas a estados sucesivos, donde unidades separadas de hardware realizan diversas operaciones de los buses para completar la ejecución de la instrucción. Las arquitecturas que operan bajo este concepto son conocidas de procesamiento vectorial, y su eficiencia solo es posible si las operaciones a ejecutar son vectorizadas, es decir, si es factible arreglarlas en un flujo continuo de datos. Algunas dificultades que presenta el pipeline son las ramificaciones que puedan existir en un programa, ya que se pierde la secuencia de los procesos sobre una secuencia de instrucciones que van en desarrollo.

El TMS320C54x opera con seis niveles de Pipeline: prebúsqueda, búsqueda, decodificación, acceso, lectura de operandos y ejecución, donde un contador de prebúsqueda contiene la dirección de la próxima instrucción a ser prebuscada. Una vez que una instrucción es buscada, entonces es cargada en un Registro de Instrucción (IR), al menos que IR contenga la instrucción en ejecución. Los niveles de pipeline son independientes entre sí y permiten solapar la ejecución de instrucciones (ver figura 2.1).

2.1. Operación pipeline a seis niveles

Los seis niveles de pipeline del C54x son independientes uno de otro, lo que permite solapar la ejecución de las instrucciones, es decir, que en un ciclo dado, cada una de las

instrucciones está activa en diferentes estados de pipeline.

1. **Búsqueda de programa: (prefetch)**
   El bus de dirección de programa (PAB) se carga con la dirección de la siguiente instrucción a ser buscada.

2. **Búsqueda (fetch):**
   Una palabra de instrucción es buscada a través del bus de programa (PB) y es cargada en el registro de instrucción (IR). Esto completa la secuencia de búsqueda de instrucción que consiste de este ciclo y el anterior, es decir, que la prebúsqueda y la búsqueda constituyen la secuencia de búsqueda.

3. **Decodificación:**
   El contenido del registro de instrucción IR es decodificado para determinar el tipo de acceso memoria en la unidad generadora de direcciones de datos (DAGEN) y las secuencias apropiadas de control del CPU para la ejecución de la instrucción.

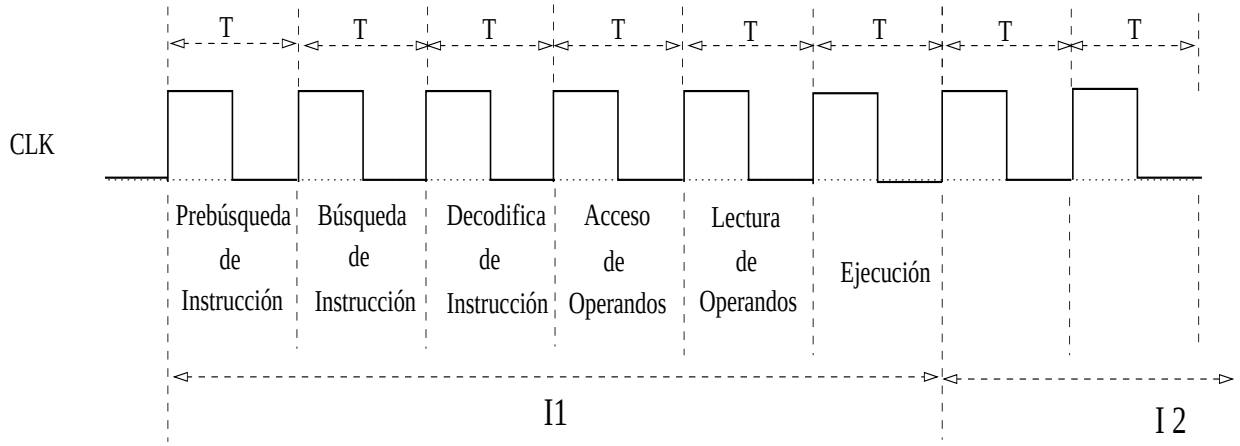
4. **Acceso:**
   La unidad DAGEN emite la dirección del operando a ser leido sobre el bus de direcciones de datos DAB. Si se requiere de un segundo operando, también se emite otra dirección en el bus CAB. En esta etapa del pipeline se actualizan los registros auxiliares y el registro SP. El acceso a memoria siempre es ejecutado en dos fases de pipeline, en la primera el bus de direcciones es cargado con la dirección de memoria y en la segunda fase el bus de datos lee o escribe el dato a memoria.

5. **Lectura:**
   Se leen los operandos necesarios a través de los buses de datos DB y CB. Si se requiere un tercer operando, se carga la dirección de escritura en el bus de direcciones EAB.

6. **Ejecución:**
   Se completa la escritura del operando, escribiendo el dato al bus de escritura del dato (EB), a la vez se ejecuta la instrucción. Cualquier operación de escritura se da sobre los estados de lectura y ejecución, es decir, durante el estado de lectura se pone la dirección del dato y en el estado de ejecución se escribe el dato a memoria.

El registro contador de programa PC contiene la dirección de la próxima instrucción a ser ejecutada, sin ser usada directamente en las operaciones de búsqueda, pero sirve como un apuntador de referencia para la posición actual dentro del programa. El PC es incrementado cada vez que una instrucción se ejecuta. Cuando ocurre una interrupción o una llamada de

CICLO DE MAQUINA EN UNA ARQUITECTURA SECUENCIAL



CICLO DE MAQUINA DE UNA ARQUITECTURA PIPELINE

UNIDADES								
Prebúsqueda de Instrucción	I1	I2	I3	I4	I5	I6	I7	I8
Búsqueda de Instrucción	X	I1	I2	I3	I4	I5	I6	I7
Decodifica Instrucción	X	X	I1	I2	I3	I4	I5	I6
Acceso de Operandos	X	X	X	I1	I2	I3	I4	I5
Lectura de Operandos	X	X	X	X	I1	I2	I3	I4
Ejecución	X	X	X	X	X	I1	I2	I3

Figura 2.1: Niveles de Pipeline del DSP C540.

subrutina, el contenido del PC es guardado en el stack para preservar su contenido al regreso de una interrupción o subrutina. Cuando existen estados de espera, éstos sólo retardan directamente la operación de pipeline. Cada estado de espera insertado durante la operación de búsqueda contribuye a agregar ciclos adicionales de máquina en la ejecución de pipeline de la instrucción.

2.2. Algunos Problemas del pipeline

En la secuencia de programación hay que tener cuidado para evitar problemas de pipeline, ya que la mayoría de cambios ocurren en la fase de ejecución de la instrucción y otros cambios ocurren en otras etapas de pipeline, los registros ARi se modifican en la fase de acceso, por lo que un registro ARi modificado en una instrucción anterior no se puede utilizar en la siguiente instrucción.

Aunque en la mayoría de los casos de las secuencias de programación es difícil que existan problemas de pipeline, sin embargo, si ambos buses CB y DB son utilizados en una instrucción que direcciona dos operandos la cual solapa una instrucción previa que es accesada en el bus E al mismo tiempo, puede generar un conflicto.

2.2.1. Accesos a memoria

- En la lectura del código de una instrucción, éste se lee en la primera mitad del ciclo de fetch.
- Cuando una instrucción lee un operando simple, éste se lee en la primera mitad del ciclo de lectura por el bus DB.
- Cuando una instrucción lee dos operandos, un operando se lee en la segunda mitad del ciclo de acceso a través del bus CB y el otro operando se lee en la primera mitad del ciclo de lectura por el bus DB.
- En una instrucción de escritura de un operando a memoria, la escritura se realiza en la segunda mitad del ciclo de ejecución a través del bus EB.
- Para la escritura de un operando doble (32 bits), la instrucción se realiza en dos ciclos de pipeline, el primer operando se escribe en la primera mitad del primer ciclo de ejecución

y el segundo operando se escribe en la otra primera mitad del primer ciclo de ejecución, ambas escrituras se realizan a través del bus EB.

- Cuando una instrucción realiza una lectura y una escritura, la lectura del operando se realiza en el primer ciclo del ciclo de lectura a través del bus DB, y la escritura del otro operando en la segunda mitad del ciclo de ejecución a través del bus EB.

2.2.2. Acceso a memoria y el pipeline

En teoría, los conflictos de pipeline que pueda ocurrir los resuelve automáticamente el CPU. La memoria RAM interna del C54x está dividida en bloques para mejorar su desempeño de acceso (ver sección de organización de la memoria). Para el *acceso simple a memoria RAM* por diferentes instrucciones consecutivas, donde los operandos están situados en diferentes bloques no ocasionan conflictos de pipelines, mientras que una instrucción en un estado de pipeline está accesando un bloque de memoria, la otra instrucción puede accesar diferentes bloques de memoria en el mismo ciclo sin ningún conflicto. Un conflicto se puede presentar cuando dos instrucciones consecutivas requieran accesar al mismo bloque de memoria. En este caso sólo se efectúa un acceso en un ciclo y el otro acceso es retardado hasta el siguiente ciclo, es decir, se realiza un ciclo de latencia de pipeline.

Algunos conflictos de pipeline para acceso simple

- Instrucción de doble operando:
  Algunas instrucciones tienen dos operandos de escritura o lectura. Si ambos operandos están apuntando al mismo bloque de memoria, ocurre un conflicto de pipeline. En este caso el CPU, automáticamente retarda la ejecución de la instrucción por un ciclo.

  Ejemplo:

```
MAC  *AR2+,*AR3+%,A,B ; instrucción  que se ejecuta en 2 ciclos
                      ; si ambos operandos están en el mismo
                      ; bloque SARAM o DARAM
```

- Instrucción con operandos de 32 bits:
  Las instrucciones de lectura de un operando de 32 bits están diseñadas para que se ejecuten en un sólo ciclo y las instrucciones de escritura de un operando de 32 bits se ejecutan en dos ciclos.

- Conflictos de lectura-escritura:
  Si una instrucción escribe a un bloque de simple acceso y es seguida de una instrucción que lee del mismo bloque de memoria de simple acceso, puede ocurrir un conflicto porque ambas instrucciones tratan de accesar simultáneamente al mismo bloque de memoria. En este caso el acceso de lectura es retardado por un ciclo.

  Ejemplo:

```
STL   A,*AR1+  ; AR1 y AR3 apuntan al mismo bloque SARAM.
LD    *AR3,B   ; esta instrucción toma un ciclo adicional
```

  Si se utilizan instrucciones en paralelo, que leen y escriben un operando como en el caso anterior, éstas no causan conflictos debido a que el acceso doble es realizado en diferentes estados de pipeline.

```
STL   A,AR2+   ; AR2 y AR3 apuntan al mismo bloque SARAM.
||ADD *AR3+,B  ; Esta instrucción se realiza en un sólo ciclo.
```

- Conflictos de código-dato:
  Cuando la memoria SARAM o ROM están mapeadas en el espacio de programa y dato, entonces si una instrucción es buscada en un bloque de memoria y el acceso a datos para lectura o escritura es ejecutado en el mismo bloque de memoria, la instrucción de búsqueda es retardada por un ciclo. Por lo que es conveniente que cada acceso a un bloque de memoria simple sea reservado ya sea para el almacenamiento de datos o programa para evitar estas latencias cada vez que sea accesada en el bloque de datos.

Conflictos no protegidos de pipeline

Este tipo de conflictos se refiere a situaciones que no resuelve o detecta el CPU, por lo que el programador tiene que percatarse. En general, los conflictos no protegidos se solucionan reordenando las instrucciones o introduciendo instrucciones NOP, como también se pueden resolver estos conflictos tratando de no utilizar instrucciones que tengan problemas de pipeline que requieran retardos antes de accesar un registro. Un ejemplo clásico es querer utilizar o actualizar los registros ARi o BK cuando éstos todavía están siendo utilizados, en estos casos lo que se debe hacer es esperarse de dos a tres ciclos de reloj o instrucciones para utilizarlos.

Cuando se utilizan instrucciones de almacenamiento seguidas por alguna instrucción que actualiza los registros ARi, BK o SP en el estado pipeline de lectura, puede ocurrir un conflicto, porque ambas instrucciones tratan de accesar registros de la unidad DAGEN. Los registros de DAGEN pueden ser escritos sólo una vez en un ciclo dado ya que el CPU retarda el estado de lectura por un ciclo.

Reglas para el acceso a registros de DAGEN

Algunas instrucciones actualizan los registros de la unidad DAGEN en el estado de lectura de pipeline, esto puede ocasionar un conflicto si la instrucción anterior trata de actualizar un registro de DAGEN en el estado de ejecución. El CPU resuelve este conflicto automáticamente retardando el estado de lectura en un ciclo. Este retardo puede causar un ciclo adicional de latencia entre la instrucción que escribe al registro en DAGEN en su estado de lectura y la instrucción que sigue a ésta.

El siguiente conjunto de condiciones determinan cuando se pueden presentar tales conflictos:

- La primera instrucción puede ser de los tipos:
  - Instrucción de almacenamiento que acceda a cualquier tipo de registro DAGEN para cargarle un nuevo valor (instrucciones ST y STL).
  - Instrucciones de movimiento que utilizan un registro de DAGEN que se conflictúa con la instrucción previa (instrucciones MVDD, POPM, POPD y DELAY).
- La segunda instrucción inicializa constantes y es del tipo de movimiento como las anteriores, o de movimiento como MVDK y MVMD que escriben a los registros BK, SP o ARi.
- La tercera instrucción utiliza el mismo registro de la segunda instrucción en el modo de direccionamiento indirecto.

Latencia para los registros ARi y BK

Un conflicto de pipeline no protegido puede ocurrir cuando se acceda a un registro ARi o BK en las siguientes condiciones:

- Una instrucción escribe a un registro o BK

- La siguiente instrucción utiliza el mismo registro auxiliar como un apuntador de dirección o index en direccionamiento indirecto, o utiliza BK en el modo de direccionamiento en modo de buffer circular. Ésta instrucción pude ser también una MVMM o una CMPR que lee BK o el mismo ARi.

Estos conflictos suelen ocurrir porque la primera instrucción actualiza a ARi o BK en el estado de lectura o ejecución de pipeline, y la siguiente instrucción utiliza el BK o el mismo ARi cuando está en el estado de acceso de pipeline. El resultado es una lectura incorrecta de ARi o BK de la segunda instrucción, porque la instrucción previa todavía no ha actualizado el contenido del registro. Las instrucciones que puede crean problemas conflictos en DAGEN son las cargas o salvados entre registros mapeados, como también, instrucciones que utilicen una registro mapeado modificado en la instrucción anterior.

Latencias del registro stack pointer SP

Cuando se utiliza el SP pueden ocurrir problemas de latencia en situaciones como:

- El SP se utiliza en modo compilador, CPL=1. En este caso el SP se utiliza como offset para direccionamiento directo y si se encuentran las siguientes condiciones simultáneas pueden haber conflictos:
  - La primera instrucción escribe al registro SP.
  - La siguiente instrucción utiliza al SP como base para el direccionamiento directo. Es obvio que existe un conflicto de pipeline ya que la segunda instrucción utiliza al SP sin que éste se haya actualizado en la instrucción previa.
- Otra situación que ocasiona conflicto al usar SP:
  - La primera instrucción actualiza al SP.
  - La segunda instrucción utiliza el stack en instrucciones PUSH, POP, CALL, RETURN, FRAME o MVMM. Aquí se presenta una situación similar en la que la segunda instrucción utiliza el SP en un estado de pipeline que aún no ha sido actualizado por la instrucción previa.

Latencias de acceso a registros de estado

Cuando se utilicen los bits: ARP, CMPT, CPL, DP, SXM, ASM y BRAF de los registros de estado, pueden ocasionar conflictos de pipeline (ver sección de registros de estado para su significado). Lo más recomendable cuando se modifiquen los bits mencionados es tener en

cuenta que la siguiente instrucción (o varias dependiendo de la latencia) no deben de hacer uso de esos bits, hasta que se efectué tal modificación.

Latencias del registro de estado PMST

Los bist OVLY, DROM, MP/MC e IPTR del registro de estado PMST configuran el espacio de memoria del DSP C54x, por lo que cuando una instrucción modifica alguno de estos bits, se requiere que transcurra un cierto número de ciclos de pipeline antes de que el nuevo espacio de memoria configurado pueda ser accesado. Estos bits son modificados en el estado de lectura o ejecución de pipeline de la instrucción.

Latencias para acceso de registros mapeados al acumulador

Cuando se presentan las dos siguientes condiciones simultáneamente puede presentarse un conflicto de pipeline:

- La primera instrucción modifica el acumulador A o B directamente.
- La segunda instrucción trata de leer una parte del acumulador como un registro mapeado. El conflicto se presenta porque la primera instrucción actualiza el acumulador en el estado de ejecución al mismo tiempo que la segunda instrucción trata de leer el acumulador como un registro mapeado, es decir AG, AH, AL, BG, BH, y BL desde el mapa de memoria en el estado de lectura.

Resumen

Se puede concluir que en la programación del DSP C54x y C54xx es necesario tener cuidado con los conflictos de pipeline para aprovechar al máximo las virtudes de estos DSPs. El programador debe tener presente al menos la problemática abordada en este capítulo o cuando se le presenten situaciones no esperadas puede consultar estas observaciones. Para mayores referencias consultar [34].

Capítulo 3

Sistema de control

Fundamentalmente, el sistema de control del C54x está dado por la lógica del generador de direcciones en memoria programa (PAGEN), el contador de programa (PC), el stack pointer (SP), la señal de reset externo, las interrupciones, los registros de estado y el contador de repetición (RC).

3.1. Generador de direcciones en memoria programa

Esta unidad genera las direcciones para accesar direcciones de instrucciones, tablas de coeficientes, operandos inmediatos de 16 bits y en general información almacenada en memoria programa. Esta unidad consiste de cinco registros:

- PC: contador de programa.
- RC: el contador de repetición.
- BRC: el contador de repetición de bloque.
- RSA: Dirección inicial de repetición de bloque.
- REA: Dirección final de repetición de bloque.

El C548 contiene un registro adicional XPC para direccionar memoria extendida.

3.2. El Contador de Programa

El contador de programa PC es un registro de 16 bits, lo que permite direccionar hasta 64 K direcciones de memoria programa externas o internas en la búsqueda de las instrucciones, un operando inmediato, o una constante de 16 bits en memoria programa. Al direccionar la memoria programa, la dirección que está en el PC es puesta en el bus PAB.

3.3. Saltos y subrutinas

La instrucciones de salto (branch), las llamadas a subrutina y la atención a interrupciones rompen la secuencia en la ejecución de las instrucciones al transferir el control del PC a otra dirección de memoria programa.

Saltos incondicionales

Un salto incondicional siempre se ejecuta, durante su ejecución, el PC es cargado con la dirección de salto e inicia la ejecución de una nueva sección de código. Cuando la instrucción de salto alcanza el estado de ejecución de pipeline, las dos instrucciones siguientes ya han sido buscadas, estas instrucciones se manejan de dos formas:

Sin retardo: Estas dos instrucciones se desvanecen en el pipeline y la ejecución continúa en la dirección de salto.

Con retardo: Se le agrega a la instrucción una letra "D". Dos instrucciones de una palabra o una instrucción de dos palabras son ejecutadas, esto evita el desvanecimiento del pipeline que implica pérdidas de ciclos. Estas dos instrucciones no pueden ser tales que causen cambios en el PC.

```
B[D]  dir_prog  ; Carga el PC con la dirección especificada

BACC[D]  src    ; Carga el PC con los 16 bits bajos de src
                ; acumuladores A o B
```

En modo algebraico:

```
[d]goto etiqueta ; Salto incondicional a etiqueta
```

Para el caso especial del C548, para saltos incondicionales largos se antepone la letra "F" al mnemónico (FB[D] y FBACC[D]) y "far " para el caso algebraico (far [d]goto).

Saltos condicionales

El DSP C54x permite efectuar saltos condicionales con la instrucción BC[D] donde puede probar hasta tres condiciones de la tabla 3.1, el salto se efectúa si todas la condiciones se cumplen, de lo contrario el programa continúa en la siguiente instrucción. Los saltos condicionales requieren uno o más ciclos que los saltos incondicionales.

```
B[D]  dir_prog,Cond1[,Cond2,Cond3] ; Carga el PC con la dirección
                                   ; especificada si se cumplen
                                   ; todas la condiciones
  ; Si se utiliza con retardo [D], se ejecutan las dos siguientes
  ; instrucciones de una palabra, esto evita el desvanecimiento
  ; de pipeline
```

En modo algebraico:

```
if(Cond1[,Cond2,Cond3])
   [d]goto etiqueta                ; Salto condicional a etiqueta
```

Otro tipo de salto condicional que se utiliza para realizar ciclos es la instrucción BANZ[D], ésta sólo evalúa si un registro auxiliar ARi es diferente de cero para efectuar el salto a alguna dirección en memoria programa.

```
BANZ[D] etiqueta,*ARi ; Carga el PC con la dirección de etiqueta
                      ; especificada si ARi es diferente de cero,
                      ; de lo contrario PC=PC+2
            ; Si se utiliza con retardo [D], se ejecutan las dos
            ; siguientes instrucciones de una palabra, esto evita
            ; el desvanecimiento de pipeline.
```

En modo algebraico:

```
if( *ARi != 0) [d]goto etiqueta ; Salto  a etiqueta si ARi es diferente
                                ;  de cero
```

Para el modo algebraico las condiciones y las categorías son las mismas que para el modo ensamblador. Hay que notar que no todas la combinaciones de condiciones son válidas. Para la combinación de las condiciones, éstas se dividen en categorías como se indican en el cuadro 3.2. Sólo se puede tomar una condición de cada categoría, si se utilizan dos condiciones, ambas condiciones deben de ser sobre el mismo acumulador.

Condición	Descripción	Operando
A=0	Acumulador A igual cero	AEQ
B=0	Acumulador B igual cero	BEQ
A≠0	Acumulador A no igual cero	ANEQ
B≠0	Acumulador B no igual cero	BNEQ
A <0	Acumulador A menor que cero	ALE
B <0	Acumulador B menor que cero	BLE
A ≤ 0	Acumulador A menor o igual que cero	ALEQ
B ≤ 0	Acumulador B menor o igual que cero	BLEQ
A >0	Acumulador A mayor que cero	AGT
B >0	Acumulador B mayor que cero	BGT
A ≥ 0	Acumulador A mayor o igual que cero	AGEQ
B ≥ 0	Acumulador B mayor o igual que cero	BGEQ
AOV=1	Existe sobreflujo en acumulador A	AOV
BOV=1	Existe sobreflujo en acumulador B	BOV
AOV=0	No existe sobreflujo en acumulador A	ANOV
BOV=0	No existe sobreflujo en acumulador B	BNOV
C=1	Existe acarreo en ALU	C
C=0	No existe acarreo en ALU	NC
TC=1	Bandera de control en uno	TC
TC=0	Bandera de control en cero	NTC
/BIO=0	Señal externa /BIO=0	BIO
/BIO=1	Señal externa /BIO=1	NBIO
nada	Operación incondicional	UNC

Cuadro 3.1: Condiciones generales.

3.3.1. Instrucciones de llamada a subrutina

Una subrutina es un subprograma que cuando es invocada se transfiere el control temporalmente a otra localidad de memoria programa, en programa principal la dirección (PC+1) de la instrucción siguiente es salvada en la pila, esta dirección es utilizada para el retorno de la ejecución de la subrutina.

Categoría A	Categoría B	Categoría A	Categoría B	Categoría C
EQ	OV	TC	C	BIO
NEQ	NOV	NTC	NC	NBIO
LT				
LEQ				
GT				
GEQ				

Cuadro 3.2: Condiciones por categorías

Llamadas incondicionales a subrutina

Este tipo de llamadas siempre es ejecutada, el PC es cargado con una dirección de programa especificada y la ejecución de la subrutina empieza en esa dirección. La dirección de la subrutina puede ser la segunda palabra de la instrucción o los 16 bits bajos de cualquiera de los acumuladores A o B. Para finalizar la subrutina debe existir una instrucción de retorno (RET), en el retorno de la subrutina se recupera de la pila la dirección (PC+1) de la instrucción siguiente al llamado de subrutina y se continúa con la ejecución del programa. Las llamadas a subrutinas pueden efectuarse con o sin retardo y en este caso se comportan de manera similar a los saltos:

```
CALL[D]  dir_prog  ; Pone en la parte alta de la pila (TOS) la
                   ; dirección de retorno y carga el PC con la
                   ; dirección especificada por dir_prog

CALA[D]  src       ; Pone en la pila la dirección de retorno y
                   ; carga el PC con los 16 bits bajos de los
                   ; acumuladores A o B (src)
```

En modo algebraico:

```
[d]call dir_prog   ; Pone en el TOS de la pila la dirección de
                   ; retorno y carga el PC con la dirección
                   ; especificada  por dir_prog

[d]call  src       ; Pone en el TOS de la pila la dirección de
                   ; retorno y carga el PC con los 16 bits bajos
                   ; de los acumuladores A o B (src)
```

Para el caso especial del C548, para llamadas incondicionales largas se antepone la letra "f" al mnemónico call (FCALL[D] y FCALA[D]) y "far " para el caso algebraico (far [d]call.

Llamadas condicionales a subrutina

Este tipo de llamadas opera similar a las llamadas incondicionales, a excepción de que el control va a ejecutar la subrutina si se cumplen todas las condiciones especificadas, este tipo de llamado también se puede ejecutar con retardo.

```
CC[D]  dir_prog,Cond1[,Cond2,Cond3]     ; Carga el PC con la dirección
                                        ; especificada si se cumplen
       ; todas la condiciones y pone en el
       ; TOS de la pila la dirección de retorno
       ; Si se utiliza con retardo [D], se ejecutan las dos siguientes
       ; instrucciones de una palabra
```

En modo algebraico:

```
if(Cond1[,Cond2,Cond3]) [d]call etiqueta ; Llamada condicional de
                                         ; subrutina
```

Retorno de subrutinas

El retorno de una subrutina es el mecanismo para la finalización correcta de una subrutina y la continuación del programa que la invocó. Este procedimiento es válido para subrutinas normales y subrutinas de interrupción. En el retorno se debe pasar la dirección del TOS de la pila al PC, es decir la dirección de la siguiente instrucción al llamado de subrutina. Como en el caso de saltos y llamados, los retornos también pueden ser incondicionales y condicionales.

Retorno incondicional de subrutinas

Un retorno incondicional siempre se encuentra al final de una subrutina y siempre se ejecuta, y en este caso el PC se carga con la dirección de retorno que está en el TOS de la pila. Este tipo de retorno puede ser con o sin retardo:

```
RET[D]        ; finaliza subrutina y recupera de la pila (TOS)
              ; la dirección de retorno

RETE[D]       ; finaliza subrutina, recupera de la pila (TOS)
              ; la dirección de retorno y habilita
              ; interrupciones mascarables
```

Retorno condicional de subrutinas

Permite que una subrutina tenga varias posibilidades de retorno. Además, se pueden utilizar retornos condicionales para evitar saltos hasta el fin de subrutina. Este tipo de retorno funciona similar a los retornos incondicionales, a excepción de que en éstos se deben cumplir todas las condiciones de la instrucción.

```
RC[D]  Cond1[,Cond2,Cond3] ; Si se cumplen todas las condiciones
                           ; finaliza subrutina y recupera de la
                           ; pila (TOS) la dirección de retorno,
                           ; de lo contrario se sigue ejecutando
                           ; las demás instrucciones de la subrutina
```

Las condiciones que se pueden probar son las mismas dadas en el cuadro 3.1 y las categorías que ya se explicaron. Además, hay que tener en cuenta que en una subrutina siempre debe de existir un retorno no condicional al final de ésta, de lo contrario cuando se ejecute la subrutina se perderá el control del programa.

3.3.2. Ejecución condicional XC

Este tipo de instrucción permite ejecutar las siguientes "n" instrucciones si se cumplen las condiciones de la instrucción, de lo contrario se salta las "n" instrucciones, n=1 o 2:

```
XC  n,Cond1[,Cond2,Cond3] ; Si se cumplen todas las condiciones
                          ; ejecuta las siguientes "n" instrucciones.
                          ; de lo contrario se ejecutan "n"
                          ; instrucciones  NOP (no operación)
```

Las condiciones deben ser establecidas dos ciclos antes que la instrucción XC sea evaluada, esto asegura la toma de las decisiones.

Almacenamiento condicional

Este tipo de instrucciones permiten almacenar los registros A, B, T y BRC en una localidad de memoria si la condición de la instrucción se cumple de lo contrario se continúa en la siguiente instrucción:

```
SACCD  src,Xmem,Cond ; Almacena el acumulador A o B (indicado en src)
                     ; en la localidad de memoria Xmem con corrimiento
```

```
                    ; ASM-16, si la condición se cumple. La condición
                    ; dada es sobre el  acumulador a almacenar

SRCCD  Xmem,Cond    ; Almacena el registro BRC en la localidad de
                    ; memoria Xmem si la condición se cumple.
                    ; La condición dada es sobre el acumulador A o B

STRCD  Xmem,Cond    ; Almacena el registro T en la localidad de
                    ; memoria Xmem si la condición se cumple.
                    ; La condición dada es sobre el acumulador A o B
```

3.4. Registros de Estado

De forma similar al DSP C50, los DSPs C54x y C54xx tienen tres registros de estado mapeados que son: ST0, ST1 y PMST, estos registros contienen algunos estados, ciertas condiciones y modos de operación del DSP. Los registros de estado pueden ser almacenados dentro de la memoria dato y cargados de memoria dato, permitiendo así que el estado de la máquina sea salvado y restaurado para interrupciones y subrutinas. Cada registro contiene conjuntos de bits o bits con diferente función que también son similares al C5x, como se muestra en los cuadros 3.3, 3.4 y 3.5.

Bits	Nombre	Función
15-13	ARP	Apuntador de registros auxiliares
12	TC	Bandera de prueba o control
11	C	Acarreo
10	OVA	Sobreflujo para acumulador A
9	OVB	Sobreflujo para acumulador B
8-0	DP	Apuntador de página en memoria dato

Cuadro 3.3: Registro de estado ST0.

3.5. Instrucciones de repetición

Con la instrucción RPT #N se puede ejecutar en el próximo ciclo N+1 veces la siguiente instrucción, donde la constante N es cargada en el contador de repeticiones RC (la constante

Bits	Nombre	Descripción
15	BRAF	"1" Repetición de bloque activa
14	CPL	"0" DP, apuntador de página para modo directo
		"1" SP, para direccionamiento directo relativo al SP
13	XF	Estado del pin externo de salida XF
12	HM	Modo Hold
		"0" DSP en ejecución interna, buses externos en alta impedancia
		"1" DSP detiene la ejecución interna
11	INTM	"0" Habilita todas las interrupciones mascarables
		"1" deshabilita todas las interrupciones mascarables
10	—	Siempre "0"
9	OVM	Modo de saturación si OVM = "1"
8	SXM	Modo de extensión de signo si SXM = "1"
7	C16	Modo de doble precisión aritmética de la ALU si C16 = "1"
		si C16 = "0" opera en aritmética dual a 16 bits
6	FRCT	Modo fraccional. Corre un bit a la izquierda de la salida
		del multiplicador (unidad MAC)
5	CMPT	Modo de compatibilidad para ARP
4-0	ASM	Modo de corrimiento del Acumulador (de -16 a 15)
		utilizado por instrucciones STH, STL, ADD, SUB y LD.

Cuadro 3.4: Registro de estado ST1.

N puede ser de 8 o 16 bits, la instrucción RPT también puede operar en modo directo e indirecto).

3.5.1. Contador de repetición (RPTC)

Es un registro de 16 bits y cuando es cargado con un número N, causa que la siguiente instrucción sea ejecutada N+1 veces. El RPTC puede ser cargado con un número desde 0 a 65,535 usando las instrucciones RPT #N o RPTZ #N. Estas instrucciones son comúnmente utilizado con instrucciones de multiplicación/acumulación, movimiento de bloques, transferencias de I/O y tablas de lectura/escritura. La instrucción RPTZ limpia el acumulador y el registro multiplicador antes de empezar la repetición de la siguiente instrucción. Cuando la instrucción de repetición (RPT o RPTZ) es decodificada todas las interrupciones mascarables son deshabilitadas, sin embargo el DSP responde a señales de HOLD mientras efectúa un

Bits	Nombre	Descripción
15-7	IPRT	Apuntador de vectores de interrupción a páginas de 128 palabras
6	MP/mc	Modo Microprocesador / Microcomputadora
5	OVLY	"0" RAM en dato, "1" RAM en dato y programa
4	AVIS	"1" visualiza direcciones internas en los buses externos
3	DROM	"1" ROM mapeada en memoria dato
2	CLKOFF	"1" Deshabilita salida CLKOUT, la fija en nivel alto
1	SMUL	"1" Saturación de la multiplicación antes de una MAC
0	SST	"1" Saturación en el almacenamiento de datos

Cuadro 3.5: Registro de estado PMST.

ciclo de repetición. Hay que hacer notar que no todas las instrucciones se pueden utilizar con las instrucciones de repetición de ciclo y algunas no tienen significado

3.5.2. Ciclo de repetición de bloque de instrucciones

Con la instrucción de repetición de bloque de instrucciones RPTB, se puede realizar ciclos tipo "for", para ejecutar un conjunto de instrucciones dentro del ciclo. El registro contador de repetición de bloque BRC es cargado con la cuenta del ciclo desde 0 a 65,535. La instrucción de repetición de bloque RPTB es la que inicia el bloque y carga la dirección de la siguiente instrucción en el registro de inicio de bloque RSA y la dirección de la última instrucción del bloque la carga en el registro REA. Cuando se inicia la ejecución de bloque el bit BRAF se pone en "uno". La dirección de PAER es comparada con el PC, si son iguales entonces el contenido de BRCR es comparado a "cero", si es igual el ciclo termina, de lo contrario éste se decrementa y el PC es cargado con la dirección de REA.

Algunas consideraciones en la repetición de bloques son:

- RPTB si es interrumpible.
- Debido a que usan diferentes registros, instrucciones RPT pueden anidarse dentro de RPTB.
- El anidamiento de RPTB no es recomendado, ya que hay que estar salvando y restaurando el registro BRC, activado y desactivando el bit BRAF.

- Interrupciones y llamadas a subrutina son permitidos dentro de un bloque de repetición.
- Un bloque debe tener al menos tres instrucciones de longitud.

3.6. La pila

El stack o la pila en el DSP C54x y C54xx está ubicada en memoria dato, utiliza un registro apuntador de stack SP para ser accesada. El stack es utilizado cuando se hace un llamado a subrutina, se atiende una interrupción y también puede ser accesado a través de las instrucciones PUSH y POP. Las instrucciones adicionales de PSHD, PSHM, POPM y POPD permiten utilizar el stack para el almacenamiento temporal de la memoria de datos. Los registros ST0, ST1 y PMST tienen asociado un nivel de profundidad en el stack para salvarse automáticamente cuando una interrupción ocurre, en la rutina de atención de interrupción con la instrucciones RETE le devuelve a los registros de estado sus valores originales antes de la interrupción.

3.7. Interrupciones

Las interrupciones son un recurso para suspender la actividad del DSP C54x (o un microprocesador) para responder a algún requerimiento por hardware, esto evita la necesidad de estar encuestando por software un evento externo. Las interrupciones típicas son generadas por dispositivos externos que quieren transferir información al DSP tal es el caso de convertidores A/D, algún puerto serial u otro periférico.. El DSP C54x tiene cuatro interrupciones externas mascarables (INT3, INT2, INT1 e INT0) disponibles para dispositivos externos que interrumpan al DSP, interrupciones internas como las de puerto serie (RINT0, RINT1, XINT0 y XINT1), una para el temporizador (TINT) y otra por software a través de la instrucción TRAP. Cada vector de interrupción consta de cuatro localidades para ubicar una instrucción de salto y la dirección de la subrutina de atención a interrupciones. Dependiendo de la versión "xx" del DSP C54xx, puede tener otras interrupciones para los puertos seriales bufereados, del puerto huésped HPI, etc. También contiene interrupciones no mascarables como la externa NMI y de reset RS.

3.7.1. Operación de las Interrupciones

Cuando una interrupción ocurre, esta es almacenada en un Registro de Bandera de Interrupción de 16 bits (IFR). Cada interrupción es identificada en un bit del registro IFR hasta que ésta es reconocida y automáticamente borrada por el Reconocimiento de Interrupciones (/IACK) o el reset (/RS) o se escribe un "uno" en el bit correspondiente en el registro IFR. El reset no es almacenado en en el registro IFR.

El DSP C54x tiene una mapa de memoria para Registro de Interrupciones Mascarables (IMR) para enmascarar las interrupciones internas y externas. Un "uno" en cualquiera de los bits 0-15 del registro IMR habilita la correspondiente interrupción siempre que el bit de estado INTM esté en cero (habilita interrupciones mascarables) . Obviamente las interrupciones /IMR y /RS no están incluidas en el registro IMR. El modo de interrupción INTM en el registro de status ST0 habilita o deshabilita todas las interrupciones mascarables (INTM=0 habilitado y INTM=1 deshabilitado). INTM es fijado a uno por la señal /IAK. Si una interrupción ocurre durante un ciclo múltiple de instrucciones, la interrupción no puede ser procesada hasta que la instrucción es completada, este mecanismo de protección también es aplicado a instrucciones de ciclo múltiple a través de la señal de READY. Tampoco se permite que se procesen interrupciones cuando una instrucción está siendo repetida por medio de instrucción RPT, la interrupción es identificada en el registro IFR hasta que el ciclo de repetición sea terminado, después la interrupción es procesada.

Las interrupciones no pueden ser procesadas cuando se está borrando el bit INTM, sino hasta la próxima instrucción en la secuencia del programa. También una interrupción no es reconocida cuando el pine externo /HOLD está activado. La instrucción RETE permite terminar una rutina de atención de interrupción, restablecer los registros principales salvados y habilitar interrupciones (INTM = 0) . Cuando una interrupción es ejecutada ciertos registros estratégicos llamados registros "shadows" son salvados automáticamente, y en el retorno de la interrupción (con instrucciones RETE) estos registros son restablecidos.

3.7.2. Interrupciones por software

La instrucción de interrupciones por software (TRAP #k) permite efectuar hasta 32 interrupciones por software con las mismas características de una interrupción por hardware. El programador sólo invoca la rutina de atención a interrupción, lo importante de esta forma de interrupción es que el usuario puede definir sus propias rutinas de atención a interrupción.

Resumen

Hasta este capítulo se han expuesto los fundamentos de hardware y software suficientes para que el lector puede inicializarse en la programación y uso de los DSPs. En este último capítulo se abundó en las estructuras de programación que son herramientas muy poderosas para la programación y sobre todo para hacer grandes aplicaciones al incluir el uso de interrupciones. Por lo que en el siguiente capítulo se introduce el uso de una tarjeta que contiene un DSP para el desarrollo, depuración y evaluación de programas y aplicaciones. Asimismo, en los capítulos de periféricos y aplicación se abundará en el manejo y configuración de interrupciones.

Capítulo 4

Tarjeta DSP Starter Kit Plus

Para el DSP C54x se tiene un ambiente de depuración más interactivo que el C50, este ambiente se ejecuta en Windows para versiones superiores a la 3.11. En la figura 4.1 se puede observar el ambiente de depuración de esta tarjeta. El DSP starter kit (DSK plus) es un paquete de desarrollo, aprendizaje y entrenamiento diseñado para principiantes y diseñadores de aplicaciones utilizando DSPs. Este paquete contiene una tarjeta desarrollo de aplicación independiente que se conecta a la computadora y permite explorar la arquitectura, la operación del CPU y los periféricos del C54x. La tarjeta DSK plus ejecuta el código del C54x en tiempo real mientras el depurador basado en Windows lo analiza línea por línea y despliega información sobre los registros internos del DSP en múltiples ventanas en tiempo real. La interface de comunicación de la tarjeta permite crear nuestro propio código DSP C54x y código del PC anfitrión. El hardware permite el uso de tarjetas de expansión de memoria, periféricos tales como codecs, lógica de interface, otros DSPs o microcontroladores.

4.1. Contenido del paquete y sus características

El paquete para desarrollo contiene:

- La tarjeta para desarrollo DSK plus.
- Cable para el puerto paralelo de la PC (DB-25M a DB-25F).
- Fuente de poder (Entrada 125/250 V, 50/60 Hz; salida: 5-V dc, 3.3 A).
- Depurador Explorador de Código para Windows: GoDSP.
- Lenguaje Ensamblador algebraico para el C54x.

- Programa de auto-prueba.
- Programas para diversas aplicaciones.

Las Características de la tarjeta DSK son:

- Un DSP TMS320C542 de punto fijo.
- 40 MIPS (tiempo para un ciclo de instrucción: 25ns).
- 10 K palabras de memoria RAM de doble acceso (DARAM).
- 2 K palabras de memoria ROM para inicio.
- Un puerto serial multiplexado por división de tiempo (TDM).
- Un puerto serial con buffer (BSP).
- Una interface para puerto huésped (HPI) para las comunicaciones PC-DSP.
- Un programador de tiempos.
- Tres modos de ahorro de energía.
- Un convertidor TLC320AC01 programable y con calidad de voz.
- Una dispositivo lógico PAL22V10.
- Un oscilador.
- Generador de reloj PLL.
- Emulador XDS510.
- Bus de expansión y señales de control para dispositivos externos de entrada y salida.
- Mini jacks mono de 1/8 de pulgada para entradas y salidas analógicas.

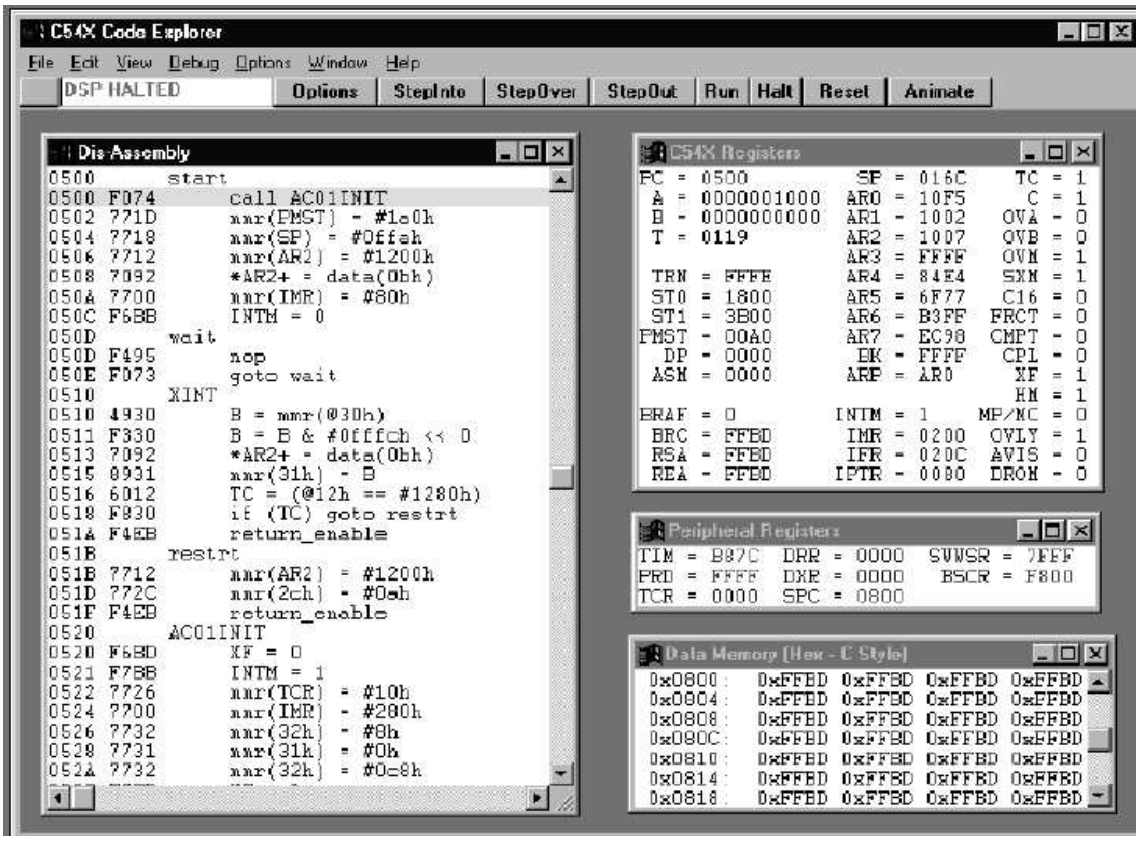


Figura 4.1: Ambiente de desarrollo del Code Explorer.

4.2. Visión general de la funcionalidad

El diagrama del DSK plus se muestra en la figura 4.2, donde se identifica el circuito que sirve de interface para las señales analógicas (el convertidor TLC320AC01), el puerto de emulación XDS510, la interface para un puerto huésped (HPI), el CPU y sus periféricos:

- La lógica para el HPI es un dispositivo PAL integrado en la tarjeta que opera como la interface principal entre el puerto paralelo de la PC y la interface HPI del C542. Como consecuencia de esto, la interface lógica da a la PC control directo del HPI del C542, de la señal de re-inicialización (reset) del DSP y la configuración de la tarjeta para operar con diferentes configuraciones de puertos paralelos (esto es, puertos de impresora de cuatro u ocho bits).

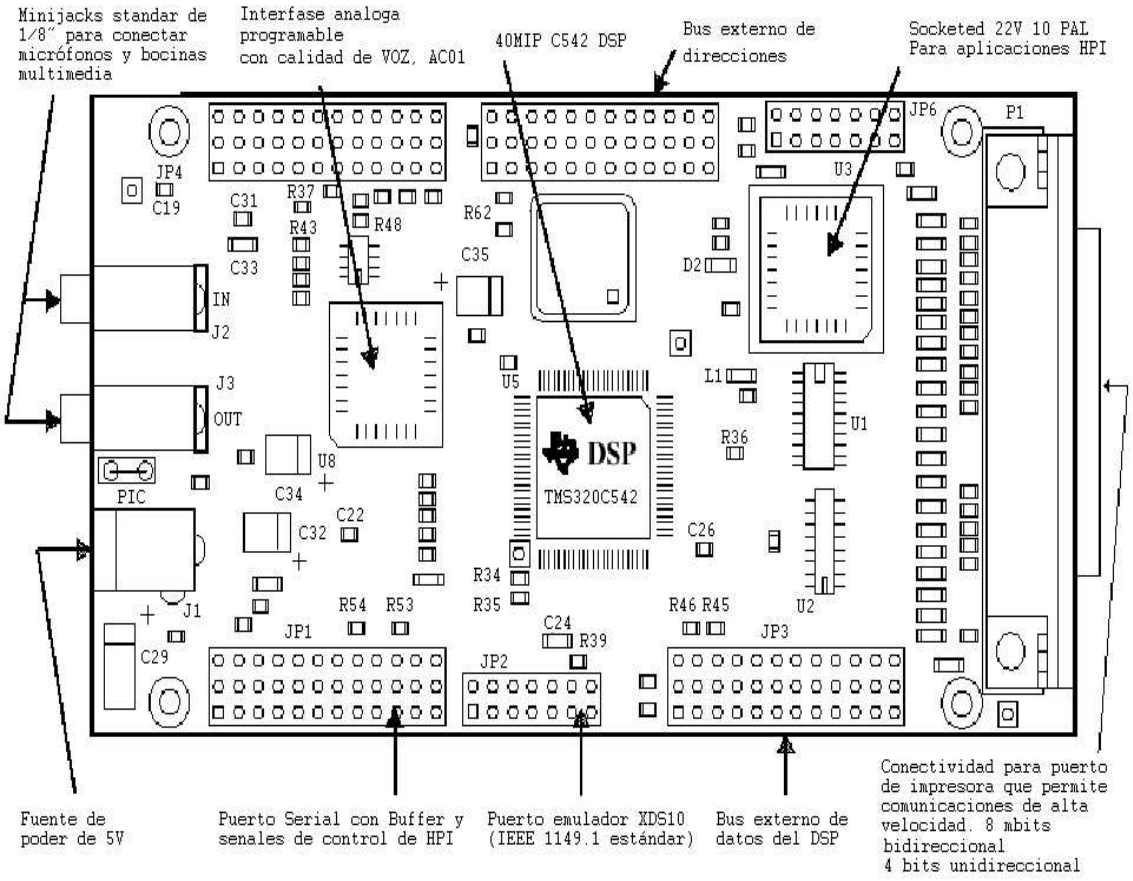


Figura 4.2: Tarjeta del DSK plus.

Cuando se energiza la tarjeta y se corre el depurador, este último inicializa la lógica para el HPI y configura el puerto paralelo para que trabaje en el modo correcto, ya sea 4 u 8 bits. En ese momento, el enlace de comunicación entre la tarjeta DSK plus y la PC está listo para operar.

Para que el DSK plus y el anfitrión se comuniquen adecuadamente, el DSP debe seguir un protocolo de comunicaciones común definido por el anfitrión. Por eso la PC carga el protocolo al software de comunicación del DSP que reside en la memoria del DSP

en las direcciones 80h-17Fh (ver figura 4.3). El protocolo también usa un buffer de comunicación común a ambos y se encuentra en la memoria del DSP en las direcciones 1000h-1009h. El buffer de comunicación es la memoria usada por la comunicación entre el anfitrión y el DSP. Todas las localidades desde 80h a 17Fh y de 1000h a 1009h están reservadas y no deben ser sobrescritas.

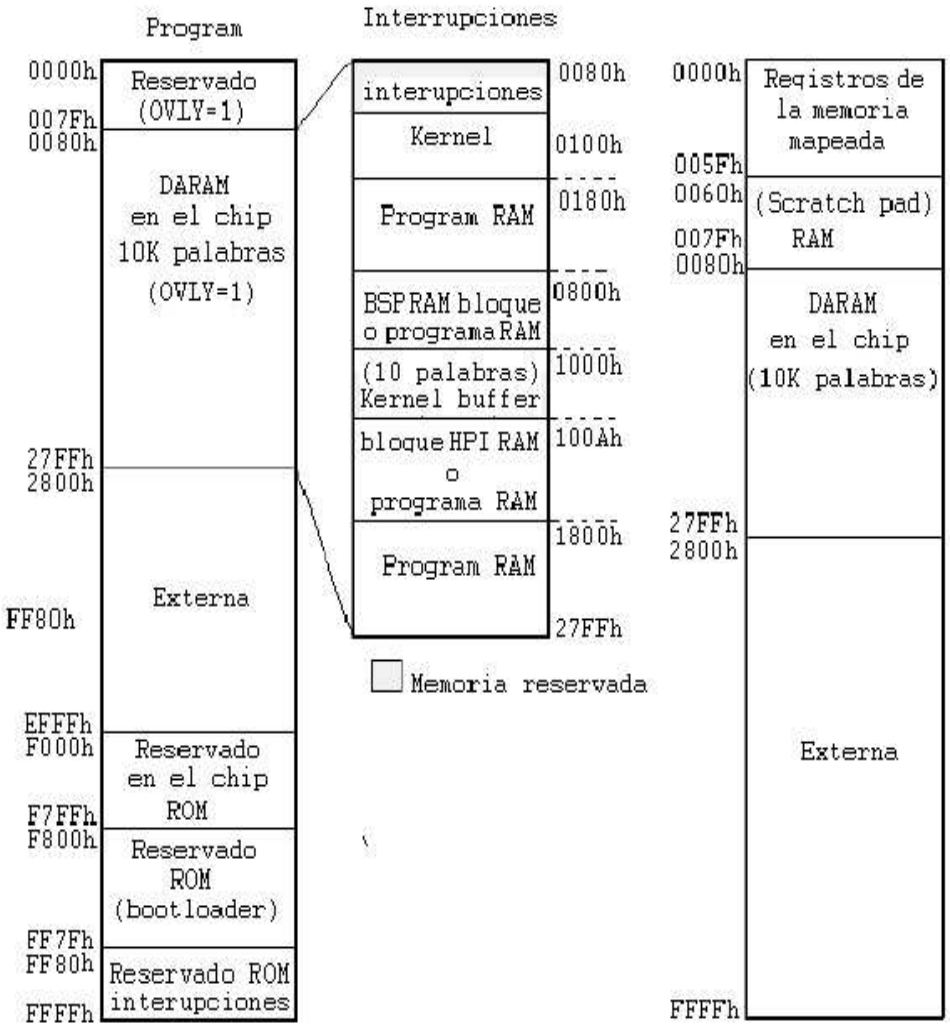


Figura 4.3: Mapa de Memoria del C542.

- El circuito de interface analógica AC01 proporciona un canal simple de adquisición de datos diseñado para preservar la calidad de voz. El AC01 interactá directamente con el puerto serial TDM del C542. El AC01 genera el reloj de requerido (SCLK) y los pulsos de sincronía de trama usados para mandar desde o al AC01. El reloj maestro es generado por el oscilador de la tarjeta.

- El oscilador de la tarjeta es de 10MHz y proporciona una señal de reloj al C542, al AC01 y al dispositivo PAL. La opción PLL del C542 está configurada por un factor de cuatro, creando un oscilador de 40MHz para el reloj interno, por lo que funcionamiento del convertidor AC01 corre a 10MHz.

- El explorador de código GoDSP proporciona una interface de depuración basada en Windows y un ambiente muy manejable. La interface de depuración puede desplegar y modificar todos los registros internos del DSP. Algunas funciones comunes son de depuración paso a paso, colocar puntos de paro en el código, ventanas para observar nuestras variables y el manejo de archivos de entrada/salida.

- Además del depurador, el DSK plus incluye un cargador de aplicaciones y un ensamblador algebraico absoluto. Este último permite programar en lenguaje ensamblador sin tener un conocimiento extenso de los mnemónicos del conjunto de instrucciones.

  Por ejemplo, en el caso de que queramos multiplicar dos números, X e Y, y colocar el resultado en el acumulador B, con el lenguaje ensamblador basado en mnemónicos, debe saberse la función de cada instrucción:

```
STM #Y, AR2        ; Dirección del primer multiplicando
STM #X, AR3        ; Dirección del segundo multiplicando
MPY *AR3, *AR2, B  ; B=X*Y
```

  Con un ensamblador algebraico se usa notación matemática simple:

```
AR2 = #Y
AR3 = #X
B   = *AR2 * *AR3
```

4.3. Conexión de la tarjeta DSK plus

Siga los siguientes pasos para asegurar una conexión correcta entre la tarjeta DSK plus y la PC:

1. Apague la PC.
2. Conecte el cable de la impresora DB25 al puerto paralelo de la PC.
3. Conecte el cable de la impresora DB25 a la tarjeta DSK plus.
4. Conecte el cable de alimentación a la fuente de 5 V_{DC}
5. Inserte el transformador en la toma de alimentación.
6. Conecte el cable adaptador para la fuente al conector de la fuente.
7. Conecte el cable adaptador para la fuente al conector para la alimentación en la tarjeta DSK plus.

4.4. Instalación del software

Haga clic en el Botón de Inicio de la barra de tareas, seleccione Ejecutar y escriba:

```
A:\SETUP.EXE
```

Por omisión el programa de instalación instala el software en el directorio C:\DSK plus. Si usted quiere cambiar el directorio puede hacerlo cuando se le solicita confirmar el directorio destino.

Después de correr el programa de instalación aparecerá un icono llamado Code Explorer. El ensamblador absoluto, el cargador de aplicaciones y la auto-prueba no son aplicaciones para ambiente Windows pero se pueden acceder mediante el shell de Windows.

Para probar la configuración, haga clic en el icono Code Explorer. Aparecerá un cuadro de diálogo para la configuración del puerto donde puede escoger uno de los puertos paralelos disponibles y la dirección de entrada/salida del puerto. El puerto correcto iniciará la interface para depurar. Una vez que se ha instalado correctamente el hardware y el software, la interface de depuración es desplegada en la pantalla como se muestra en la figura 4.1. Si ocurre un error cuando se trata de inicializar el depurador, puede ser debido a un problema de conexiones. Para probar la configuración del hardware, corra el programa de auto-prueba.

4.5. Descripción del depurador Code Explorer

El ambiente DSK plus nos permite experimentar y utilizar el DSP C542 para procesamiento en tiempo real. Además nos da la libertad de crear nuestro propio software para que corra en la tarjeta tal como está, o construir nuevas tarjetas y expandir el sistema de muchas maneras. El depurador del DSK plus trabaja con el ensamblador y el cargador de aplicaciones para ayudarnos a desarrollar, probar y refinar nuestros programas en lenguaje ensamblador.

El depurador Code Explorer consiste: de cuatro ventanas por omisión: desensamblado, registros del CPU, registros de periféricos y la ventana de memoria de datos.

- La ventana de desensamblado muestra el código DSP y la localidad de memoria. La localidad del contador de programa (PC) es realzada por una línea amarilla sobrepuesta en el código.
- La interface también soporta lenguaje simbólico que hace la depuración del código más sencillo, esto es, puede hacerse referencia a localidades y variables en el código ya sea por el nombre de ensamblado o por un etiqueta por lo que no es necesario conocer la dirección física.
- Los puntos de paro (breakpoint) pueden ser agregados o borrados simplemente con apuntar y dar un clic en la instrucción que se quiere analizar. Las propiedades de la ventana de desensamblado pueden cambiarse usando los menús y botones.
- La ventana para los registros del CPU permite visualizar los registros internos y los campos de bit más importantes del DSP. Para cambiar el valor de un registro, apunte y de un clic en el registro y escriba el nuevo valor.
- La ventana para los registros de periféricos es semejante a la ventana de registros del CPU, excepto en que incluye solamente los registros que son usados por los periféricos del DSP, tales como los puertos seriales.
- La ventana de memoria de datos despliega la memoria por omisión. La dirección de inicio y la longitud puede definirse usando un botón. Pueden usarse varias ventanas de memoria de datos al mismo tiempo permitiendo ver cualquier variable, por ejemplo la pila del sistema o las variables a ensamblar. Usando las propiedades de la ventana puede renombrarse para asociarla al nombre de una variable.

- La barra de herramientas en la parte superior de la pantalla incluye botones para depurar paso a paso, correr el programa o reinicializar la tarjeta DSP. El botón de animación permite una representación gráfica de una variable o un buffer. Los datos pueden ser graficados en el dominio del tiempo o en la frecuencia.

- Los puntos de prueba del Code Explorer son usados para conectar archivos en el disco duro a puntos dentro del código de la aplicación. Una vez conectados, estos archivos pueden ser usados como entradas o salidas del código. Para activar un punto de prueba posicione el cursor sobre la instrucción y de un clic al botón derecho del ratón. Para activar/desactivar un punto de prueba seleccione "Toggle probe point". Después de que el punto de prueba es activado, deben ser asignados atributos específicos a la ventana para los puntos de prueba.

- La ayuda en línea está disponible a través de un botón de la interface. Puede ser útil para dar respuestas a preguntas comunes que surjan al utilizar esta herramienta.

4.6. Aplicación para cargar el software

El cargador de aplicaciones, llamado LoadApp, carga nuestra aplicación a la memoria del DSP y comienza a ejecutarla. LoadApp carga el kernel en localidades 0x80h a 0x17Fh y después procede a cargar el código de la aplicación. La forma general del comando para cargar una aplicación es:

```
loadapp  -a  c:\ruta\archivoapp.obj e  etiqueta [opciones]
```

Este comando carga "archivoapp.obj" al DSP y empieza a ejecutar el código en la etiqueta especificada. La etiqueta debe ser una etiqueta válida en los archivos fuentes en ensamblador (.asm).

Las opciones para la línea de comando son:

- -a: Especifica un archivo de aplicación. Inmediatamente seguido por un espacio y la ruta y nombre de archivo.

- -bx: Especifica el modo del puerto: x = 4: modo de 4 bits, x = 8: modo de 8 bits

- -px: Forza el puerto de la impresora a otro puerto donde no este nuestra tarjeta, x puede ser 1,2 o 3.

- -e: Especifica la dirección inicial de ejecución en el DSP. La opción es seguida por un espacio y una etiqueta válida o una dirección. Si se usa una dirección puede ser en formato 0x00 o en 0h. Las etiquetas son sensibles a las mayúsculas o minúsculas.
- -k: Carga un kernel diferente. El kernel debe ser una sección contigua y no puede exceder 7F6h en longitud. La ruta de acceso a este kernel y el nombre de archivo deben especificarse después de esta opción de línea de comando.
- -?: Lista las opciones en la pantalla.

Después que el código ha sido cargado, el sistema sale de LoadApp y restablece el prompt de DOS. El cargador regresa al dispositivo PAL a un estado no inicializado, así que debemos asegurarnos de volver a inicializarlo si tenemos una aplicación basada en la interacción de la PC y la tarjeta DSK plus.

4.7. Ejemplos

A continuación se ilustran algunos ejemplos básicos para que el usuario se familiarice con la arquitectura del C54x y sus instrucciones. Cabe mencionar que todos los programas escritos están en lenguaje algebraico..

4.7.1. Suma utilizando direccionamiento inmediato

```
*     Suma cinco constantes en modo inmediato
      .title   "Suma cinco constantes" ; Título
      .mmregs                    ; Reconoce abreviaturas de
                                 ; registros mapeados y nombres de bits
      .width   80                ; Ancho de la página editada
      .length  55                ; Número de líneas de la página editada
      .setsect ".text",0x1800,0  ; Ubica en memoria programa el código
      .setsect ".data",0x0200,1  ; Ubica en memoria dato inicio de datos

        .sect ".data"            ; datos
D1  .set 1
D2  .set 2
D3  .set 3
D4  .set 4
D5  .set 5
total   .word 0

        .sect".text"             ; Código
        A = #0
        A = A+#D1
        A = A+#D2
        A = A+#D3
        A = A+#D4
        A = A+#D5
        @total = A               ; Salva el resultado de la suma
                                 ; en la localidad  total
FIN    NOP
       GOTO FIN
       .end
```

4.7.2. Suma utilizando direccionamiento indirecto

```
*       Suma cinco números en modo indirecto
*       Salva el resultado en modo directo

        .mmregs
        .width   80
        .length  55
        .setsect ".text",0x1800,0
        .setsect ".data",0x0200,1

        .sect ".data"               ; Datos
D1  .word 1
D2  .word 2
D3  .word 3
D4  .word 4
D5  .word 5
total   .word 0

        .sect".text"                ; Código

        DP = #D1
        A = #0
        AR0 = #D1
        repeat(#4)
          A = A + *AR0+
        @total = A                  ; La suma se almacena en loc. total

FIN     NOP
        GOTO FIN
        .end
```

4.7.3. Suma de varios números utilizando buffer circular

```
*  Suma de 5 números utilizando instrucción repeat(#num)
*  y direccionamiento indirecto utilizando buffer circular en el C54x

        .mmregs
        .width   80
        .length  55
        .setsect ".text",0x1800,0
        .setsect ".data",0x0200,1

        .sect ".data"
x       .word 1,2,3,4,5,5,4,3,2,1
y       .word 0    ;resultado

N1      .set 10    ;tamaño del buffer circular
N2      .set 9     ;diez sumas con buffer circular
        .sect".text"

        A = #0          ; inicialización de A a cero
        BK = #N1        ; inicialización del tamaño del buffer circular
        AR1 = #x        ; dirección donde comienza el buffer circular
        repeat(#N2)     ; repetición de la siguiente instrucción N2 +1
               ; veces
            A = A + *AR1+%   ; AR1 en uso incrementándose en 1 en
                             ; forma de buffer circular
        @y = A          ; se asigna el valor de A en la dirección
                        ; que apunta y

FIN     NOP
        GOTO FIN
        .end
```

4.7.4. Multiplicación de dos vectores

```
*       Multiplicación de dos vectores con suma de
*       productos utilizando dos registros auxiliares
*       C54x

        .width   80
        .length  55
        .mmregs
        .setsect ".text",0x1800,0
        .setsect ".data",0x0200,1

        .sect ".data"          ; datos
x       .word 1,2,3,4,5
y       .word 1,2,3,4,5
y1      .word 0
N2      .set 4

        .sect ".text"           ; Código

        A = #0                  ; inicialización de A a cero
        B = #0
        AR2 = #x                ; AR2 apunta al valor inicial de x
        AR3 = #y                ; Ar3 apunta al valor inicial de y
        repeat(#N2)             ; repetición siguiente instrucción N2 +1
                                ; veces
          A = A + *AR2+ * *AR3+ ; suma de productos

        *(y1) = A               ; Salva el valor de AL en la dirección
                                ; que apunta 'y'
FIN     NOP
        GOTO FIN
        .end
```

4.7.5. Multiplicación de una matriz por un vector

```
        .title   "Multiplicación de Matriz por vector"
        .mmregs
        .width   80
        .length  55
*
*      Ubica segmentos:  Código, datos y vectores de interrupción
*
     .setsect ".text",0x1800,0   ; Sección de código, mapa 0
     .setsect ".data",0x0200,1   ; Sección de datos, mapa 1
     .setsect "vectors",0x0180,0 ; Vectores de Interrupción
*
     .sect ".data"             ;  Sección de Datos y variables
XA      .word    1,2,3,4,5,6,7,8,9    ; Matriz A
V       .word    1,2,1                ; Vector V
Y       .word    0,0,0,0,0,0,0,0,0,0  ; Matriz C=A.V
N1      .set     2
N       .set     2
*
        .sect".text"
        dp = #XA   ; PAGINA DE DATOS DE SUMA

        AR1=#XA    ; AR1 apunta a vector x
        AR2=#Y     ; AR2 salida de datos AxV

        brc = #N1
        blockrepeat(FIN_B)     ; for(i=1;i<2;i++)
           a = #0              ;   acc A = 0
           repeat(#N)          ;   for(j=1;i<2;j++)
             macp(*AR1+,V,A)   ;      A+=A(i,j)*V(i)
          *AR2+ = A            ;   C(i,j)=A(i,j)
FIN_B   nop

FIN     nop
        goto  FIN
        .end
```

4.7.6. Multiplicación de una matriz por una matriz

```
        .title   "Multiplicación de Matrices"
        .mmregs
        .width   90
        .length  55
*
*      Ubica segmentos:  Código, datos y vectores de interrupción
*
        .setsect ".text",0x1800,0
        .setsect ".data",0x0200,1
        .setsect "vectors",0x0180,0
*
*       Sección de Datos y variables
*
        .sect ".data"

X       .word 1,2,3,4,5   ; Matriz X(5x5), puede ser de NxN
X1      .word 6,7,8,9,10
X2      .word 1,2,3,4,5
X3      .word 1,2,3,2,1
X4      .word 2,2,2,2,2
V       .word 1,1,1,1,1   ; Matriz V(5x5), puede ser de NxN
V1      .word 2,2,2,2,2
V2      .word 3,3,3,3,3
V3      .word 4,4,4,4,4
V4      .word 5,5,5,5,5
Y       .word 0,0,0,0,0,0,0,0,0,0,0 ; Matriz Y=AxV
Y1      .word 0,0,0,0,0,0,0,0,0,0,0 ; Matriz Y=AxV
Y2      .word 0,0,0,0,0             ; Matriz Y=AxV

TEM     .word 0          ; Salva AR1
CON     .word 0          ; Contador
*          Aquí se puede cambiar el orden de las matrices NxN
CONTOT  .word 5          ; Orden N
N5      .set  5          ; Orden N
N1      .set  4          ; Orden N-1
```

```
*
        .sect".text"
         dp = #TEM       ; Página de datos
         sp = #0FFAh     ; Ubica el Stack Pointer
         AR1 = #X        ; AR1 apunta a vector
         AR2 = #V
         AR3 = #Y
         A = MMR(AR1)    ; A = contenido de lo que apunta AR1
         @TEM = A        ; Salva dir. inicio de AR1

MISMAL
        CALL MULVV
        B = @CON
        B = B +  #1
        @CON = B
        B = B - #N5         ; Ve cuantos vectores ha multiplicado
        if (BEQ) goto OTRAL ; Va a otra linea de X, V regresa
         A = @TEM           ; Lo contrario
         MMR(AR1) = A       ; AR1 regresa a inicio línea
         goto MISMAL
OTRAL   AR2 = #V            ; AR2 regresa a inicio matriz V
        B =#0
        @CON = B            ; Reincia contador Vec x Vec = 0
        B = @CONTOT
        B = B - #1
        @CONTOT = B
        if (BEQ) goto FIN   ;--->
        A = @TEM
        A = A + #N5         ; Actualiza TEMP para sig. línea
        @TEM = A
        goto MISMAL         ; ---->

FIN     NOP
        goto FIN
**================================================
MULVV   NOP     ; subrutina Vector x Vector
        AR5 = #N1       ; N-1
```

```
        A = #0
SIGUE   T = *AR1+
        A = A + T**AR2+
        if (*AR5- != 0) goto SIGUE
        *AR3+ = A
        RETURN
        .end
```

4.7.7. Ordenamiento de un vector en forma descendente

```
*     Ordena 25 números: en orden descendente
*     Por el método de la burbuja
*
   .title   "Ordena números de mayor a menor"
        .mmregs
        .width   90
        .length  55
*
*   Ubica segmentos: Código, datos y vectores de interrupción
*
        .setsect ".text",0x1800,0
        .setsect ".data",0x0200,1
        .setsect "vectors",0x0180,0
*
*       Sección de Datos y variables
*
   .sect ".data"

MA      .word 1,2,3,4,15,6,7,8,9,10
MA1     .word 11,20,13,24,22,19,23,12,17
MA2     .word 21,14,18,25,16
*
*
N25     .set  25              ; N
N24     .set  24              ; N-1
N23     .set  23              ; N-2
*
              .sect".text"

               sp = #0FFAh           ; Ubica el Stack Pointer
               AR3 = #N23            ; Para ciclo externo

CICLO_N        AR1 = #MA             ; AR1 apunta a inicio de datos
               BRC = #23             ; repetición de bloque
               BLOCKREPEAT(FIN_B)    ; for(i=0;i<23;i++)
```

```
                A = *AR1+           ; A = MA(i)
                B = *AR1            ; B = MA(i+1)
                B = B - A           ; MA(i+1) - MA(i)
                if(BGT) goto CAMBIA
*
                goto FIN_B
CAMBIA          B = *AR1-           ; mar(*AR1-)
               *AR1+ = B            ; Intercambia
               *AR1 = A
FIN_B           NOP                 ; fin de for
*
             if (*AR3- != 0) goto CICLO_N   ; Ciclo externo
*
FIN          NOP
             goto FIN
```

4.7.8. División de dos números

Este programa está basado en un algoritmo rápido de obtención de división y raíces cuadradas [23] que funciona en forma similar a un convertidor de aproximaciones sucesivas.

```
*     División = Numerador (32b) / Denominador (16 b)
*     El Numerador y denominador son positivos
*     Numerador > Denominador
*
        .title   "División"
        .mmregs
        .width   90
        .length  55
*
*      Ubica segmentos:  Código, datos y vectores de interrupción
*
        .setsect ".text",0x1800,0
        .setsect ".data",0x0200,1
        .setsect "vectors",0x0180,0
*
*       Sección de Datos y variables
*
        .sect ".data"
NUMH    .word 0064h         ; Numerador H
NUML    .word 0AB90h        ; Numerador L
DEN     .word 055Ch         ; Denominador (q8)
TEM     .word 8000h         ; Bit de prueba
COC     .word 0             ; Cociente de división
MASK    .word 07FFFh        ; Máscara para operación AND
MASKH   .word 07FFFh        ; Parte alta de la máscara
N       .set  15            ; N-1 pruebas

        .sect".text"

        SXM = #0            ; Suprime extensión de signo
        DP = #TEM           ; Página de datos
        SP = #0FFAh         ; Ubica el Stack Pointer
```

```
         BRC = #N             ; Para repetición de bloque

        blockrepeat(FIN_B)
           A = @TEM
           A = A + @COC       ; Pone bit de prueba
           @COC = A           ; COC = COC + TEM
           T  = A
           B = T*uns(@DEN)    ; B = COC*DEN (no signado)
           A = @NUMH << 16
           A = A + @NUML
           B = B - A          ; B = C*DEN - NUM
           if (BLT) goto MUEVE_M ;--->
           A = @COC
           A = A & @MASK      ; Borra bit de prueba
           @COC = A
MUEVE_M    A = @MASKH << 16
           A = A + @MASK
           A = A >> 1
           @MASK = A
           A = @TEM >> 1
FIN_B    @TEM = A

FIN        NOP
           goto FIN
           .end
```

Resumen

Con base a la arquitectura del C54x expuesta anteriormente, el lector ya está en capacidad de entender los siguientes programas básicos de aplicación y generar sus propias aplicaciones. En los siguientes ejemplos se trabaja con el ensamblador en línea para el ambiente Code Explorer del TMS320C54x de TI. Los programas desarrollados se han realizado en lenguaje algebraico combinado con instrucciones de ensamblador. Estos ejemplos pueden ser útiles como una breve introducción a la programación y utilización del C54x.

Capítulo 5

DSK DSP TMS320C5402

En este capítulo se muestra una introducción a la tarjeta de desarrollo y evaluación del DSP TMS320C5402, describiendo de forma resumida, los dispositivos y componentes que la integran. Además, se incluye una introducción al ambiente de desarrollo Code Composer Studio y ejemplos de programación tanto en lenguaje ensamblador como en lenguaje C.

5.1. Tarjeta de desarrollo y evaluación DSKC5402

La tarjeta de desarrollo y evaluación (DSK) permite que los usuarios puedan probar y evaluar aplicaciones en tiempo real del dispositivo TMS320C5402, mediante una interface entre una terminal huésped (PC) y la tarjeta DSK por medio de un ambiente de desarrollo (Code Composer). Dicha plataforma es útil para la evaluación de arquitecturas C54x y C54xx, así como desarrollos de programas y depuración de éstos. La tarjeta DSK para el C5402 consta de los siguientes componentes en hardware para el desarrollo de aplicaciones:

- Un DSP TMS320VC5402 a 100 MHz.
- Un controlador de bus de prueba JTAG TBC SN74ACT8990, con el estándar IEEE 1149.1, para la emulación e interface con el puerto huésped conectado a la PC por medio de un puerto paralelo (HPI).
- Dos convertidores A/D - D/A TLC30AD50 con interface a micrófono y altavoz.
- Una interface de puerto telefónico.
- Una interface de datos asíncronos RS - 232.

- Memoria SARAM Externa (64k x 16 bits).
- Memoria Flash Externa (256k x 16 bits).
- Una interface para expansión con tarjetas "daughter board".

La plataforma DSK es una tarjeta que incluye un regulador de voltaje lineal que provee 1.8 V_{DC} para la alimentación principal del DSP, 3.3 V_{DC} para dispositivos digitales y 5 V_{DC} para dispositivos analógicos. Además, el DSK consta de una barra de interruptores de 8 posiciones para opciones externas de los usuarios, un botón para reinicializar manualmente al DSP y cuatro leds indicadores; donde uno es para indicar la alimentación de la tarjeta y los tres restantes pueden ser controlados por el usuario.

La interface entre las memorias externas SARAM, Flash y el conector de expansión de memoria es a través de una interface de memoria externa de 16 bits (EMIF). El DSK consta de dos conectores de expansión para el acceso a una tarjeta tipo "daughter board", con las siguientes características:

- Extiende las capacidades del DSK.
- Provee al usuario aplicaciones específicas para interfaces de entrada y salida.
- Un conector de expansión que está dedicado para accesar a la interface de memoria externa asíncrona del DSP (EMIF), mientras que el otro, para los periféricos del DSP y el acceso directo a los pines de señalización de estado y control del DSP.

Mediante una circuitería analógica con interface a un convertidor análogo - digital AD50 se puede tener acceso a un puerto telefónico. La conexión con el puerto telefónico y el DSP es través del primer puerto serial multicanal bufereado multicanal (McBSP0) que integra el DSP. La interface al micrófono y altavoz es por medio del segundo convertidor AD50 que está conectado al DSP con el segundo puerto serial bufereado multicanal (McBSP1).

La figura 5.1 muestra un diagrama de bloques de la tarjeta DSK, observando que además de contar con los subsistemas en hardware anteriormente mencionados, se integran dispositivos lógicos como son; multiplexores, controladores específicos para el puerto paralelo, la UART, etc, así como un dispositivo lógico programable (CPLD) encargado de la señalización necesaria para la interface de la terminal huésped y el DSP. A lo largo de esta sección se describirán las características principales de estos dispositivos.

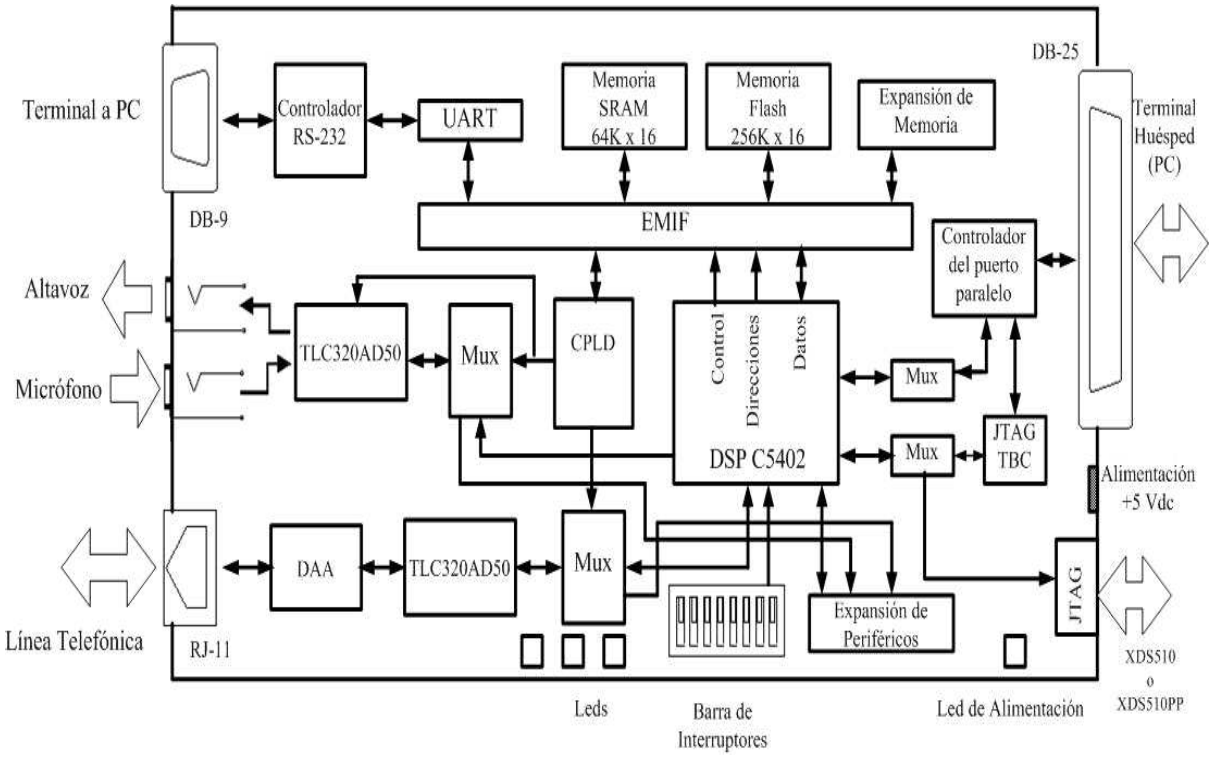


Figura 5.1: Diagrama de Bloques de la tarjeta DSK.

5.1.1. Descripción General de la Interfaz DSK

A continuación se dará una breve descripción de los componentes que integran a la plataforma DSK, así como del mapa de memoria y modos de inicialización, considerando sólo las características más significativas de éstos.

DSP TMS320C5402

El DSK soporta un DSP TMS320VC5402 el cuál opera a 100 MHz con un voltaje de alimentación a 1.8 V_{DC} y voltajes a 3.3 V_{DC} para pines de entradas y salidas, además de proveer todas las interfaces y señalizaciones de control necesarias que se citan a continuación:

- Interface de emulación JTAG ("Joint Test Action Group").
- Interface de control para reinicialización e interrupciones externas.

- Interface de puerto huésped (HPI) que proporciona transferencia bi-direccional entre la PC y el DSP.
- Diferentes modos de inicialización que permiten al usuario seleccionar diferentes métodos de arranque desde la memoria ROM y el puerto HPI, dependiendo del ambiente de ejecución.
- Interface de expansión de memoria externa para la conexión de memorias externas del tipo SARAM y FLASH.
- Interface de Temporizador.
- Interface entre circuitos telefónicos (DAA) y el puerto McBP0.
- Interface a micrófono y altavoz entre el puerto McBP1.

Interface de Puerto Paralelo

La comunicación entre el DSK y la PC es por medio de una interface de puerto paralelo de 8 bits (HPI), que se implanta usando un microsistema estándar de interface de puerto paralelo periférico PPC34C60, con las capacidades de:

- Interface con el puerto HPI del DSP.
- Proveer controles de reinicialización.
- Acceso con algunos registros internos del dispositivo lógico programable complejo (CPLD).
- Control del bus de prueba TBC.
- Programación del CPLD.
- Transferencia secuencial, con auto-incremento, o transferencia de acceso aleatorio.
- Interrupción de huésped y capacidad para interrupciones del C5402.
- Acceso a toda la memoria RAM interna a través del bus de DMA.
- Capacidad para continuar un proceso de transferencia después de una pausa emulada.

Esta interface, por medio de un conector DB-25, se conecta al puerto paralelo de la PC, siendo compatible con el estándar IEEE-1284 y los puertos tipo EPP ("Enhanced Parallel Port") y ECP (" Extended Capabilities Port").

Modos de Inicialización

El DSK puede ser inicializado por medio de:

- La memoria FLASH externa.
- Tarjetas de Expansión de memoria tipo ROM.
- Controladores de bus de prueba JTAG TBC.
- La interfaz JTAG XDS510 externa.
- Una Interface de puerto paralelo huésped via HPI.

Dispositivo lógico programable

Para realizar la interface entre la plataforma y el software de desarrollo en cuanto al control y estado lógico se integra un dispositivo lógico programable complejo CPLD Cypress CY37128, el cual realiza las siguientes funciones:

- Control de reset.
- Registros de estado y control mapeados en memoria del DSP.
- Decodificación de periféricos (registros de control e interface UART).
- Control de leds indicadores para usuarios y estado de la barra de interruptores.
- Control de transmisión de datos.
- Opciones de usuarios.

El CPLD CY37128V-125 trabaja con un voltaje de alimentación de 3.3 V, y una frecuencia máxima de reloj de 125 MHz. El código fuente del CPLD está escrito en lenguaje VHDL, con el inconveniente de no poder accesar a este código y no incluirse en la documentación de la tarjeta DSK.

En la figura 5.2 se muestra el diagrama de bloques de las funciones que realiza el CPLD.

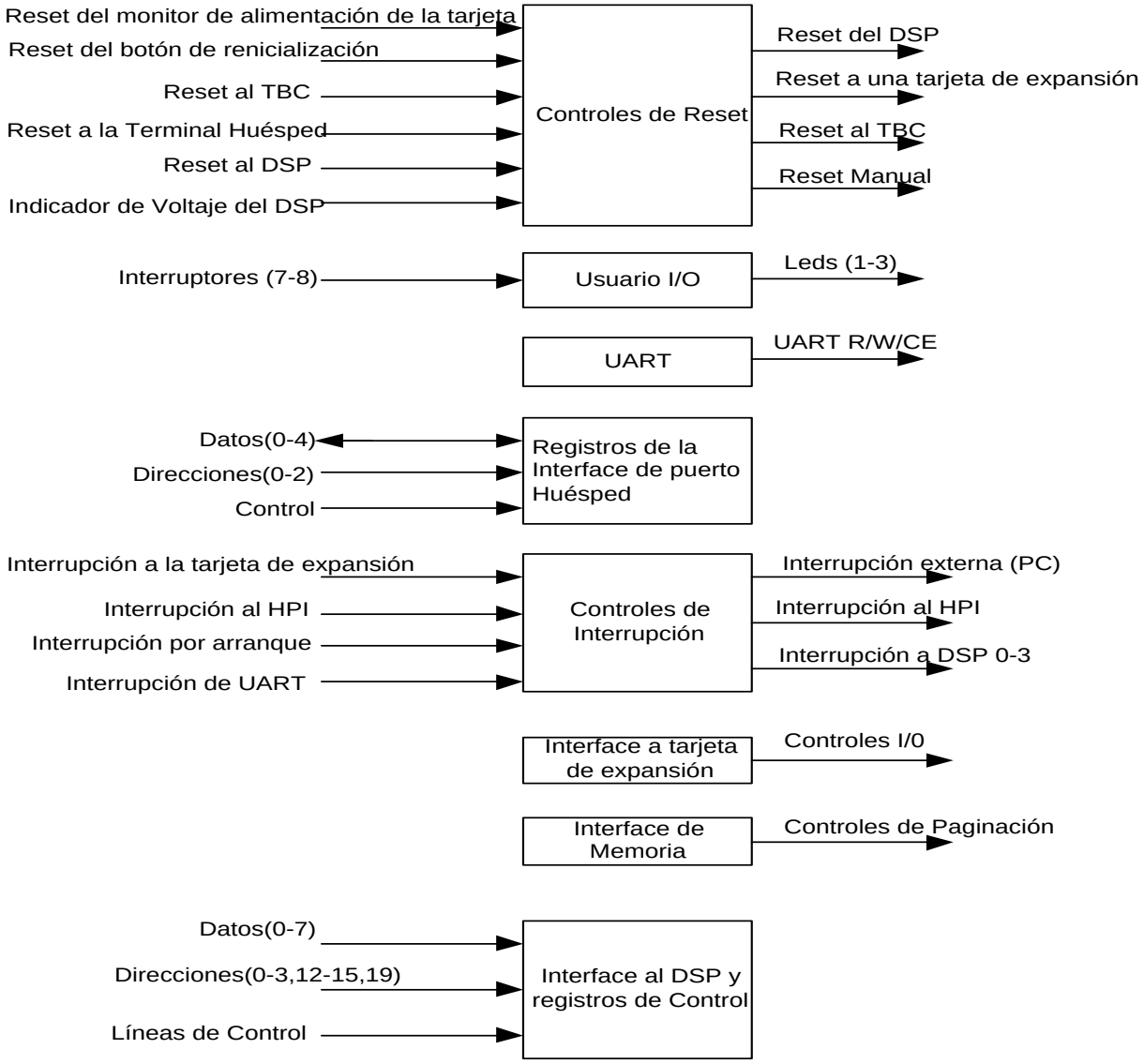


Figura 5.2: Diagrama de Bloques de las funciones que realiza el CPLD.

El CPLD trabaja en conjunto con un supervisor de voltaje TI TP3707-33D para proveer distintos tipos de señales de reset a la tarjeta DSK. La tarjeta DSK soporta varias formas de reinicialización que pueden ser desde el DSP, el JTAG TBC, o bien, por medio de una tarjeta de expansión. Por tanto, podemos enumerar cuatro distintas fuentes de reinicialización:

1. Reset para encender y por falta de voltaje desde el supervisor de voltaje.
2. Reinicialización manual por medio del botón de reset.
3. Reset proveniente de una tarjeta de expansión.
4. Reinicialización desde la terminal huésped (PC), el controlador de prueba TBC o el DSP.

El CPLD incluye siete registros de control y estado que están mapeados en memoria en el espacio I/O del DSP (Ver figuras 5.2 y 5.4). Estos registros son de 8 bits y se encuentran mapeados en los bits menos significativos del bus de datos EMIF (ED[7..0]). Los registros mapeados en memoria del CPLD pueden configurarse para realizar las siguientes acciones:

- Control por el usuario de los tres leds indicadores.
- Control y monitoreo de las tarjeta de expansión.
- Habilitación de la interrupción NMI desde el ambiente de desarrollo.
- Selección de las conexiones de los puertos McBSP0 y McBSP1.
- Determina el estado de las interrupciones.
- Lectura del estado de la barra de interruptores controlados por el usuario.
- Lectura de la selección de la opción de modo de operación del DSP.
- Habilitación o inhibición de las interrupciones producidas por la tarjeta de expansión y la UART.
- Proporciona los mecanismos de semaforización por software de las señalizaciones del ambiente de desarrollo.

En el cuadro 5.1 se presenta la definición y localidades de memoria de los registros del CPLD mapeados en memoria espacio I/O.

El registro de control 1 (CNTL1) controla la señalización de algunas interfaces como se muestra en el cuadro 5.2. Para configurar algunas acciones de las interfaces de puerto telefónico, micrófono y altavoz, así como la selección de las fuentes para los puertos seriales utilizamos el registro CNTL2 (cuadro 5.3). El registro STAT proporciona el estado de algunas acciones en la tarjeta DSK como se muestra en el cuadro 5.4. Los registros SEM0 y SEM1 son

Dirección (hex)	Nombre	Descripción
0	CNTL1	Registro de control 1
1	STAT	Registro de estado
2	DMCNTL	Registro de control de memoria dato
3	DBIO	Registro para la tarjeta Daughter Board de I/O
4	CNTL2	Registro de control 2
5	SEM0	Semáforo 0
6	SEM1	Semáforo 1

Cuadro 5.1: Registros del CPLD mapeados en memoria espacio I/O.

# bit	Nombre	Descripción
7	INT2SEL	Selecciona la fuente para la interrupción INT2 0=boot trap, 1=tarjeta de expansión
6	DB_RST	Reinicializa la tarjeta de expansión
5	DB_INT	Habilita la interrupción de la tarjeta de expansión 0= inhibida, 1=habilitada
4	NMIEN	Habilita la interrupción NMI 0= inhibida, 1=habilitada
3	INT3SEL	Selecciona la fuente para la interrupción INT3 0=desde la terminal huésped, 1=tarjeta de expansión
2	USR LED2	Control del led indicador 2 0=apagado, 1=prendido
1	USR LED1	Control del led indicador 1 0=apagado, 1=prendido
0	USR LED0	Control del led indicador 0 0=apagado, 1=prendido

Cuadro 5.2: Campos de bits del registro de control 1 (CNTL1)

utilizados como semáforos por hardware que son utilizados por el DSP y la terminal huésped (la PC) para tener control de acceso a memoria compartida.

El registro DMCTRL (cuadro 5.5) habilita al software del DSP para controlar el espacio de memoria dato seleccionado entre:

- La Memoria dato externa de la tarjeta DSK y la tarjeta de expansión.

# bit	Nombre	Descripción
7	DAAOH	Control de colgado para la interface DAA 0=colgado, 1=descolgado
6	DAAID	Habilitación para la identificación de llamada (caller ID) 0=deshabilitado, 1=habilitado
5	FLASHENB	Selecciona el tipo de memoria externa a usar 0=memoria SARAM, 1=memoria FLASH
4	INT1SEL	Selecciona la fuente para la interrupción INT1 0= UART, 1=Tarjeta de expansión
3	FC1CON	Bit de control para la interface Mic/Altavoz AD50 (0)
2	FC0CON	Bit de control para la interface DAA AD50 (0)
1	BPSEL1	Selecciona la fuente para el puerto McBSP1 0=Mic/altavoz, 1=tarjeta de expansión
0	BPSEL0	Selecciona la fuente para el puerto McBSP0 0=DAA, 1=tarjeta de expansión

Cuadro 5.3: Campos de bits del registro de control 2 (CNTL2)

- El Tamaño y modo de acceso a la memoria dato de la tarjeta de expansión y los espacios I/0.
- Las páginas de memoria dato y espacio I/0.

El registro DBIO proporciona cuatro bits (DBIO[3-0]) de propósito general para usarse como entradas/salidas hacia la tarjeta de expansión. Estas cuatro señales pueden ser definidas como entradas o salidas según se configuren en este registro. Por tanto, en los bits DBIO[7-4] se configura la dirección de dichas señales, un cero en estos bits configura a la señal como entrada, mientras que al escribir un uno, se configura como salida.

Interface UART

El DSK dispone de una interface UART que conecta los espacios de entrada y salida empezando en la dirección 4000h. La interface es implantada por un estándar industrial TL16C550 a través de un driver convertidor de nivel RS-232 MAX328 a un conector hembra BD-9 que usa tres líneas de direcciones del DSP (A0 a la A2) como decodificador lógico para accesar a ocho registros internos del DSP.

# bit	Nombre	Descripción
7	-	Sin funcionamiento
6	USR SW1	Estado del interruptor para usuario 1 0=apagado, 1=prendido
5	USR SW0	Estado del interruptor para usuario 0 0=apagado, 1=prendido
4	NMI	Estado de la interrupción NMI controlada por la terminal huésped 0= no válido, 1=válido
3	DB_INT0	Estado de la interrupción INT0 controlada por la tarjeta de expansión 0=no interrupción, 1=interrupción.
2	HST_INT3	Estado de la interrupción INT3 (terminal huésped)
1	DB_DET	0=No interrupción, 1=interrupción Detección de la tarjeta de expansión 0=no detectada, 1=detectada e instalada
0	DAARING	Indicación de llamada recibida 0=sin llamada recibida, 1=llamada recibida

Cuadro 5.4: Campos de bits del registro de estado (STAT)

# bit	Nombre	Descripción
7	DMSEL	Selección de memoria dato 0=tarjeta DSK, 1=tarjeta de expansión
6	DBWIDE	Selección del tamaño de la palabra para la memoria dato de la tarjeta de expansión 0=16, 1=32
5	DB32 ODD	Modo de acceso para el direccionamiento a 32 bits de la tarjeta de expansión 0=par, 1=impar
4-0	M_PG	Página para memoria SARAM/FLASH Las páginas para la memoria SARAM están divididas cada 32k, mientras que para la FLASH se dividen en 16k

Cuadro 5.5: Campos de bits del registro de control de memoria dato (DMCTRL)

Interface de micrófono y altavoz

La interface de micrófono y altavoz consta de dos conectores estándares de 3.5 mm que son usados para la interface de audio. Uno para conectar la entrada de audio desde un micrófono y el otro para conectar la salida de audio a un altavoz. La entrada de audio está diseñada para acoplar señales AC incluye un amplificador de hasta 10 dB de ganancia y una etapa para la conversión con interface al convertidor TLC320AD50 que a su vez está conectado al puerto MCBSP1 del DSP.
Se incluye una polarización para el micrófono desde el conector, es decir, la entrada de audio está diseñada para micrófonos electret, por tanto, si se requiere conectar a la entrada un micrófono dinámico se necesitará desacoplar el voltaje de polarización[1]. El nivel de voltaje máximo permitido es de 500 mV (350 mV_{RMS}). La salida de audio puede configurase mediante software de control para elegir ganancias de salida desde +0 a -12 dB con decrementos de 6 dB. En el caso de la entrada de audio, se puede programar ganancias de +0 a 12 dB en incrementos de 6 dB. Adicionalmente, la configuración en hardware de las interfaces soporta para la salida cargas directas o de baja impedancia (8 Ohms) o bien, cargas con impedancias alrededor de los 600 Ohms.

Interface de puerto Telefónico (DAA).

La interface de puerto telefónico (DAA) permite una conección al PSTN ("Public Switched Telephone Network") o a una línea por medio de un conector estándar RJ-11. El circuito DAA está basado en un circuito integrado de interface con acoplamiento óptico CPC5604A. Dicha interface está diseñada bajo los requerimientos de los estándares de EUA, Japón y Europa (CTR21).

Esta Interface tiene las siguientes características:

- Detección de llamada de la señal.
- Detección de "Caller ID".
- Función de "Hookswitch".
- Respuesta en frecuencia de 30 a 4000 Hz.
- Transmisión de datos en intervalos de 50 kbits o mayores.

[1]Por ejemplo, con un capacitador conectado en serie para el desacoplamiento podría ser una solución práctica.

El uso de esta interface tiene las siguientes limitaciones:

- El circuito no detecta lazos de corrientes o de voltajes.
- El limite de ruido permisible en el CPC5604A limitará el intervalo de transmisión de datos a 53 kHz.

Cabe mencionar que la interface de DAA en el DSK puede ser conectado a redes telefónicas comerciales.

Mapa de Memoria

El mapa de memoria programa y dato compatible con la tarjeta de desarrollo y el ambiente CCS se muestran en las figuras 5.3 y 5.4, respectivamente, como también el mapa de memoria de los espacios de puertos de entrada - salida (figura 5.4). Para mayores referencias de las interfaces mencionadas en [35].

5.2. Desarrollo de aplicaciones usando el Code Composer studio (CCS)

El ambiente de desarrollo Code Composer Studio (CCS) agiliza y optimiza los procesos de desarrollo para programadores quienes diseñan e implantan aplicaciones incrustadas para el procesamiento digital de señales en tiempo real. El CCS extiende sus capacidades en un entorno gráfico al incluir herramientas para el análisis en tiempo real en conjunto con una tarjeta de desarrollo para un DSP (DSK). En esta sección se dará una breve explicación de la configuración, manejo y de las herramientas que constituyen al ambiente de desarrollo CCS.

5.2.1. Generalidades del CCS

El CCS es una aplicación de Texas Instruments que permite a los usuarios programar, editar, depurar y analizar programas para su implantación en una arquitectura DSP. El CCS soporta diversos DSP's de Texas Instruments para la creación de proyectos donde el CCS consta de un gestor de proyectos que permite manejar tareas de programación. Para propósitos de depuración consta de puntos de interrupción ("breakpoints"), observadores de variables ("watches"), ventanas para la visualización del mapa de memoria, de los registros y de la pila. Además, tiene opciones para configurar puntos de prueba para el flujo de datos

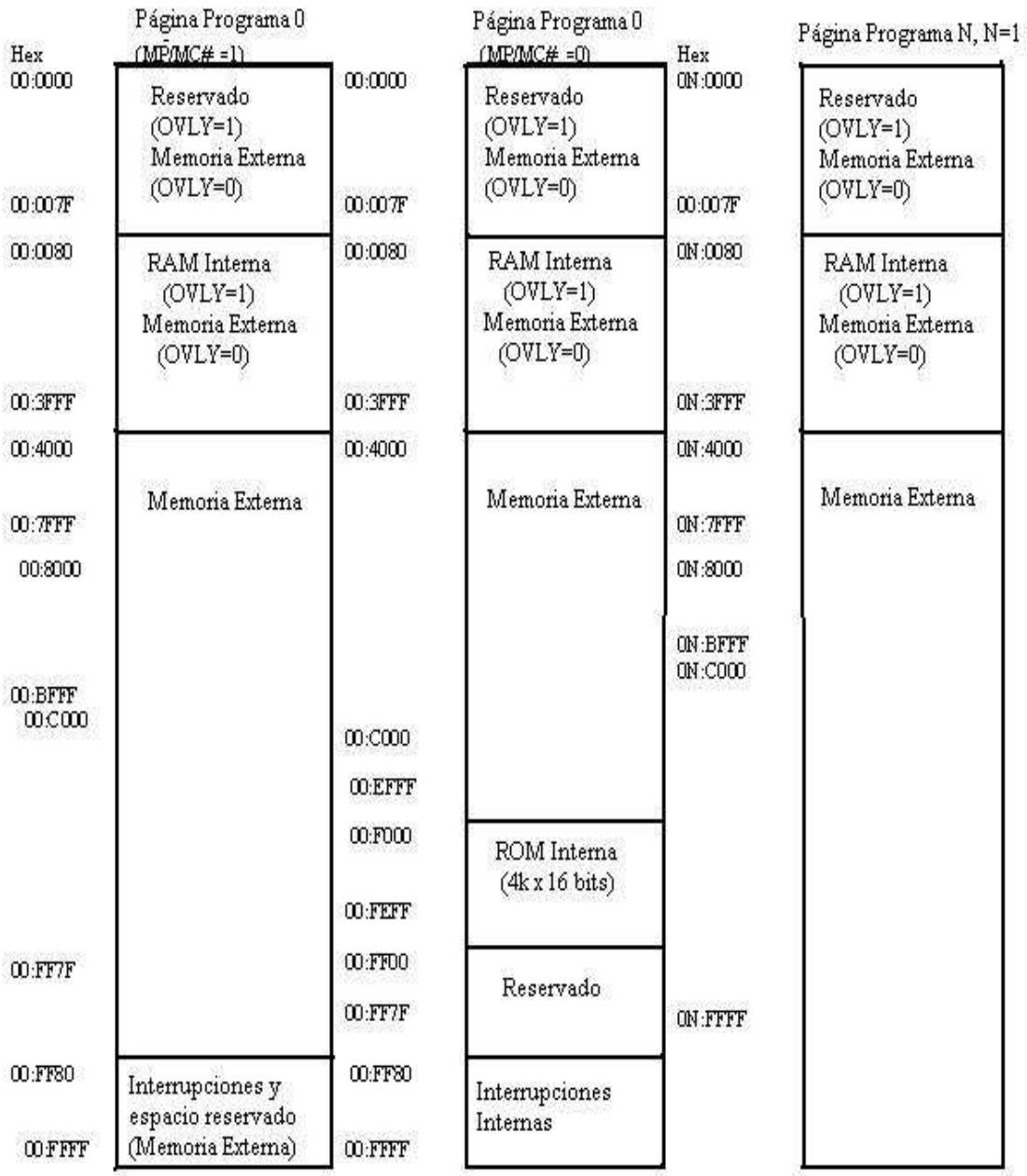


Figura 5.3: Mapa de Memoria Programa.

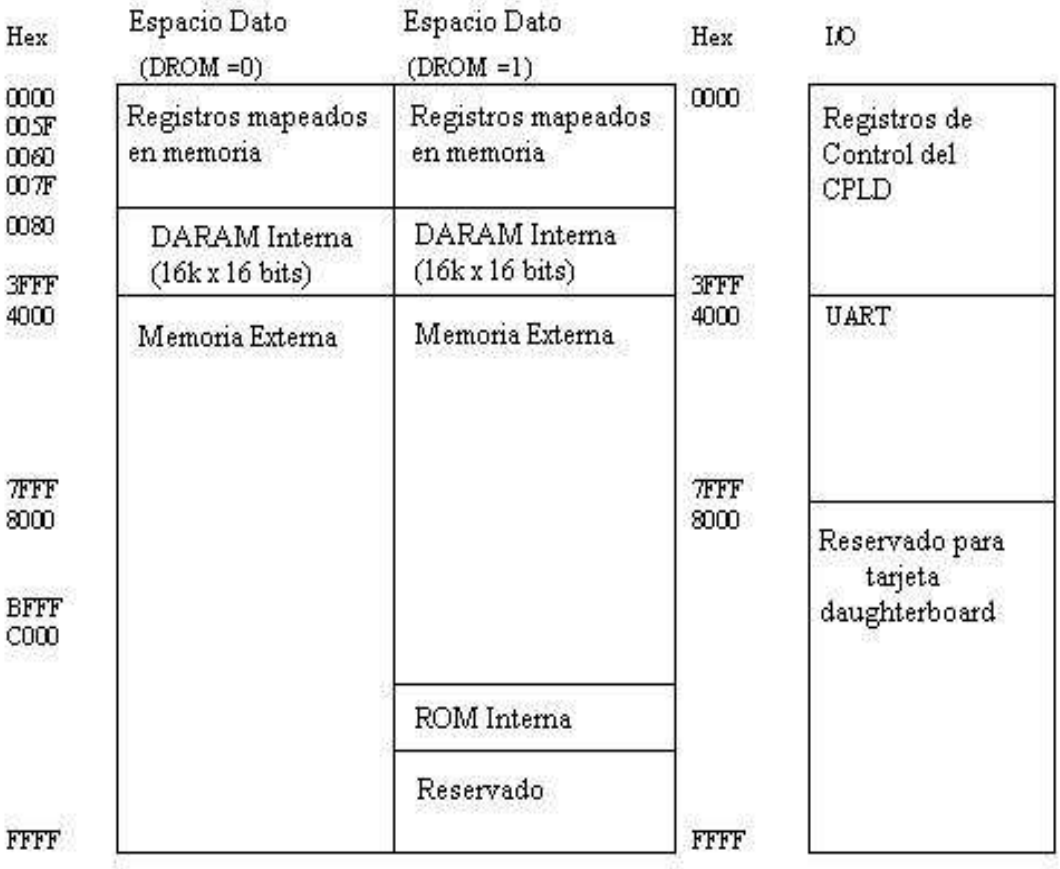


Figura 5.4: Mapa de Memoria Dato y espacio I/O.

hacia y desde la tarjeta de evaluación (DSK), análisis gráfico, ejecución personalizada y capacidad para la programación y visualización de código mixto en lenguaje ensamblador y en lenguaje C. Una opción muy importante y útil es poder manejar grandes proyectos desde un ambiente gráfico para el usuario. Lo anterior, se puede resumir en lo siguiente:

- Entorno integrado de desarrollo con editor, depurador, gestor de proyectos, profiler, puntos de interrupción, observadores de variables, puntos de prueba, ventanas para la visualización de registros/mapa de memoria/pila.
- Herramientas de generación de código: Compilador de C, optimizador de lenguaje ensamblador y ligador de archivos y de código fuente.

- Funciones para la realización de sistemas operativos en tiempo real (DSP/BIOS)
- Intercambio de datos en tiempo real (RTDX) entre una terminal (PC) y el DSP.
- Herramientas de análisis y visualización de datos en tiempo real.

5.2.2. Herramientas para la Generación de Código

Las herramientas para la generación de código permiten configurar los programas mediante un proceso de ensamblado, compilación, ligado y referencias a librerías a usar como se muestra en la figura 5.5. Dicho proceso puede realizarse de manera transparente al usuario gracias a las opciones de compilación del proyecto del ambiente CCS, o bien, efectuarlo de modo manual. A continuación se da una breve descripción de las herramientas para la generación de código:

- El compilador de lenguaje C acepta código fuente en C produciendo código fuente en lenguaje ensamblador nativo del DSP.
- El programa ensamblador traslada archivos o códigos fuente escritos en lenguaje ensamblador a archivos objeto de lenguaje máquina, donde el lenguaje máquina está basado en el estándar de formato de archivos de objetos comunes (COFF).
- El ligador combina archivos objetos en un módulo único de objetos ejecutables. Al crear este módulo ejecutable se realiza una relocalización e inserción de las referencias externas al proyecto. El ligador de archivos permite conjuntar los diferentes archivos del proyecto en un archivo único en modo de librería.
- La utilidad de traducción de mnemónicos de lenguaje ensamblador a modo algebraico convierte el código escrito en lenguaje ensamblador a un código descrito de forma algebraica (instrucciones algebraicas) y viceversa.
- El generador de librerías podemos utilizarlo para el diseño de librerías propias del usuario.
- La utilidad de conversión a sistema hexadecimal convierte archivos del tipo COFF a archivos con formato objeto de TI, ASCII-HEX, Intel, Motorola y Tektronic. Por tanto, se puede utilizar dichos archivos con un programador de memoria EPROM.
- El listado de referencia cruzada usa los archivos objetos para la referencia cruzada de símbolos, sus definiciones, y sus referencias en los archivos fuente ligados.

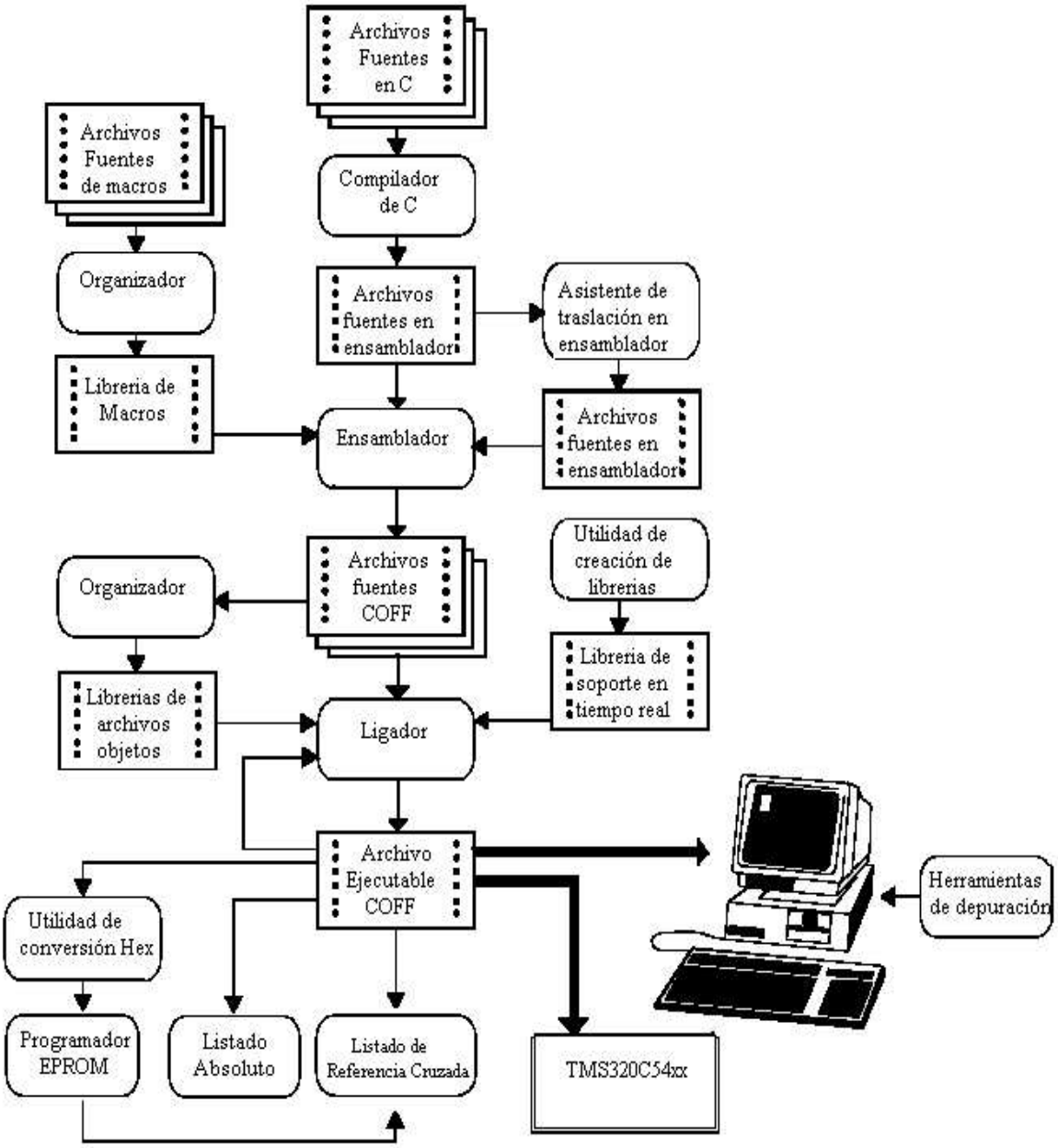


Figura 5.5: Diagrama de Bloques de las herramientas para la generación de código.

- El listado absoluto acepta archivos objeto ligados como archivos de entrada y crea archivos con extensión ".abs" como archivos de salida, de tal manera que al ensamblar los archivos con extensión ".abs" se produce una lista que contiene las direcciones absolutas del código.

5.2.3. Creación de un proyecto mediante el CCS

Para realizar un programa en el CCS lo habitual es la creación de un proyecto en el cual se tendrá que realizar los siguientes pasos para su ejecución:

- Creación del proyecto.
- Creación de los archivos de código fuente.
- Ligar los diversos archivos con códigos fuente con un archivo de ligado de comandos con extensión ".cmd" donde se definen los espacios de memoria permisibles de la tarjeta de evaluación (DSK).
- Especificar la localización de los ficheros "include" y las librerías a utilizar.
- Compilar el proyecto y configurar opciones de compilado tales como indicar conflictos de pipeline, optimizar espacio memoria y visualizar problemas de traslape en memoria de variables, reservar registros auxiliares para su uso específico en archivos en C, etc.
- Cargar el proyecto compilado a la tarjeta DSK.

Para la creación del proyecto se escoge la opción "Project" de la barra de herramientas para después seleccionar el tópico "New", de tal manera que se abrirá un cuadro de dialogo donde se nombrará al proyecto con extensión ".mak".

De igual manera para la creación y edición de los códigos fuente se escoge la opción "File" de la barra de herramientas con la selección de "New" donde se nombrará al código con la extensión debida (".asm", ".C", ".cmd").

La figura 5.6 muestra la ventana principal de ambiente CCS. Es muy importante definir bien el archivo de ligado de comandos (".cmd") dado que en éste se encuentran la definición de los segmentos de memoria a usar

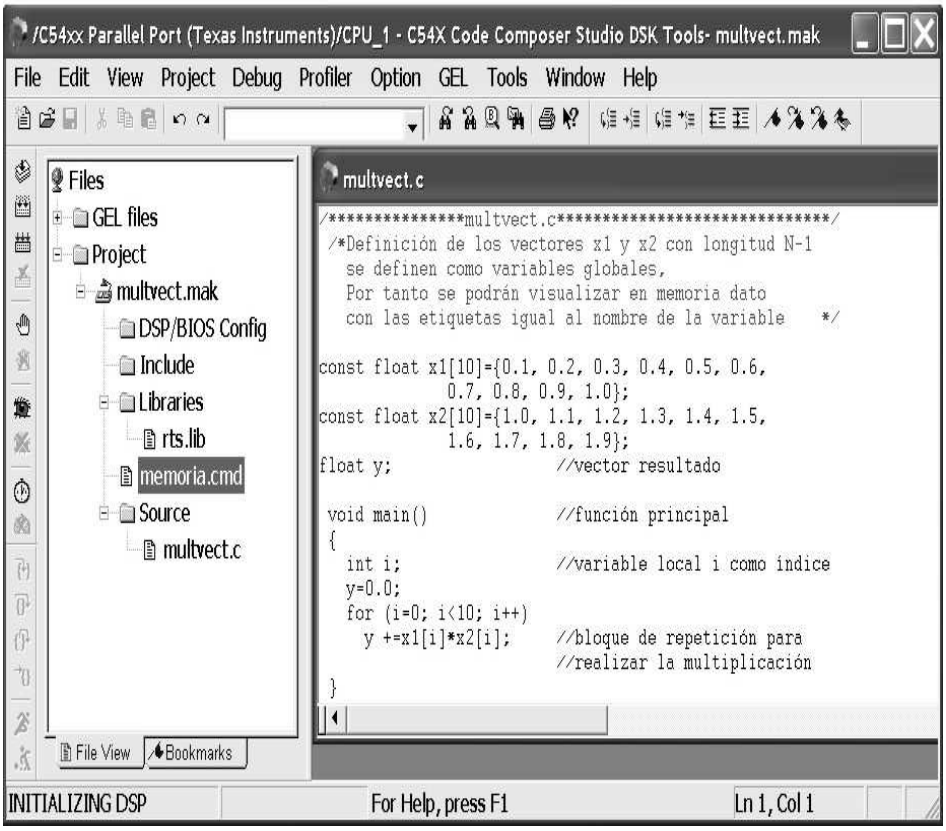


Figura 5.6: Visualización del ambiente CCS.

Por ejemplo:

```
/* Memoria.cmd */

 MEMORY
   {
     PAGE 1:   /* espacio dato     */

        STACK  : origin = 0x1180, length = 0x0560 /* memoria para la pila */
        DATA   : origin = 0x1F00, length = 0x2080 /* código dato          */

     PAGE 0:   /* espacio programa */
```

```
      PROG   : origin = 0x0100, length = 0x1E00 /* código programa      */
             /* vectores de interrupción                               */
      VECS   : origin = 0x0080, length = 0x007f
}


SECTIONS
{
  vectors > VECS    PAGE 0 /* sección de vectores de interrupción */
  .text   > PROG    PAGE 0 /* sección de código programa          */
  .const  > DATA    PAGE 1 /* sección de constantes               */
  .data   > DATA    PAGE 1 /* sección de código dato              */
  .stack  > STACK   PAGE 1 /* sección de memoria para la pila     */
}
```

5.2.4. Ejemplo de creación de un proyecto

El procedimiento para la creación de un proyecto es como sigue:

- Creamos un proyecto con nombre *mulvect.mak*, el cual realizará una multiplicación de dos vectores en punto flotante.
- Editamos el código fuente relacionado con la multiplicación de dos vectores:

```
/* multvect.c*/
/* Definición de los vectores x1 y x2 con longitud N-1 se definen
   como variables globales, por tanto se podrán visualizar en
   memoria dato con las etiquetas igual al nombre de la variable */

const float x1[10]={0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
             0.7, 0.8, 0.9, 1.0};
const float x2[10]={1.0, 1.1, 1.2, 1.3, 1.4, 1.5,
             1.6, 1.7, 1.8, 1.9};
float y;                  //vector resultado

void main()               //función principal
```

```
{
  int i;                  //variable local i como índice
  y=0.0;
  for (i=0; i<10; i++)
     y +=x1[i]*x2[i];     //bloque de repetición para
                          //realizar la multiplicación
}
```

Guardamos el archivo y lo agregamos al proyecto.

- Agregamos el archivo de ligado de comandos, el cual tendrá las definiciones de los espacios de memoria (memoria.cmd).
- Agregamos la librería rts.lib con ubicación */ti/5400/cgtools/rts.lib* en la cual se definen las librerías del compilador C, el símbolo _ c_ int00 que marca el inicio del programa o punto de entrada en el espacio memoria programa y las subrutinas de punto flotante escritas en ensamblador nativo del DSP para usarse desde el compilador de C. Antes de hacer la llamada de la función principal main( ), existe un bloque de código en lenguaje ensamblador donde el compilador de C define el espacio de memoria destinado a la pila y los espacios de memoria dato y programa, tomando esta información del archivo ".cmd".
- Ya realizados los anteriores pasos compilamos el proyecto es cogiendo la opción "Project" en la barra principal del CCS, donde se abrirá un cuadro de dialogo, eligiendo la opción 'Build'.
- Para ejecutar el proyecto compilado, debemos cargar el archivo de salida mulvect.out a la tarjeta DSK, esto al escoger la opción "File" de la barra principal, y en el cuadro de dialogo que se abrirá, elegimos la opción "load", donde especificamos la carpeta y nombre de archivo a cargar (mulvect.out). Finalizada la carga del archivo ".out" ejecutamos el proyecto escogiendo la opción 'Debug', en donde aparece un cuadro de dialogo eligiendo la opción de ejecución completa "Run".

5.3. Ejemplos de programación en lenguaje C

Por medio del CCS podemos programar al DSK utilizando tanto lenguaje ensamblador como lenguaje C. En esta sección se presenta la implantación de la función de autocorrelación

(ecuación 5.1) en diversas formas; en lenguaje ensamblador nativo del C5402, en lenguaje algebraico, en lenguaje C y en lenguaje mixto usando el CCS.

$$y(l) = \sum_{n=0}^{N-1-l} x(n)x(n+l) \tag{5.1}$$

Cabe mencionar que todos los códigos aquí presentados están programados en aritmética de punto fijo, asumiendo el formato q_0.

5.3.1. Creación de un programa en Ensamblador usando el CCS

Para crear un proyecto en el cual el código fuente esté escrito en lenguaje ensamblador nativo del DSP o en el modo algebraico, es necesario adjuntar a dicho proyecto un archivo de ligado de comandos y los archivos en ensamblador. En la figura 5.7 se observa los archivos adjuntos en un proyecto con sólo código en ensamblador. En los siguientes apartados se presenta código escrito tanto en ensamblador convencional como en lenguaje algebraico. Cabe mencionar que para poder trabajar en el modo algebraico se debe activar está opción en la configuración de ensamblado en el CCS. Además, es necesario referenciar el punto de entrada en el espacio programa al incluir una etiqueta llamada "_ c_ int00", de esta manera el ensamblador sabrá desde que dirección se ejecuta el código programa[2].

Programa #1

El programa realiza la función de autocorrelación (ecuación 5.1) en lenguajes ensamblador nativo del C5402 como se muestra:

```
        .mmregs             ;directiva para la definición de las etiquetas de
                            ;los registros mapeados en memoria
        .global _c_int00    ;Referencia del punto de entrada

N_1     .set 63             ;asignación a etiqueta

        .data               ;comienzo del segmento dato

x       .include "seno.dat"   ;se incluye un archivo donde se
                              ;define a una señal senoidal de 64 puntos
```

[2]Esta etiqueta es el homólogo de la directiva ".entry" en el ensamblador del DSP TMS320C50

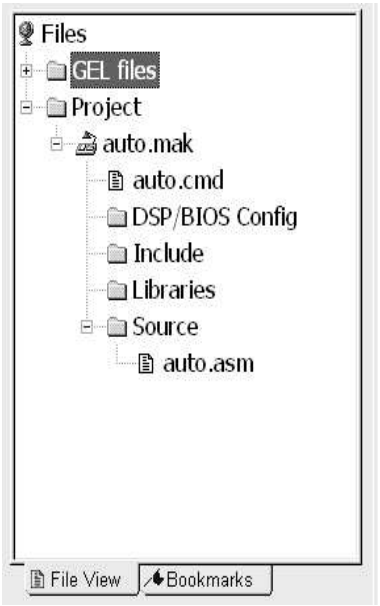


Figura 5.7: Creación de un proyecto con código en ensamblador.

```
y       .word  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0   ; señal resultado
        .word  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        .word  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        .word  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        .word  0,0

temp     .word 1       ;variables auxiliares indiceL .word N_1

       .text           ;comienzo del código programa
_c_int00:              ;punto de entrada o inicio de ejecución del
                       ;programa

        SSBX SXM           ;se activa modo de signo extendido
        LD   #temp,DP      ;apuntador de página en la página de temp
        STM  #x,AR1        ;AR1= dirección de la variable x
        STM  #y,AR2        ;AR2= dirección de la variable y
        STM  #N_1,BRC      ;BRC=N_1, registro contador de bloque
```

```
                        ;de  repetición
        RPTB ciclo      ;comienzo del bloque de repetición
        LD #0,A         ;A=0
        RPT indiceL     ;ciclo de repetición de la siguiente instrucción
        MACP *AR1+,#x,A ;operación de multiplicación - acumulación
        STL  A,*AR2+    ;se guarda el valor de A  en la dirección de y
                        ;apunta AR2 y se incrementa AR2
        LD   #x,0,A     ;A=#x
        ADD  temp,A     ;A=A+temp
        STLM A,AR1      ;AR1=A
        ADDM 1,temp     ;temp=temp+1
        ADDM 0FFFFh,indiceL ;indiceL=indiceL-1
ciclo   NOP               ;Fin del bloque de repetición

fin     B    fin          ;bucle infinito
        .end
```

Programa #2

El siguiente código presenta la implantación de la función de autocorrelación en ensamblador en modo algebraico:

```
        .mmregs       ; directiva para la definición de las etiquetas de
                      ; los registros mapeados en memoria
        .global _c_int00 ; Referencia del punto de entrada

N_1     .set 63       ; asignación a etiqueta

        .data         ; comienzo del segmento dato

x       .include "seno.dat"   ; se incluye un archivo donde se
                              ; define a una señal senoidal de 64 puntos
y       .word  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0   ; señal resultado
        .word  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        .word  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        .word  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        .word  0,0
```

```
temp    .word 1      ; variables auxiliares indiceL .word N_1

      .text          ; comienzo del código programa
_c_int00:            ; punto de entrada o inicio de ejecución de
                     ; programa
       SXM=#1
       DP=#temp      ; Cargar al apuntador de página la página de la
                     ; variable temp
       AR1=#x        ; AR1= con dirección de la variable x
       AR2=#y        ; AR2= con dirección de la variable y
       BRC=#N_1      ; Inicializar BRC=N_1

       blockrepeat(ciclo)  ; comienzo del bloque de repetición
       A=#0
       repeat(@indiceL)    ; ciclo de repetición de instrucción
                           ; siguiente
          macp(*AR1+,#x,A) ; multiplicación-acumulación
       *AR2+ = A           ; salva A en variable y
       A=#x
       A=A+@temp
       AR1=A
       @temp= @temp+1
       @indiceL=@indiceL-1
ciclo  nop                 ;fin del bloque de repetición

fin    goto fin            ;bucle infinito

       .end
```

5.3.2. Creación de un programa en lenguaje C usando el CCS

Para la creación de un proyecto con código en lenguaje C es necesario incluir la librería de soporte en tiempo real (rts.lib), de tal manera que el punto de entrada para el código fuente escrito en lenguaje C será la función main(). Además, al incluirse esta librería se podrá programar en punto flotante debido a que ésta integra las subrutinas para operaciones en aritmética en punto flotante. Como se ha mencionado en otras secciones, el punto de

entrada es la etiqueta _c _ int00, y al incluirse la librería rts.lib, se ejecuta un bloque de instrucciones que comienza en dicho punto de entrada tales que definen la memoria reservada para la pila tomando tal definición del archivo de ligado de comandos, así como la activación del modo de overflow y de extensión de signo. Este bloque de instrucciones finaliza con una llamada a subrutina a la función main() (código en lenguaje C).

Programa #1

El siguiente código presenta el modo de implantar la función de autocorrelación en lenguaje C convencional:

```
#define  N 64            // definición del símbolo N

#include "senoc.dat"     // incluye un archivo donde se define un vector
                         // ''x'' entero de una señal senoidal de 64 puntos

short y[N];              // definición de un vector de salida en formato
                         // entero de longitud N

short n,l;               // definición de variables auxiliares

void  main()                     // función principal
 {
    for (l=0; l<=N-1;l++)        // ciclo externo
     {
       y[l]=0;
       for (n=0; n<=N-1-l; n++) // ciclo interno
         y[l]=y[l]+x[n]*x[n+l]; // multiplicación - acumulación
     }
 }
```

Programa #2

Una forma de optimizar el código al programar en lenguaje C es usando las funciones intrínsicas que incluye el compilador de lenguaje C, es decir, emplear funciones que están escritas en ensamblador para realizar diversas operaciones aritméticas, las cuales pueden utilizarse como si fuesen funciones escritas en lenguaje C. El siguiente programa ejemplifica el uso de la función _ smac( ) la cual efectúa la multiplicación y acumulación entre dos valores

dados y almacena el resultado en una tercer variable. Cuando el programador requiera usar las funciones intrínsicas no es necesario incluir alguna librería debido a que el compilador de C del C5402 las reconoce en forma implícita. Para mayores referencias de las funciones intrínsicas en [35].

```
#define  N 64            // definición del símbolo N


#include "senoc.dat"     // incluye un archivo donde se define un vector
                         // ''x'' entero de una señal senoidal de 64 puntos

short y[N];              // definición de un vector de salida en formato
                         // entero de longitud N

short n,l;               // definición de variables auxiliares

void  main()             // función principal
 {
    short i;
    for (l=0; l<=N-1;l++)             // ciclo externo
     {
      y[l]=0;
      for (n=0; n<=N-1-l; n++)        // ciclo interno
        y[l]=_smac(y[l],x[n],x[n+l]); // uso de la función _smac( )
     }

  for (i=0; i<=N-1; i++)
    y[i]=y[i]>>1; // un corrimiento hacia la derecha, Esto es para
                  // mantener el resultado con el formato de punto fijo q0
 }
```

Programa #3

Otra forma de optimizar el código en la creación de un proyecto es utilizar las funciones definidas por la librería dsplib.h. Estas funciones están escritas en lenguaje ensamblador pero de igual manera que las funciones intrinsicas pueden usarse como funciones definidas en lenguaje C. Las funciones que integran la librería dsplib.h, son un conjunto de funciones básicas del procesamiento digital de señales tales que calculan correlaciones, funciones de

autocorrelación, filtros FIR, filtro IIR, FFT, etc. Cabe destacar que el formato numérico en que están definidas estas funciones es en punto fijo en q_{15}, por tanto si se trabajase en otro formato se tendría que adecuar tanta los valores de entrada como de salida. El siguiente código ejemplifica el uso de la librería dsplib.h, así como la adecuación del formato numérico:

```
#include "dsplib.h"     // librería dsplib.h

#include "senoc.dat"    // incluye un archivo donde se define un
                        // vector entero de una señal senoidal de 64 puntos
short xi[N],y[N];       // definición de dos vectores enteros

void  main()            // función principal
 {
   short i;
   for (i=0; i<=N-1; i++)
     xi[i]=x[i]<<8;     // se adecúa el vector de entrada al
                        // formato q15
   acorr(xi,y,N,N,raw); // función de autocorrelación definida por
                        // la librería dsplib.h
   for (i=0; i<=N-1; i++)
     y[i]=y[i]>>1;      // se adecúa la salida para recuperar el
                        // formato original
 }
```

Además de incluir la librería dsplib.h se debe adjuntar al proyecto las librerías tms320.h y 54xdsp.lib tal como se muestra en la figura 5.8. Para mayores referencias de las funciones definidas en la librería dsplib.h consultar [38].

Programa Mixto

El compilador de C que integra el CCS tiene la ventaja de poder incluir funciones propias de usuario que estén definidas en lenguaje ensamblador, al incluir al proyecto archivos en código en C y en ensamblador. Para poder usar funciones externas escritas en lenguaje ensamblador desde un archivo escrito en C es necesario definir dichas funciones mediante la referencia "extern", y en el archivo escrito en lenguaje ensamblador referenciar las funciones y variables a usar como funciones y variables globales. A continuación se presenta el código

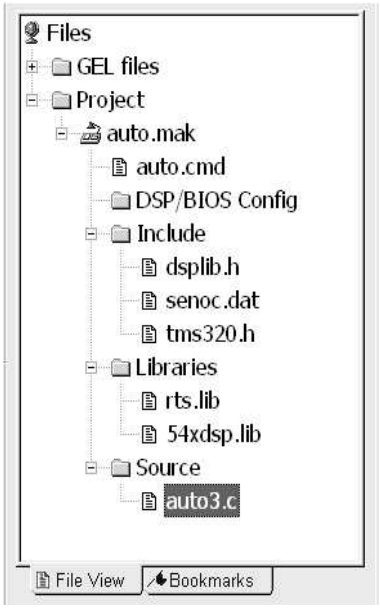


Figura 5.8: Creación de un proyecto con código en lenguaje C usando la librería dsplib.h.

en lenguaje C y en lenguaje ensamblador para el cálculo de la función de autocorrelación ejemplificando el uso de funciones externas escritas en ensamblador.

```
//mixto.c
/* Código fuente en lenguaje C */

#include "dsplib.h"     // librería dsplib.h

#include "senoc.dat"    // incluye un archivo donde se define un
                        // vector entero de una señal senoidal de 64 puntos
short xi[N],y[N];       // definición de dos vectores enteros

extern void corri1();   // referencia de funciones externas

extern void corri2();

void  main()             // función principal
  {
    corri1();            // llama la función externa escrita en ensamblador
```

```
    acorr(xi,y,N,N,raw); // llama la función de autocorrelación definida
                         // por la librería dsplib.h
    corri2();            // llama la función externa escrita en ensamblador
  }
```

En la figura 5.9 se observan los diferentes archivos que integran al proyecto y el código para las funciones externas definidas desde un archivo en ensamblador es:

```
;asmfunc.asm ;funciones externas que realizan corrimientos

     .mmregs
     .global _corri1,_corri2 ; se referencian las funciones
                             ; corri1 y _corri2 como funciones globales
     .global _x,_xi,_y       ; se referencian estas variables como globales
N_1  .set 63                 ; definición del símbolo N_1

_corri1                      ; función corri1( )
      AR3=#_x                ; _x vector de entrada
      AR4=#_xi               ; _xi vector de salida
      BRC=#N_1
      blockrepeat(ciclo)
      A = *AR3+
      A= A << 8              ; 8 corrimientos hacia la izquierda
      *AR4+ = A
ciclo nop
      return                 ; regreso de subrutina

_corri2                      ; función corri2( )
      AR3=#_y
      BRC=#N_1
      blockrepeat(ciclo1)
      A= *AR3
      A=A>>1                 ; un corrimiento hacia la derecha
      *AR3+ = A
ciclo1 nop
      return                 ; regreso de subrutina

      .end
```

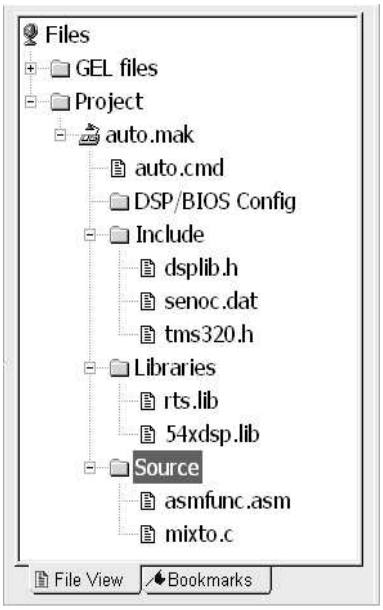


Figura 5.9: Creación de un proyecto con código en lenguaje C usando la librería dsplib.h y funciones externas escritas en ensamblador.

Además, mostramos algunos ejemplos de aplicación escritos en lenguaje C, para la generación de funciones seno en formato de punto fijo q_{15} y un ejemplo de implementación para filtros FIR tipo paso-bajas, paso -banda y paso - altas.

Ejemplo de generación de funciones seno en q_{15}

```
/*
    gen_sen.c - Generación de funciones seno en Q15
    generación de una señal x(n) con tres funciones
    senoidales a diferente frecuencia
    FUNCIONAL:

    - Incluir librerías \rst.lib para c_boot
    - Salida x(i) en 0x1f2e
*/
```

```
#include <math.h>

    #define T 0.000125               // 8000 Hz
    #define f1 800                   // 800  Hz
    #define f2 1800                  // 1800 Hz
    #define f3 3300                  // 3300 Hz
    #define PI 3.1415926
    #define dos_pi_f1_T (2*PI*f1*T)  // 2*pi*f1/Fs
    #define dos_pi_f2_T (2*PI*f2*T)  // 2*pi*f2/Fs
    #define dos_pi_f3_T (2*PI*f3*T)  // 2*pi*f3/Fs
    #define a1 0.7                   // Magnitud de onda 1
    #define a2 0.2                   // Magnitud   "     2
    #define a3 0.333                 // Magnitud   "     3
    #define N 512

    const int sineTable[5]={0x01,0x02,0x03,0x04,5};
    static unsigned int n=0;
    volatile int x[N];

void main()
{
    float temp;
    int   i;
    for (i=0; i<N; i++)
    {
        temp  = a1*cos((double)dos_pi_f1_T*n);
        temp += a2*cos((double)dos_pi_f2_T*n);
        temp += a3*cos((double)dos_pi_f3_T*n);

        n++;
        x[i] = (int)(0x0007f*temp+0.5);  // queda en locc 7fffh
    }
    while(1) {}
}
```

Ejemplo de implementación de filtros FIR

El siguiente ejemplo incluye la generación de tres señales senoidales y se implementan tres filtros FIR, filtro paso - bajas (hl), filtro paso - banda (hb) y paso - altas (hh), en la función "filtra" se pueden cambiar la respuesta al impulso para seleccionar cualquiera de estos filtros:

```
/*   GENERA FUNCIONES SENO Y FILTRO FIR, FPB, FPBW  y FPA
     Con Nf=25 y ventana de Hamming
     gen_sen.c - Generación de funciones seno en Q15
     generación de una señal x(n) con tres funciones
     senoidales a diferente frecuencia

    - Incluir librerías \rst.lib para c_boot
    - x(i)  en  0x1f26
    - y(n)  en  0x2326


                          ________
    Sistema               |       |
        x=x1+x2+x3 -------->|  h    |-------> y(n)
                          |_______|

    Utilizando Funciones y paso de vectores */

    #include <math.h>
    #define T 0.000125              // 8000 Hz
    #define f1 100                  // 100  Hz
    #define f2 1800                 // 1800 Hz
    #define f3 3900                 // 3300 Hz
    #define PI 3.1415926
    #define dos_pi_f1_T (2*PI*f1*T) // 2*pi*f1/Fs
    #define dos_pi_f2_T (2*PI*f2*T) // 2*pi*f2/Fs
    #define dos_pi_f3_T (2*PI*f3*T) // 2*pi*f3/Fs

    #define a1 0.7                  // Magnitud de onda 1
    #define a2 0.2                  // Magnitud   "     2
    #define a3 0.333                // Magnitud   "     3
```

```
    #define N  512
    #define Nf 25

/* Coeficientes del filtro FIR hl=FPB con N=25 */

  float hl[Nf]={-0.0019,-0.0028,-0.0042,-0.0052,-0.0038, 0.0023, 0.0151,
                 0.0351, 0.0609, 0.0890, 0.1143, 0.1321, 0.1384, 0.1321,
                 0.1143, 0.0890, 0.0609, 0.0351, 0.0151, 0.0023,-0.0038,
                -0.0052,-0.0042,-0.0028,-0.0019};

// h(n) del filtro paso Banda

  float hb[Nf]={ 0.0033,-0.0131, 0.0000, 0.0307, 0.0141,-0.0562,-0.0482,
                 0.0717, 0.0967,-0.0614,-0.1402, 0.0243, 0.1578, 0.0243,
                -0.1402,-0.0614, 0.0967, 0.0717,-0.0482,-0.0562, 0.0141,
                 0.0307, 0.0000,-0.0131,-0.0033};

// h(n) del filtro paso altas
  float hh[Nf] ={ 0.0015,-0.0015,-0.0042, 0.0015, 0.0123, 0.0037,-0.0264,
                 -0.0233, 0.0432, 0.0761,-0.0570,-0.3065, 0.5611,-0.3065,
                 -0.0570, 0.0761, 0.0432,-0.0233,-0.0264, 0.0037, 0.0123,
                  0.0015,-0.0042,-0.0015,0.0015};

  float x[N]; float y[N];  // Salida float

   filtra(float h[ ], float x[ ], int );
   void   genera_sen3(void);

void main()
 {
   int n;

//  GENERA x(n) senoidal
     genera_sen3( );

//  FILTRA
     for (n=0; n<N; n++)
```

```
        y[n]=filtra(hl,x,n);
    while(1) {}
}   // main

//  GENERA
void genera_sen3(void)
 {
    int n  ;
    float temp;
    extern float x[N];
    temp=0.0;
    for (n=0; n<N; n++)
      {
        temp  = a1*cos((double)dos_pi_f1_T*n);
        temp += a2*cos((double)dos_pi_f2_T*n);
        temp += a3*cos((double)dos_pi_f3_T*n);
        x[n]=temp;
      }
 }
// FILTRA
  float filtra(float h[ ],float x[ ], int n)
   {
     int j;
     float x1[Nf],temp;
        for(j=0; j<Nf ;j++)
          {
            if ( (n-j) < 0 )
               x1[j] = 0.0;
            else
               x1[j] = x[n-j];
          }
        temp = 0.0 ;
        for(j=0;j<Nf;j++)
          temp = temp + h[j]*x1[j];
        return temp;
   }
```

Archivo ".cmd" para ligar las aplicaciones anteriores

```
/* Archivo memoria.cmd  para el ligado de comandos */

MEMORY
 {
  PAGE 1: SCRATCH : origin = 0x0060, length = 0x0020
          STACK   : origin = 0x1180, length = 0x0560
          DATA    : origin = 0x1f00, length = 0x3f7f /*memoria dato    */
  PAGE 0: PROG    : origin = 0x0100, length = 0x3f70 /*memoria programa*/
  PAGE 0: VECS    : origin = 0x0080, length = 0x007f
                  /*vectores de interrupción*/
  }

 SECTIONS
   {
     vectors > VECS    PAGE 0
     .text   > PROG    PAGE 0
     .cinit  > PROG    PAGE 0
     .switch > PROG    PAGE 0

     .const  > DATA    PAGE 1
     .data   > DATA    PAGE 1
     .bss    > DATA    PAGE 1
     .stack  > STACK   PAGE 1
     .trap   > SCRATCH PAGE 1
   }
```

Resumen

En este capítulo se presentaron diferentes formas de programar una aplicación en el DSP TMS320C54xx, en el sentido de optimizar el código necesario, usando lenguaje ensamblador, algebraico, C y mixto. La elección de la forma de programar una aplicación será acorde a las necesidades de ésta, es decir, si la aplicación consta de funciones aritméticas podemos emplear las funciones intrínsicas desde un archivo en código en C, optimizando recursos de memoria y tiempo de cálculo. Si la aplicación integra operaciones tales como convoluciones, correlaciones, filtros, etc. podemos usar la librería dsplib.h, ahorrándonos la implementación

de estas funciones en lenguaje C y optimizando recursos debido a que estas funciones están escritas en ensamblador. Si necesitamos implementar algoritmos en aritmética de punto flotante, podemos usar el lenguaje C y evitarnos la tarea de programar subrutinas en lenguaje ensamblador para aritmética de punto flotante. O bien, crear un proyecto que incluya funciones escritas tanto en lenguaje C como en lenguaje ensamblador para facilitar la implementación de una aplicación, tal que pudiera integrar algoritmos u operaciones tales que su programación fuera más sencilla en cualquiera de éstos lenguajes.

Capítulo 6

Periféricos

Otros de los aspectos que le dan potencialidad de procesamiento a un DSP son sus periféricos, ya que a través de éstos puede transferir información con el mundo exterior, de esta forma el DSP se convierte en el centro de un sistema de procesamiento digital de señales para grandes aplicaciones, y sobre todo aquellas que se necesiten hacer en tiempo real.

Los periféricos que maneja el DSP TMS320C54xx son controlados por un conjunto de registros mapeados en memoria, donde el programador puede configurar, inicializar y habilitar el modo de operación de estos periféricos. A continuación se da una breve exposición de los periféricos internos que integran a la familia TMS32054xx[1], haciendo hincapié en su descripción, constitución, registros e interrupciones asociados. Sin embargo, no todos lo integrantes de la familia TMS320C54xx incluyen los periféricos aquí presentados, por tanto, mostramos en el cuadro 6.1, los periféricos que incluye cada DSP perteneciente a la familia TMS320C54xx.

6.1. Puertos paralelos de entrada/salida

La familia TMS32054xx puede direccionar hasta de 64k puertos de entrada/salida (I/O) permitiendo el acceso a periféricos utilizados en aplicaciones para el PDS. El acceso a puertos es multiplexado sobre el mismo bus de direcciones y datos utilizados para accesar a memoria programa y dato. El espacio I/O es distinguido por la activación de las señales externas $\overline{IS}$ e $\overline{IOSTRB}$ en bajo cuando el DSP ejecuta una instrucción de PORTW o PORTR:

```
PORTR 05h,DATO   ;  lee un dato del puerto 5h
```

[1] A lo largo de este capítulo nos referimos a las direcciones de los registros mapeados para el DSP C5402.

DSP/periférico	C542	C543	C545	C546	C548	C549	C5402
Puertos paralelos I/0	√	√	√	√	√	√	√
Pines Externos	√	√	√	√	√	√	√
Temporizador	√	√	√	√	√	√	√
Puerto serial			√	√			
Puerto serial bufereado	√	√	√	√	√	√	
Puerto serial bufereado multicanal							√
Puerto TDM	√	√			√	√	
Puerto HPI 8 bits	√		√		√	√	
Puerto HPI 8 bits mejorado							√
Canales DMA	√	√	√	√	√	√	√
Generador de Reloj	√	√	√	√	√	√	√
Generador de estados de espera	√	√	√	√	√	√	√
Banco de Interrupción	√	√	√	√	√	√	√

Cuadro 6.1: Periféricos que incluyen los DSP's de las familias TMS320C54x y TMS320C5402.

```
                ;  y lo escribe en la localidad de memoria DATO
PORTW SALIDA,06h ;  lee un dato en la localidad de memoria SALIDA
                ;  y lo escribe en el puerto 06h.
```

6.2. Pines de Entrada/Salida de Propósito General.

La familia TMS320C54xx tiene dos pines entrada y salida de propósito general: $\overline{BIO}$ y XF. $\overline{BIO}$ es un pin de entrada el cual puede ser usado para monitorear el estado de dispositivos externos, en especial, es muy útil como una alternativa para impedir que en los ciclos o bloques de repetición no sean interrumpidos en llamadas de atención a interrupción. Una instrucción de salto puede ejecutarse condicionalmente dependiendo del estado de la entrada del $\overline{BIO}$. XF es un pin de salida controlado por software que permite señalizar dispositivos externos, configurándolo en estado alto cuando se pone en uno el bit XF (registro ST1) y en estado bajo cuando se limpia dicho bit.

6.3. Temporizadores de 16 bits con preescalador de 4 bits

El C54xx dispone de circuitos temporizadores de 16 bits con preescalador de cuatro bits programables por software que consisten en tres registros mapeados en memoria para su operación y pueden ser utilizados periódicamente para generar interrupciones. La resolución del temporizador depende del intervalo del reloj del CPU del procesador. Los registros mapeados en memoria del temporizador son el registro del temporizador (TIM), el registro de Período (PRD), y el registro de control del temporizador (TCR). El registro TIM cuenta de forma descendente en la caída de CLKOUT. Los registros TIM y PRD son fijados a su máximo valor 0FFFFh en el reset. Una interrupción del temporizador (TINT) es generada cada vez que el registro TIM llega a cero, el registro TIM es recargado con el valor almacenado en el registro PRD en el próximo ciclo. Está característica es útil para operaciones de control y sincronización de muestreo o escritura de periféricos. El diagrama de bloques del temporizador se presenta en la figura 6.1.

La razón de tiempo de interrupción del temporizador está dada por la ecuación (6.1):

$$f_{TINT} = \frac{f_c}{(TDDR+1)(PRD+1)} \tag{6.1}$$

Donde f_{TINT} es la razón de interrupción del temporizador, f_c es la frecuencia de reloj CLKOUT, TDDR es el escalamiento de cuatro bits y PRD es el escalamiento de 16 bits.

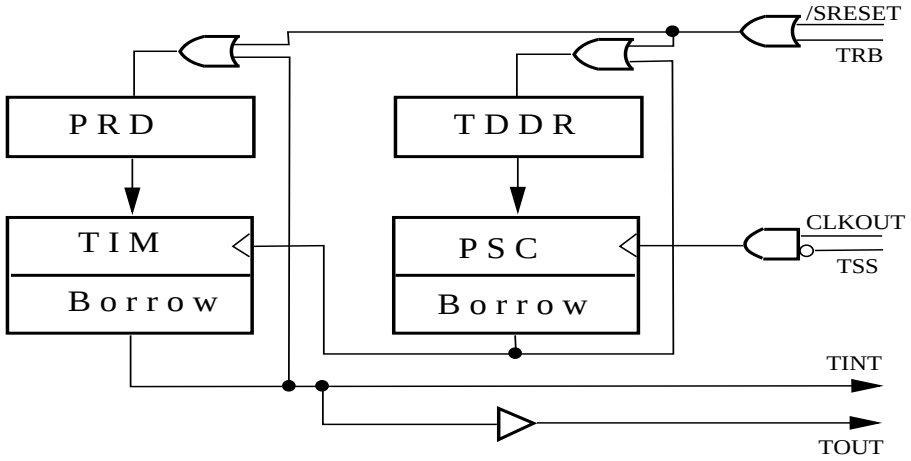


Figura 6.1: Diagrama de bloques del temporizador de 16 bits

El primer contador (descendente) TDDR se encuentra en el registro TCR (0-3 bits), los bits 6-9 del registro TCR corresponden al contador del preescalor (PSC). Cuando el registro PSC llega a cero es cargado con el valor almacenado en el registro TDDR. La operación del temporizador es controlada por el registro de control TCR donde el bit TSS (bit 4) permite detener la operación del temporizador al escribirle un uno, y el bit TRB (bit 5) permite cargar de nuevo el período. Los bits SOFT y FREE (bits 11 y 10 en TCR, respectivamente) determinan el estado del temporizador cuando éste es detenido.

La secuencia para la configuración del temporizador es como sigue:

- Deshabilitar Interrupciones poniendo en uno al bit de estado INTM (Habilitación de interrupciones mascarables)
- Escribir los valores necesarios al registro PRD y al campo de bits TDDR al registro TCR.
- Habilitar la interrupción del temporizador (desenmarcararla) en el registro IMR (Registro de Interrupciones Mascarables.)
- Limpiar cualquier interrupción pendiente en el registro IFR (Registro de Banderas de Interrupción).
- Habilitar interrupciones limpiando el bit de estado INTM.

6.3.1. Ejemplo de uso del temporizador con interrupciones

Presentamos un ejemplo escrito en lenguaje C mostrando el uso de la interrupción del temporizador, así como la escritura a un puerto de entrada/salida. El programa consiste en un contador de 0-7 visualizado mediante los leds indicadores que integra la tarjeta DSK del DSP TMS320C5402. El proyecto consta de cuatro archivos: timer.c, vectores.asm, timer.cmd y rts.lib.

Programa principal

```
/* timer.c */

#define GLOBAL_INT_ENABLE     asm( " rsbx intm ")
#define GLOBAL_INT_DISABLE    asm( " ssbx intm ")

ioport short port0;  //referencia al puerto 0 de la tarjeta DSK

// Es conveniente utilizar puntero de puntero para la lectura y
// escritura de los registros mapeados desde el lenguaje C

volatile short *imr   =   (short *) 0x00; // reg. mapeado IMR
volatile short *ifr   =   (short *) 0x01; // reg. mapeado IFR
volatile short *pmst  =   (short *) 0x1d; // reg. mapeado PMST
volatile short *prd   =   (short *) 0x25; // reg. mapeado PRD
volatile short *tcr   =   (short *) 0x26; // reg. maepado TCR
volatile short *clkmd =   (short *) 0x58; // reg. mapeado CLKMD

char dato=0x00;  //variable auxiliar

// subrutina de atención a interrupción del temporizador 0 del  C5402

interrupt void inttimer(void)
  {
    if (dato==0x08)
      {
        port0=0x00;
        dato= 0x00;
      }
```

```
      else
        {
         port0 = dato; // en ensamblador algebraico: PORT(0h)=@dato
         dato+=0x01;
        }
    }

 void main(void)        // programa principal
 {
    GLOBAL_INT_DISABLE; // deshabilitar interrupciones
    *pmst  =  0x00A0;   // mapeamos los vectores de interrupción
    *clkmd =  0x4007;   // reloj a 100 MHz
    *imr   =  0x0008;   // habilitamos interrupción del timer0
    *tcr   =  0x062F;
    *prd   =  0x0FFFF;  // interrupción cada 100 ms.
    *ifr   = *ifr;      // limpiar banderas
    GLOBAL_INT_ENABLE;  // habilitar interrupciones
    while(1){};         // bucle infinito
  } // fin de programa principal
```

Código de definición de los vectores de interrupción ".asm"

```
    ; vectores.asm

    .mmregs             ; definición de las etiquetas de los reg. mapeados

    .sect    " vectors" ; sección de los vectores de interrupción

    .ref     _main      ; referencia a la función principal
    .ref     _c_int00   ; referencia al punto de entrada
    .ref     _inttimer  ; referencia a la subrutina de interrupción


reset:  b _c_int00
        nop
```

```
        nop

       .space 72*16     ; se apartan 72*16 bits
       ;(vectores de interrupción no usados)

tint0:  b _inttimer     ; salto a la subrutina de atención de
        nop             ; interrupción asociada al temporizador 0
        nop
        .end
```

Definición de los espacios de memoria a emplear

```
  /* timer.cmd */

 MEMORY
  {
     PAGE 0:

     VECS:    org=0080h, len=007fh /*vectores de interrupción*/
     P:       org=0100h, len=2B00h /*espacio programa*/

 PAGE 1:
     D:       org=2C00h, len=13ffh /*espacio dato*/
}

SECTIONS
  {
      vectors : load = VECS PAGE 0
     .text    > P   PAGE 0
     .bss     > D   PAGE 1
     .stack   > D   PAGE 1
  }
```

6.3.2. Ejemplo de uso de escritura de puertos de entrada/salida en lenguaje ensamblador

Presentamos un ejemplo escrito en lenguaje ensamblador mostrando el uso de la escritura a un puerto de entrada/salida. El programa consiste en un contador de 0-7 visualizado mediante los leds indicadores que integra la tarjeta DSK del DSP TMS320C5402. El proyecto consta de 2 archivos: leds.asm y leds.cmd.

Programa principal

```
*     leds.asm
*     ESCRITURA A PUERTO 0h para escritura de LEDS

          .title   "ESCRITURA A LEDS"
          .mmregs
          .width   90
          .length  55
          .ref _c_int00          ; Para reset
          .ref _main             ; Inicio de programa

*
*       Sección de Datos y variables
*
    .sect ".data"

 DATO  .word   0               ; Dato temporal para escribir a PTO.
 N     .set    2000
              .sect".text"
_c_int00 _main
               SP = #0FFAh     ; Ubica el Stack Pointer
               DP = #DATO      ; Pag. para direc. directo.

CICLO
               PORT(0h)= @DATO ; Escribe DATO a PTO. 0h (mod. directo)
               CALL  TARDA     ; Va un retardo
               A = @DATO
               A = A + #1      ; Incrementa dato=dato+1
               A = A & #07h    ; Solo cuenta de 0 a 7 ==> Leds [D2 D1 D0]
```

```
               @DATO = A
CAMBIA         GOTO  CICLO     ; Regresa al loop

*
*  RETARDO
* * TARDA        AR5 = #N

SIGUE        REPEAT(#N)        ; N + 1 veces
             NOP
             REPEAT(#N)        ; N + 1 veces
             NOP

             IF(*AR5- !=0) GOTO SIGUE   ; (N**3)*10ns
             RETURN

             .end
```

Archivo ".cmd"

```
/* lcds.cmd */

  MEMORY
   {
     PAGE 1:  /* memoria dato */

         SCRATCH : origin = 0x0060, length = 0x001f
         STACK   : origin = 0x1180, length = 0x0560
         DATA    : origin = 0x1f00, length = 0x4000

     PAGE 0:  /* memoria programa */
             PROG    : origin = 0x0100, length = 0x3f70
             /* vectores de interrupción*/
             VECS    : origin = 0x0080, length = 0x007f
   }
```

```
SECTIONS
{
   vectors > VECS    PAGE 0
  .text    > PROG    PAGE 0
  .cinit   > PROG    PAGE 0
  .switch  > PROG    PAGE 0

  .const   > DATA    PAGE 1
  .data    > DATA    PAGE 1
  .bss     > DATA    PAGE 1
  .stack   > STACK   PAGE 1
  .trap    > SCRATCH PAGE 1
}
```

6.4. Puerto Serial

La familia TMSC54xx incluye puertos serie bidireccional full duplex, donde esta clase de puerto provee una comunicación directa a dispositivos tales como codecs, convertidores seriales A/D, otros sistemas seriales y para aplicaciones multiprocesos a través del puerto serial TDM. Las señales del puerto serie son compatibles con codecs y otros dispositivos seriales.

La máxima frecuencia de operación del puerto serie cuando utiliza el reloj interno es de 10 Mbits/s a 25 ns y 12.5 Mbits/s a 20 ns. Para la operación del puerto serie (figura 6.2) contiene los pines externos:

- CLKX: señal de reloj de transmisión.
- CLKR: Señal de reloj de recepción.
- DX: señal de transmisión de dato serial, envía el dato actual.
- DR: Señal de recepción de dato serial, recibe el dato actual.
- FSX: señal de sincronización de "frame" de transmisión, inicia la transferencia de un "frame".
- FSR: señal de sincronización de "frame" de recepción.

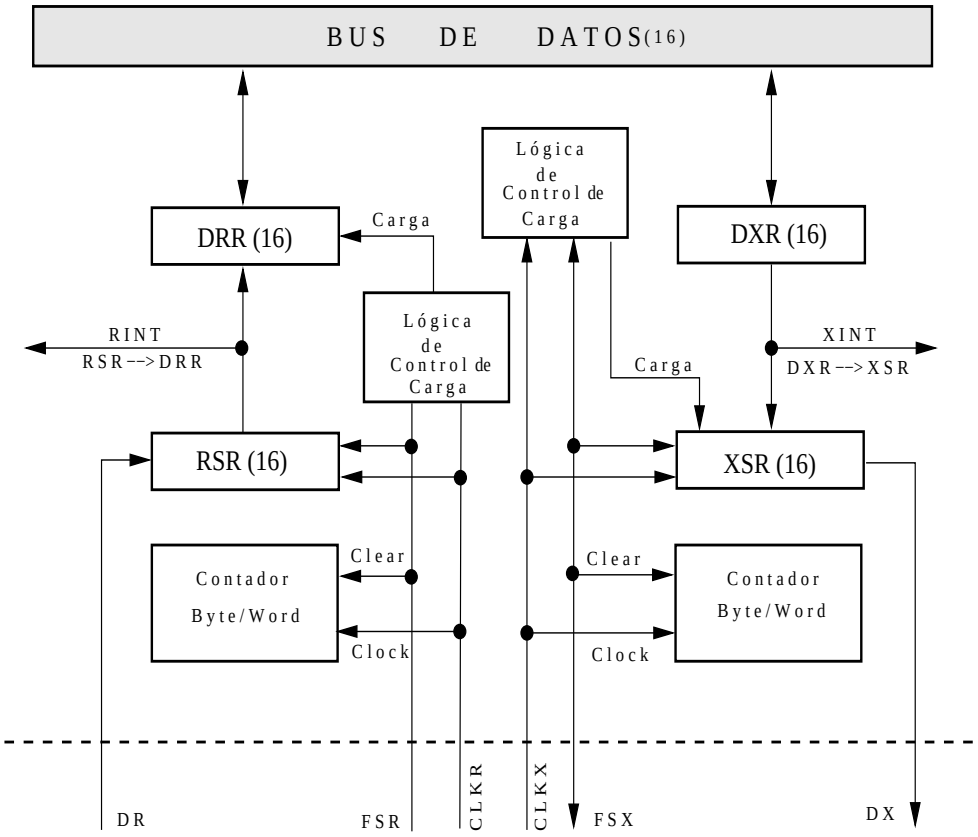


Figura 6.2: Diagrama del puerto serie del C54x

Además, para su operación el puerto serie utiliza los registros mapeados:

- SPC: Registro de control del puerto serie.
- DXR: Registro de transmisión de datos.
- DRR: Registro de recepción de datos.
- XSR: Registro de corrimiento en la transmisión.
- RSR: Registro de corrimiento en la recepción.

Un dato a transmitir es escrito en el registro DXR el cual copia el dato en el registro XSR que efectúa el corrimiento de bits para hacer la transmisión serial sobre el pin DX, tan

pronto como la copia DXR a XSR se termine otro dato puede escribirse en el registro DXR, a la vez ocurre una transición 0-1 en el bit de transmisión lista (XRDY bit 11) del registro de control SPC y se genera una interrupción de transmisión.

Para la recepción un dato en el pin DR es corrido dentro del registro RSR, el cual es copiado en el registro de recepción de dato (DRR) y el dato puede ser leído. Una vez completada la copia de RSR a DRR existe una transición 0-1 en el bit de recepción lista (RRDY bit 10) del registro de control SPC y a la vez se genera una interrupción de recepción (RINT). El puerto serie es doblemente buffereado porque los datos puede ser transferidos de DXR o DRR mientras otra transmisión o recepción se efectúa. En el cuadro 6.2 se muestra la descripción del campo de bits del registro de control SPC.

Para la configuración del puerto serie es necesario escribir a los bits 1-5 del registro SPC. Sin embargo, antes de escribir a estos bits hay que efectuar un reset sobre los bits 6 y 7 (XRST y RRST), es decir escribir ceros a estos bits, y luego escribir a los bits 1-5, para que la configuración del puerto quede habilitada se ponen los bits 6 y 7 a unos.

6.5. Puerto Serial Bufereado.

La familia TMS320C54xx incluye en sus periféricos internos puertos seriales bufereados (BPs) que permiten realizar interfaces directas con dispositivos externos como otros DSP's de la familia C54xx, codecs y otros dispositivos. Los BPs están basados en las interfaces de puertos serie estándar que se encuentran en los DSP's de la familia C50. Los puertos BPs constan de las siguientes características:

- Comunicación Full-Duplex.
- Registros de datos de doble Buffer que permiten una comunicación de datos continua.
- "Framing" y "Cloking" independiente para la recepción y transmisión.

Además tienen capacidades para:

- Una amplia selección de tamaños de datos incluyendo 8,12,16,20,24 o 32 bits.
- Polaridad programable para la sincronización de "frames" y reloj.
- Reloj interno programable y generación de "frames".

Bit	Nombre	Descripción
15	Free	Este bit es usado en conjunción con el bit Soft para determinar el estado del reloj del puerto serial.
14	Soft	Este bit es usado en conjunción con el bit Free para determinar el estado del reloj del puerto serial.
13	RSRFULL	Este bit indica cuando en el registro de recepción ha ocurrido un sobreflujo. El sobreflujo ocurre cuando el RSR está lleno y el DDR no ha sido leído desde la última transferencia RSR-DDR.
12	$\overline{XSREMPTY}$	Este bit indica cuando el XSR está vacío y el DXR no ha sido cargado desde la última transferencia DXR-XSR.
11	XRDY	Una transición de 0 - 1 en este bit indica que el contenido de DXR ha sido copiado en XSR y DXR puede ser cargado con un nuevo dato.
10	RRDY	Una transición de 0 - 1 en este bit indica que el contenido de RSR ha sido copiado a DDR y el dato está listo para ser léido.
9	IN1	Este bit permite que el pin del reloj CLKX puede ser usado como un pin de entrada.
8	IN0	Este bit permite que el pin del reloj CLKR puede ser usado como un pin de entrada.
7	$\overline{RRST}$	Este bit es usado para reinicializar y habilitar la recepción.
6	$\overline{XRST}$	Este bit es usado para reinicializar y habilitar la transmisión.
5	TXM	Este bit configura a FSX como un pin de entrada (TXM=0) o de de salida (TXM=1).
4	MCM	En este bit se especifica la fuente para el reloj CLKX.
3	FSM	En este bit se indica la forma de sincronización de los pulsos FSX y FSR.
2	FO	Este bit especifica la longitud de la palabra a ser usada. FO=0 $\Rightarrow$ 16 bits, FO=1 $\Rightarrow$ 8 bits.
1	DLB	Modo digital de lazo cerrado.
0	RES	Reservado.

Cuadro 6.2: Registro de control de puerto serie SPC.

- Para las versiones de puerto serie bufereado multicanal (C5402) se tiene transmisión Multicanal y recepción de 128 canales y compresión-expansión de la ley μ y la ley A.

Los BPs consisten de canales separados para la recepción y transmisión que operan de manera independiente. La interface externa de cada BP (figura 6.3) consiste de los siguientes pines externos:

- BCLKX: Reloj de referencia para la transmisión.
- BDX: Transmisión de datos.
- BFSX: "Frame" de sincronía para la transmisión.
- BCLKR: Reloj de referencia para la recepción.
- BDR: Recepción de datos.
- BFSR: "Frame" de sincronía para la recepción.

En el transmisor, el "frame" de sincronía de transmisión y el reloj para la transmisión son señalizados por los pines BFSX y BCLKX, respectivamente. El CPU o los canales de DMA pueden iniciar la transmisión de datos escribiendo al registro de transmisión de datos (DXR). Los datos escritos en el registro DXR son corridos hacia el pin BDX a través del registro de corrimiento de transmisión (XSR). Esta estructura permite al DXR recargarse con la siguiente palabra a ser transmitida mientras la palabra presente está en progreso. En el receptor, el "frame" de sincronía de recepción y el reloj para la recepción son señalizados por los pines BFSR y BCLKR, respectivamente.

EL CPU o los canales de DMA pueden leer el dato recibido desde el registro de recepción de datos (DRR). Los datos recibidos en el pin BDR son corridos en el registro de corrimiento de recepción (RSR) para después ser guardados en el registro Buffer de recepción (RBR). Si el registro DRR está vacío, el contenido del RBR es copiado dentro del DRR. La operación contraria, el RBR contiene el dato hasta que el DRR esté disponible. Esta estructura permite el almacenamiento de dos palabras previas mientras la recepción de una nueva palabra está en progreso.

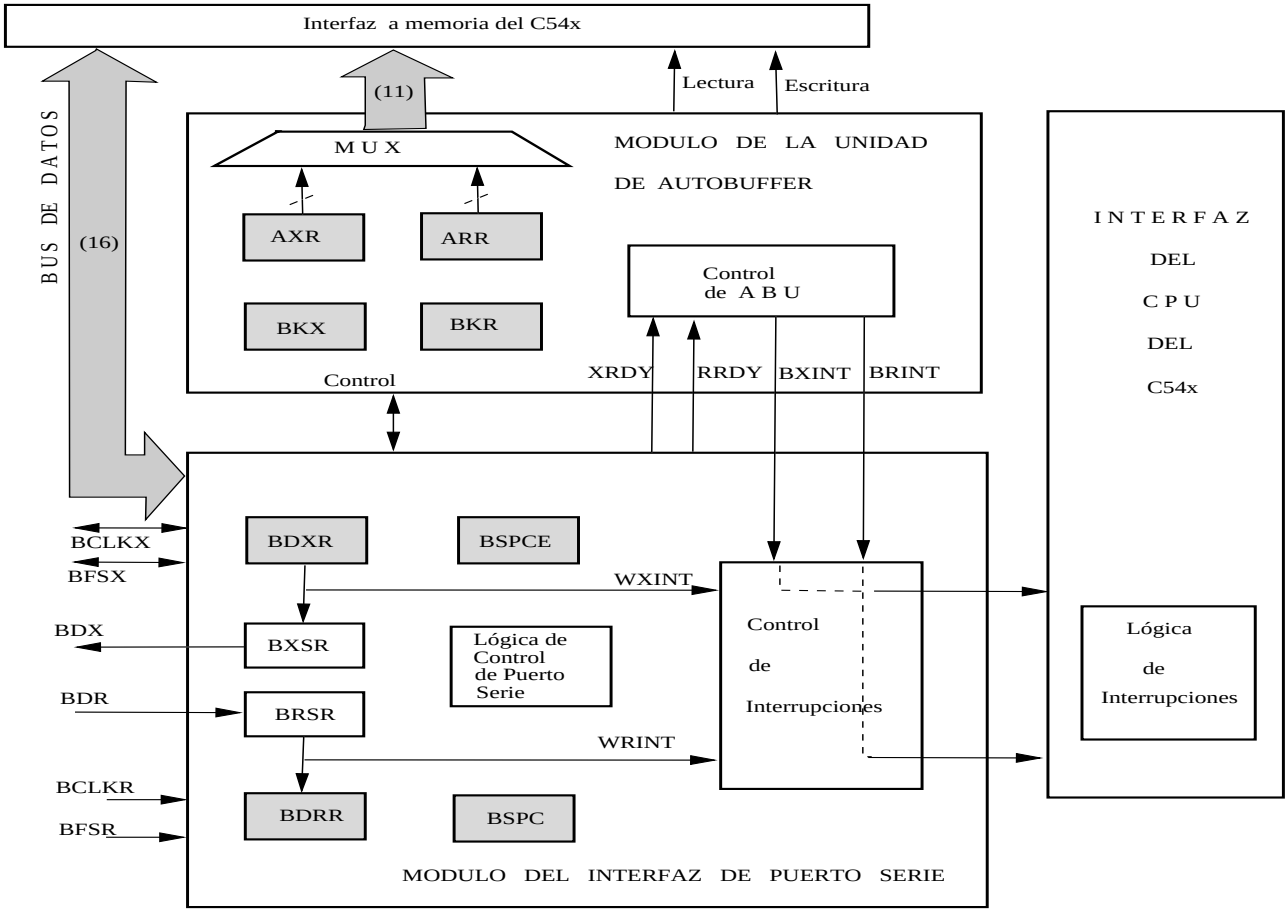


Figura 6.3: Diagrama del puerto serie bufereado del C54xx

6.6. Puerto serie multiplexado por división de tiempo (TDM)

El puerto serie multiplexado por división de tiempo (TDM) permite una comunicación serial con otros siete DSPs proveyendo una poderosa interface para aplicaciones multiprocesos. Para la operación del puerto serie TDM se pone un uno en el bit TDM del registro de configuración de puerto serie TDM (TSPC). Si el bit TDM = 0, el puerto TDM puede operar como otro puerto serie convencional del DSP. El registro TSPC es similar al registro SPC del puerto serie a excepción de que el bit 0 (TDM) indica la operación de este puerto.

El concepto de división de tiempo es que cada dispositivo toma una parte del tiempo disponible a compartir. El puerto TDM requiere juntar las líneas de transmisión de datos (TDX) y de recepción (TRD) en una línea simple llamada línea de datos (TDAT), para que los datos fluyan sobre esta línea. Similarmente, los relojes se unen en una línea de reloj TDM (figura 6.4). Una señal simple de "frame" es utilizada para indicar el inicio de un evento de 8 palabras TDM.

La línea TADD es manejada por un DSP en particular para un tiempo particular y determina qué dispositivo de la conexión TDM puede efectuar una recepción válida en el tiempo de slot.

El puerto TDM tiene asociados seis registros mapeados en memoria. En una lectura válida TDM, el dato es transferido del registro TRSR al registro TRCV y una interrupción de recepción es generada indicando que el registro TRCV tiene un dato válido y que puede ser leído. Todos los puertos TDM operan sincronizados por las líneas TCLK y TFRM las cuales son generadas por cada dispositivo.

6.7. Interface de Puerto Huésped Mejorado (HPI-8)

La interface de Puerto Huésped Mejorado referida como HPI8 es una versión mejorada del estándar HPI de 8 bits que se encuentran en los DSP's de la familia C54xx. El HPI8 es un puerto paralelo de 8 bits que permite conectar un dispositivo o procesador Huésped al DSP TMS320C5402.

Las características estándares que incluye el HPI8 son:

- Transferencia secuencial (con autoincremento) o transferencia de acceso aleatorio.
- Capacidad de Interrupciones tanto en el dispositivo Huésped como en el DSP.
- Pines de control de datos y direcciones para la flexibilidad de la interface.

Las características mejoradas en el HPI8 son:

- Acceso a toda la memoria RAM interna a través del bus de DMA.
- Capacidad para transferencia continua después de una pausa emulada.

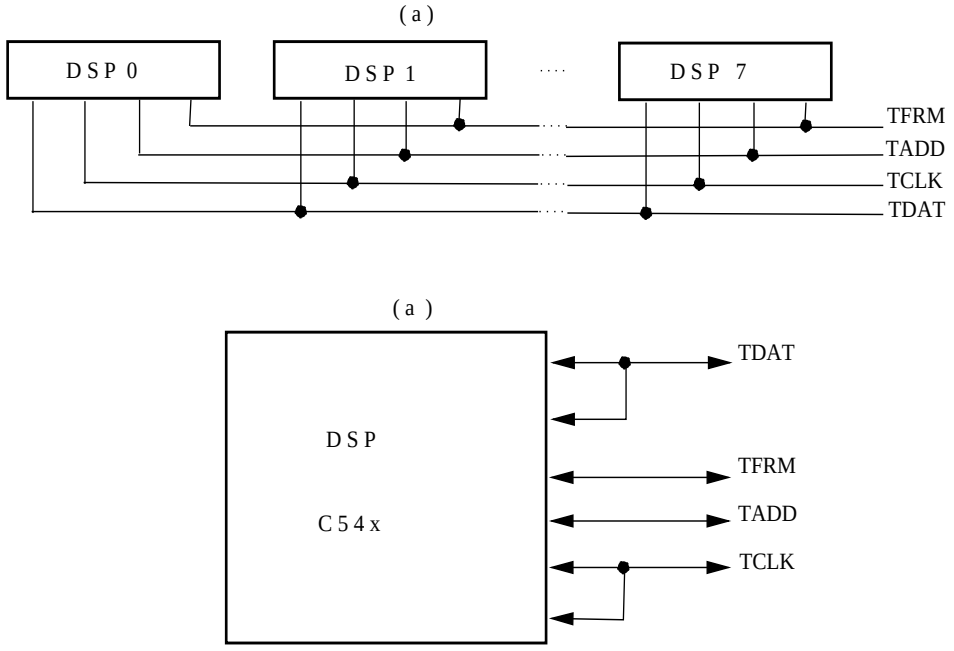


Figura 6.4: Diagrama de bloques del puerto serie TDM

El puerto HPI-8 (figura 6.5) funciona en modo esclavo y habilita al dispositivo huésped el acceso de la memoria interna del DSP. Dicha interface consiste en un bus de datos de 8 bits bidireccional y varias señalizaciones de control sincronizadas con el reloj del DSP. La comunicación con el HPI es por medio de 3 registros dedicados, el registro de direcciones del HPI8 (HPIA), el registro de datos del HPI8 (HPID) y el registro de control del HPI8 (HPIC). Los registros HPIA y HPID solo pueden ser accesados por el dispositivo huésped mientras que el registro HIPC puede ser accesado por ambos. El registro HPIC está mapeado en memoria en la dirección 2Ch. Si el dispositivo Huésped y el DSP quieren accesar a la misma localidad, el dispositivo Huésped tiene la máxima prioridad mientras que el DSP tendrá que esperar un ciclo del HPI8.

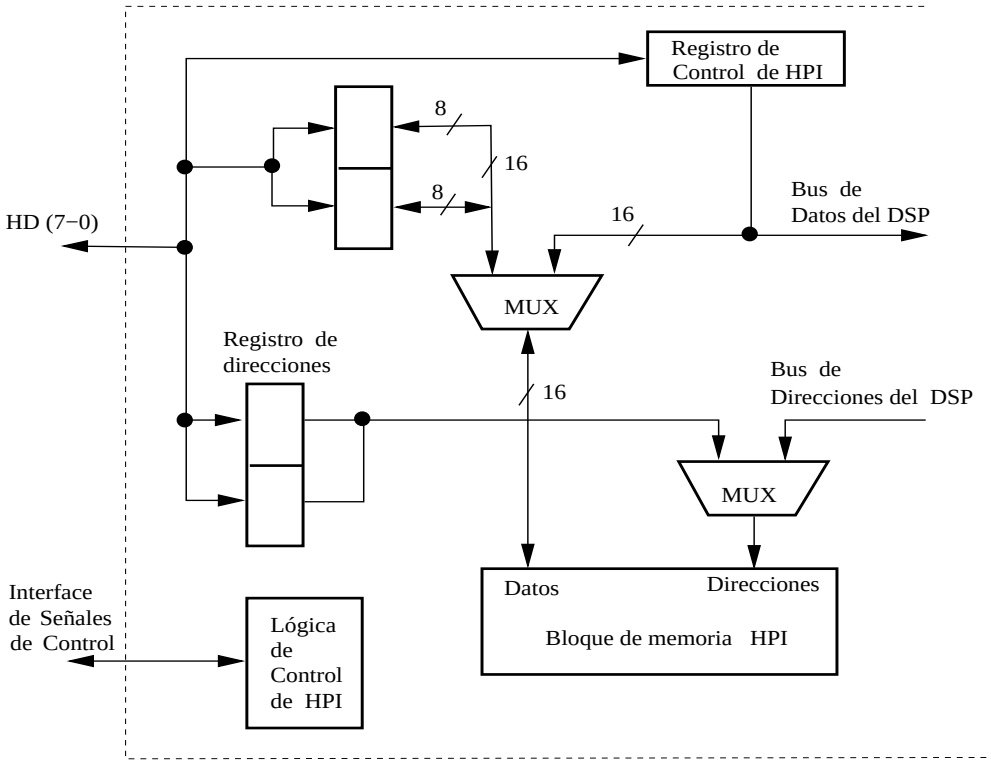


Figura 6.5: Diagrama de bloques de la Interface de Puerto Huésped HPI.

6.8. Controladores de Canales de DMA

Los controladores de canales de acceso directo a memoria transfieren datos entre los espacios del mapa de memoria (programa/dato) sin la intervención del CPU, como también a periféricos internos (por ejemplo los BPS). Estos controladores de canales DMA son independientes, por lo que permiten realizar diferentes procesos para la operación DMA.

Las características de los controladores de canales de DMA son las siguientes:

- Operaciones DMA independientes del CPU.
- Seis Canales de DMA para realizar seis diferentes procesos de transferencia independientes, para el caso del DSP TMS320C5402.
- El acceso directo a memoria tiene mayor prioridad que los accesos internos realizados por el CPU.

- Cada canal de DMA tiene prioridades independientes programables.
- Cada registro de fuente de canal y dirección destino pueden tener indices configurados a través de la memoria para cada operación de transferencia de lectura y escritura.
- Cada transferencia de lectura y escritura puede ser inicializado por eventos seleccionados.
- Una vez completa la transferencia de la mitad del bloque o de todo el bloque de memoria configurada para trabajar en DMA, cada controlador de DMA puede enviar una interrupción al CPU.
- El acceso directo a memoria puede realizar transferencia de palabras dobles, es decir, transferencia de 32 bits o dos palabras de 16 bits.

6.9. Generador de Reloj

El generador de Reloj consiste en un oscilador interno y un circuito de malla de fase enlazada (PLL), los cuales necesitan para su funcionamiento de una entrada de reloj de referencia. Dicha entrada de referencia puede ser un cristal resonador en conjunto con el oscilador interno o una fuente de reloj externa.

La entrada de reloj de referencia es dividida por un factor igual a dos (modo DIV) para generar señales de reloj o mediante el uso del circuito de PLL (modo PLL) que genere las señales de reloj multiplicando la frecuencia del reloj de referencia por un factor de escala, definiendo de esta manera, una fuente de reloj a bajas frecuencias. El PLL es un circuito adaptivo que una vez sincronizado enlaza y sigue una señal de entrada de reloj.

Cuando el circuito de PLL es inicializado, éste entra a un modo transitorio y durante esta condición, se enlaza con la señal de entrada. Una vez enlazado el circuito PLL, continua y sosteniendo la sincronización con dicha señal. Entonces, otro circuito de reloj interno permite la síntesis de nuevas frecuencias de reloj para ser usadas como reloj maestro del DSP.

El circuito PLL programado por software es controlado usando el registro de modo de reloj (CLKM) mapeado en memoria en la dirección 58h, el cual define la configuración del modulo del reloj del PLL.

6.10. Generador de Estados de Espera Programados por Software.

El generador de estados de espera programado por software concede ciclos para bus externo de hasta 14 ciclos de máquina para realizar una interface con memorias de baja velocidad y dispositivos de entrada y salida. Cuando todos los accesos externos estén configurados para cero estados de espera, el reloj interno del generador de estados de espera estará automáticamente inhabilitado, lo cual reduce significativamente el consumo de potencia del DSP.

Para controlar y configurar el número de estados de espera como también el intervalo de direcciones de memoria externa para el acceso será por medio del registro de estados de espera por software (SWWSR) y el registro de control de estados de espera por software (SWCR) mapeados en memoria en las direcciones 28h y 2Bh, respectivamente.

6.11. Banco de Interrupción Programable.

El Banco de interrupción programable puede insertar un ciclo cuando se tenga un límite de banco de memoria de acceso cruzado dentro de memoria dato o programa, así como la inserción de un ciclo cuando se presente un acceso cruzado desde memoria programa a memoria dato. Este ciclo extra previene problemas de tráfico en los buses de tal manera que dispositivos de memoria puedan liberar el uso de un bus antes de que éste empiece a ser usado o controlado por otros dispositivos. Para definir el tamaño de los estados de espera del banco de interrupción programable es por medio del registro de control del banco de interrupciones (BSCR) mapeado en memoria en la dirección 29h.

Resumen

En este capítulo sólo se presentaron las características generales de los periféricos internos que incluyen los DSPs de la familia TMS320C54xx, por lo que se recomienda para mayores referencias consultar [44] y [46]. Asimismo, en los siguientes capitulos se presentan ejemplos de aplicaciones que hacen uso extensivo de los recursos en hardware y software del DSP.

Capítulo 7

Implementación de filtros digitales en el DSP TMS320C54x

En esta parte se considera la forma de implementar las ecuaciones y estructuras de sistemas lineales y filtros digitales en la arquitectura del C54x[1]. Se hace la observación que los DSP se han desarrollado para efectuar cálculos eficientes de algoritmos matemáticos utilizados en sistemas discretos y dentro de la operación fundamental que ejecutan es la convolución que es utilizada ampliamente en los filtros digitales..

Un sistema lineal e invariante en el tiempo discreto puede ser descrito (SLITD) por una ecuación en diferencias:

$$y(n) = \sum_{m=0}^{q} b_m x(n-m) - \sum_{k=1}^{p} a_k y(n-k) \tag{7.1}$$

con función de transferencia:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_q z^{-q}}{1 + a_1 z^{-1} + \cdots + a_p z^{-p}} \tag{7.2}$$

La respuesta al impulso unitario del sistema, $h(n)$ es la transformada inversa Z de la función de transferencia $H(z)$. Un filtro digital es un sistema lineal y también se puede caracterizar por una ecuación en diferencias, su función de transferencia o su respuesta al impulso. Un filtro digital de respuesta finita al impulso (FIR), es aquel cuya respuesta al impulso es de

[1]Se asume que el lector conoce la teoría básica del diseño de filtros digitales y el cálculo de los coeficientes. Para mayor información de diseño de filtros consultar [14], [24] y [19].

duración finita. De la ecuación general (7.1) los coeficientes a_k son cero para $k \geq 1$, entonces, un filtro FIR queda descrito por:

$$y(n) = \sum_{m=0}^{q} b_m x(n-m) \tag{7.3}$$

y su función de transferencia:

$$H(z) = b_0 + b_1 z^{-1} + \cdots + b_q z^{-q} \tag{7.4}$$

donde se puede observar que la salida $y(n)$ del sistema FIR es una suma ponderada de los coeficientes b_m por la entrada actual $x(n)$ y las muestras retardadas, además un filtro FIR se caracteriza por ser siempre estable y de fase lineal. La sumatoria que permite efectuar un filtro FIR es la operación de convolución de los coeficientes por una ventana temporal de una señal como se observa en la figura 7.1.

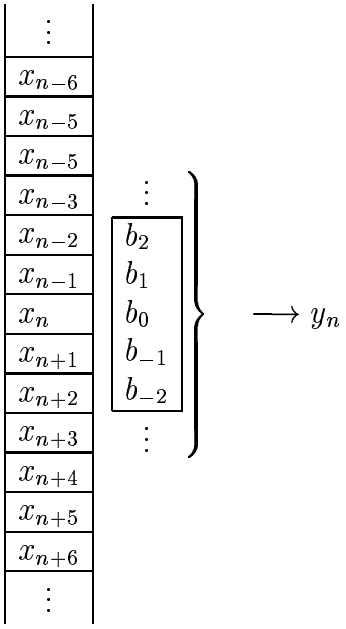


Figura 7.1: Convolución de una señal con los coeficientes del filtro.

7.1. Implementación de líneas de retardo

En esencia un filtro FIR es una suma de productos de los coeficientes b_k por las muestras retrasadas $x(n-i)$ de la señal de entrada. Para la implementación de estos filtros eficientemente es necesario tener la posibilidad de generar una línea de retardos.

7.1.1. Buffer lineal

La forma más simple de implementar una línea de retardo en un microprocesador es vía un buffer lineal, donde un filtro de $N-1$ retardos opera sobre las N muestras más recientes. Cada vez que se efectúa la sumatoria del filtro FIR se adquiere un nuevo dato y es agregado a la parte superior del buffer y el dato inferior es descartado. Es decir, que una vez calculada la salida, un dato es movido a la localización siguiente para realizar el retardo.

En un procesador convencional una línea de retardo involucraría el acceso a dos localidades de memorias continuas y un almacenamiento temporal del dato (por lo menos tres instrucciones por cada dato desplazado).

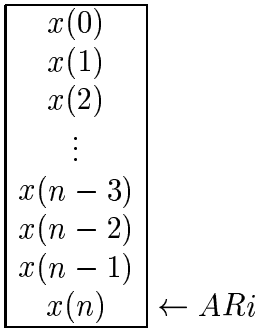


Figura 7.2: Buffer lineal.

El DSP TMS320C54x puede generar eficientemente las líneas de retardo con instrucciones orientadas para estos fines. La instrucción DELAY efectúa una copia del dato direccionado a la siguiente localidad en memoria dato sin afectar al dato direccionado, esto es válido sólo en memoria interna.

```
DELAY  Smem
```

El proceso anterior puede ser más eficiente si se realiza en instrucciones que contengan el movimiento de datos implícitamente, por ejemplo si se combina simultáneamente con un carga de un dato al registro T y la acumulación del producto anterior, todo esto lo efectúa la instrucción LTD:

```
LTD    Smem  ; Carga el registro T con el dato en Smem, copia el
             ; contenido de Smem a localidad Smem+1
```

Las líneas de retardo para la implementación de filtros son aún más eficientes utilizando la instrucción MACD. Se hace notar que el movimiento de bloque sólo funciona para memoria RAM interna.

```
MACD   Smem,pmad,src ; multiplica el valor Smem en memoria dato por
                     ; un dato pmad en memoria programa y suma este
                     ; producto a src y lo almacena en src. El dato
                     ; direccionado por Smem es copiado en Smem+1
                     ; src puede ser el acumulador A o B
```

Una implementación muy eficaz de un filtro FIR también se puede realizar utilizando el concepto de buffer circular, en el que para un buffer lineal se determina una localidad inicio en memoria, una longitud del buffer y un registro auxiliar ARi que trabajará en forma circular. Si el ARi correspondiente se incrementa en "uno" dentro del buffer circular, al llegar a la última localidad de buffer automáticamente regresará al inicio del buffer después en el post-incremento.

7.2. Implementación y estructuras de un filtro FIR

Los filtros de respuesta finita al impulso (FIR) se caracterizan porque su salida $y(n)$ sólo depende de la entrada actual $x(n)$ y N-1 retardos de $x(n)$ multiplicado por los coeficientes del filtro, además sólo tiene polos múltiples en el origen del plano Z y N-1 ceros por lo que se considera que siempre son estables.

7.2.1. Estructuras de los filtros FIR

A nivel de implementación de un sistema discreto en una arquitectura digital es necesario hacer ciertas re-adaptaciones de las ecuaciones teóricas para su manipulación sin afectar el desempeño del sistema. La estructura de un filtro digital se entiende como la forma de

implementar la ecuación en diferencias del filtro, o la función de transferencia, o cómo se efectúan las operaciones matemáticas.

Como la función de transferencia de un filtro FIR tiene la forma:

$$H(z) = h_0 + h_1 z^{-1} + \cdots + h_{N-1} z^{-N+1} = \sum_{i=0}^{N-1} h(i) z^{-i} \tag{7.5}$$

y su respuesta al impulso:

$$h(n) = \begin{cases} h_i & 0 \leq i \leq N-1 \\ 0 & \text{otro i} \end{cases} \tag{7.6}$$

y la ecuación en diferencias es:

$$y(n) = h_0 x(n) + h_1 x(1) + \cdots + h_{N-1} x(n-N+1) = \sum_{i=0}^{N-1} h(i) x(n-i) \tag{7.7}$$

la cual es una convolución lineal de la entrada $x(n)$ con la respuesta al impulso $h(n)$ del filtro FIR, N es la longitud del filtro y es igual al número de coeficientes del filtro. Dependiendo como se implemente la ecuación (7.7) se pueden tener varias estructuras o formas como se mencionan a continuación.

Forma directa

En esta forma la ecuación (7.7) se implementa directamente utilizando los bloques básicos de un sistema discreto, como se muestra en el diagrama de bloques en la figura 7.3

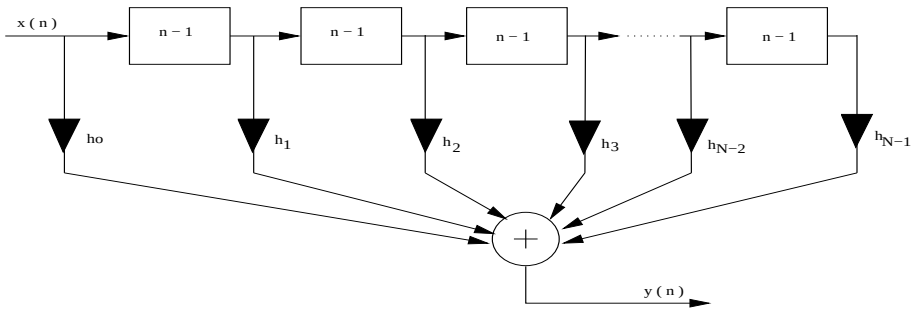


Figura 7.3: Forma directa de un filtro FIR.

La siguiente subrutina se utiliza para la realización de un Filtro FIR, para su funcionamiento adecuado tanto los coeficientes del filtro como la entrada al buffer circular se deben escribirse en orden inverso.

```
* Subrutina para la implementación de un filtro FIR
* utilizando la instrucción MAC y Buffer circular
* Acumulador A = y(n) = h(n)*x(n-i) para i = 0,1,2,3,4,....N-1
* AR6 apunta a dato de entrada x(n)
* AR7 apunta a localidad de salida y(n)
* AR4 apunta a datos x(n) en el buffer circular x(n-N+1)
* AR5 apunta a los coeficientes del filtro h(N-1)
      LD #FIR,DP
      STM #LFIR-1,BRC ; Número de veces del ciclo FIR
      RPTB FIR_LOOP-1
         STM #K,BK  ; Tamaño del buffer circular
         LD *AR6+,A ;
*Carga valor de entrada Convolución
         STL A,*AR4+% ; Reemplaza muestra vieja con nueva
         RPTZ A,#K
          MAC *AR4+%,*AR5+%,A
         STH A,*AR7+ ; Salida y(n)
 FIR_LOOP
      RET
```

Forma en Cascada

La función $H(z)$ del sistema se puede descomponer en factores de segundo y primer orden dependiendo si la longitud N es par o impar.

$$H(z) = h_0 + h_1 z^{-1} + \cdots + h_{N-1} z^{-N+1} = \sum_{i=0}^{N-1} h(i) z^{-i} \tag{7.8}$$

$$H(z) = h_0 (1 + \frac{h_1}{h_0} z^{-1} + \cdots + \frac{h_{N-1}}{h_0} z^{-N+1}) \tag{7.9}$$

$$H(z) = h_0 \prod_{k=1}^{K} (1 + B_{k,1} z^{-1} + B_{k,2} z^{-2}) \tag{7.10}$$

$$\tag{7.11}$$

Donde $K = \frac{N-1}{2}$ si N es impar, es decir se tienen K factores de segundo orden. Si N es par, $K = \frac{N}{2} - 1$, se tiene K factores de segundo orden y uno de primer orden.
$B_{k,1}$ y $B_{k,2}$ son números reales que representa los coeficientes de las secciones de segundo orden. La forma en cascada un filtro FIR se muestra en la figura 7.4.

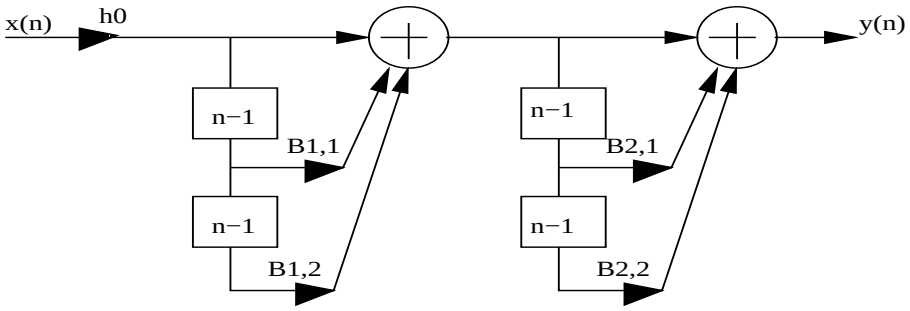


Figura 7.4: Forma Cascada de un filtro FIR.

Forma de Fase lineal

Esta forma explota la simetría de $h(n)$ de un filtro FIR y reduce el número de multiplicadores a la mitad. Para filtros selectivos en frecuencia, es deseable tener una respuesta en fase lineal, es decir, se desea que:

$$\angle H(e^{j\omega}) = \beta - \alpha\omega; \qquad -\pi < \omega < \pi \tag{7.12}$$

Para filtros causales con respuesta finita al impulso sobre el intervalo $[0, N-1]$, la condición de fase lineal ecuación (7.12) impone condiciones de simetría en la respuesta al impulso $h(n)$:

$$h(n) = h(N-1-n); \qquad \beta = 0; \qquad 0 \leq n \leq N-1 \qquad (\text{ simetría }) \tag{7.13}$$

$$h(n) = -h(N-1-n); \quad \beta = \pm\pi/2; \qquad 0 \leq n \leq N-1 \qquad (\text{asimetría}) \tag{7.14}$$

Cuando $H(\omega)$ cambia de signo de positivo a negativo (o viceversa), la fase sufre un cambio abrupto de π radianes. Si los cambios de fase ocurren fuera de la banda de paso del filtro no tiene importancia [24]. Para la respuesta al impulso con simetría:

$$\begin{aligned} y(n) &= h_0x(n) + h_1x(n-1) + h_2x(n-2) + \cdots + h_{N-2}x(n-N+2) + h_{N-1}x(n-N+1) \\ y(n) &= h_0\Big(x(n) + x(n-N+1)\Big) + h_1\Big(x(n-1) + x(n-N+2)\Big) + \cdots \end{aligned} \tag{7.15}$$

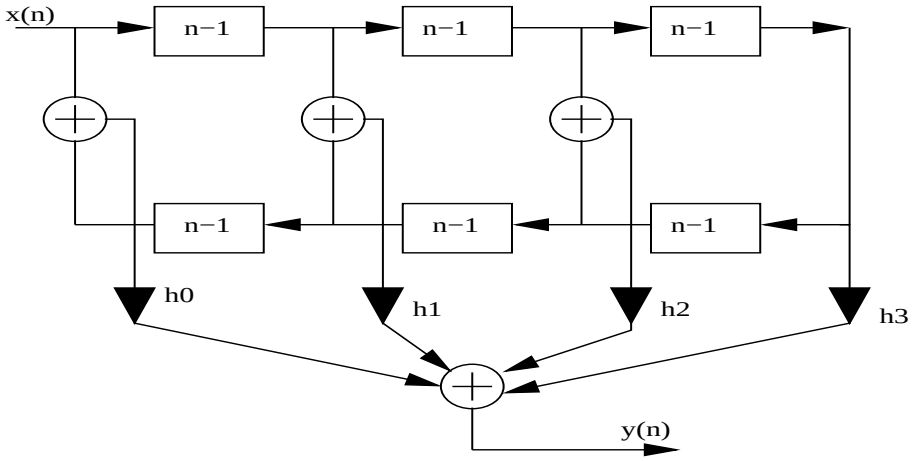


Figura 7.5: Forma Fase lineal de un filtro FIR.

Como se observa de la última ecuación y de la figura 7.5, en esta estructura se requiere 50 % menos multiplicadores que en la forma directa, es decir, que se utilizan nada más la mitad de los coeficientes, lo que puede ser útil en el ahorro de memoria en la implementación en una arquitectura DSP. El C54x facilita la implementación de un filtro FIR de fase lineal utilizando direccionamiento circular y la instrucción FIRS. Esta instrucción multiplica el acumulador A parte alta con el dato direccionado en memoria pma y suma el resultado con el acumulador B, al mismo tiempo suma los dos operandos Xmem e Ymem direccionados en memoria dato, corre el resultado 16 bits a la izquierda y carga el resultado en el acumulador A, en la próxima iteración pma es incrementada en uno, una vez establecido el pipeline, la instrucción se convierte de un ciclo de instrucción. Los filtros de fase lineal son ampliamente utilizados en procesamiento de voz para evitar distorsión por fase. Para implementar la ecuación (7.15), se deben utilizar dos apuntadores ARi, uno que inicie en la localidad $x(n)$ y el otro en la localidad $x(n-N+1)$ y que en un ciclo se vayan acercando hacia $x(n-N/2)$. Para efectuar retardos sobre la señal de entrada $x(n)$, se debe de implementarse dos buffers circulares de longitud N/2.

7.3. Filtros de respuesta infinita al impulso (IIR)

Una característica de un filtro IIR que lo diferencia de un filtro FIR es que realimenta la señal de salida y que puede llegar a ser inestable. En un filtro IIR los valores de los coeficientes a_m son diferentes de cero y también pueden existir todos los coeficientes b_m. La

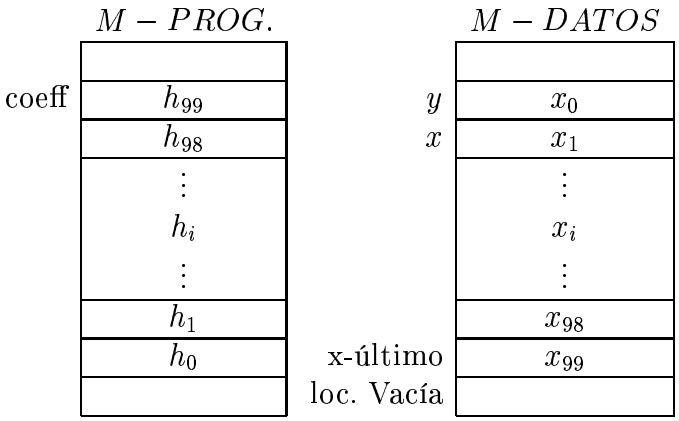


Figura 7.6: Filtro FIR usando MACD.

implementación de un filtro IIR se puede ver como la convolución de los coeficientes b_m con la señal de entrada menos la convolución de la señal de salida retardada con los coeficientes a_m, esto se aprecia en la figura 7.7.

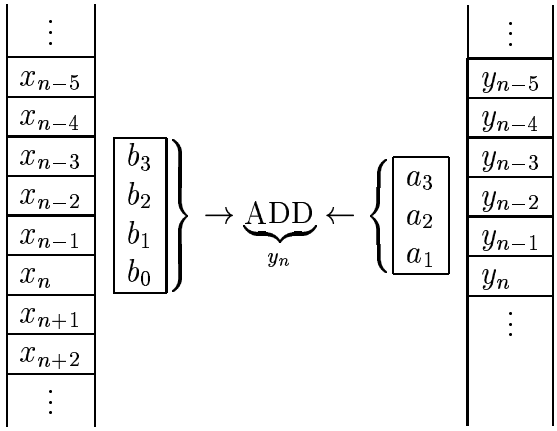


Figura 7.7: Líneas de retardo de un filtro IIR.

7.3.1. Estructuras de filtros IIR

De la ecuación en diferencias general existen dos formas de implementar un filtro IIR: la forma directa I, donde existen $2p$ (si $p = q$) elementos de retardo mientras que el orden del filtro es p. En la segunda forma o canónica, el número de retardos es igual al orden del filtro, ver figuras 7.8 y 7.9.

7.3.2. Separación en estructuras de segundo orden

De la teoría del procesamiento digital de señales [14], [19], [24], [25], [26], para la implementación de un filtro digital IIR es conveniente implementarlo en estructuras de segundo orden ("biquad") para evitar la acumulación de errores cuando se opera en aritmética de punto fijo.

7.3.3. Estructuras de filtros digitales IIR

La ecuación en diferencias que se utiliza para implementar un filtro IIR es:

$$y(n) = \sum_{i=0}^{q} b(i)x(n-i) - \sum_{i=1}^{p} a(i)y(n-i) \tag{7.16}$$

Aplicando la Transformada Z (TZ) a esta ecuación, se obtiene la función de transferencia H(z) del filtro:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{B(z)}{A(z)} = \frac{\sum_{i=0}^{q} b(i)z^{-i}}{\sum_{i=0}^{p} a(i)z^{-i}} = \frac{b_0 + b_1 z^{-1} + ... + b_q z^{-q}}{1 + a_1 z^{-1} + ... + a_p z^{-p}} \quad , \quad a_0 = 1 \tag{7.17}$$

Donde p es el número de polos, q es el número de ceros de $H(z)$, a_i y b_i son los coeficientes del filtro. Dependiendo como se traten las dos ecuaciones anteriores se pueden obtener tres formas o estructuras más comunes de implementación[2]:

- Forma directa.
- Forma cascada.
- Forma paralela.

[2]Otras estructuras como Lattice y Lattice-Ladder [24]

Filtro IIR Forma directa

En esta forma la ecuación (7.16) es implementada directamente. En este filtro existen dos partes denominadas movimiento promedio (MA) y la parte regresiva (AR). A la vez esta implementación conduce a dos formas de implementación: forma directa I y forma directa II.

Forma directa I no canónica

Si la ecuación (7.16) se desarrolla, entonces se tiene:

$$y(n) = b_0x(n) + b_1x(1) + ...b_qx(n-q) - a_1y(n-1) - a_2x(n-2) - ... - a_py(n-p) \quad (7.18)$$

Se observa que existen dos líneas de retardo, una para la señal de entrada $x(n)$ y la otra para la señal de salida $y(n)$, es decir, que se tendrían $p + q$ bloques de retardo. La forma directa I se muestra en la figura 7.8.

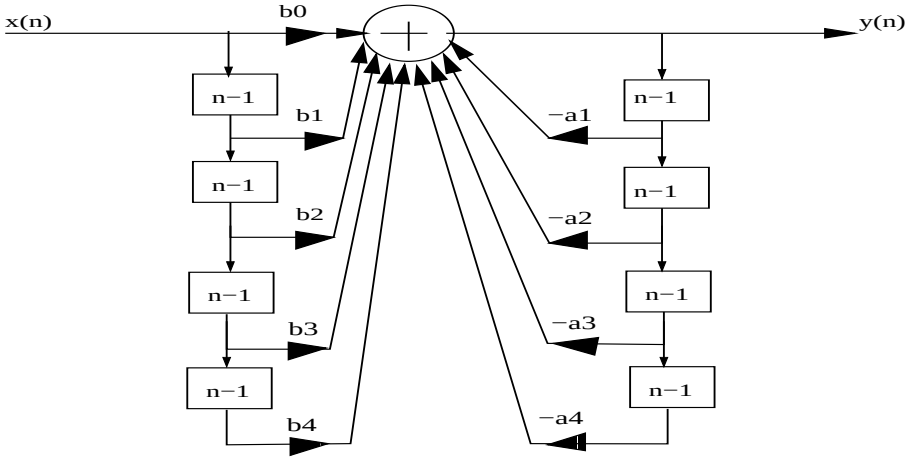


Figura 7.8: Filtro IIR, forma directa I no canónica.

Forma directa II o canónica

Las líneas de retardo se pueden convertir a una sola, lo que conduce a la forma directa II de un filtro IIR o *estructura canónica*, es decir, que a nivel de hardware se ahorrarían localidades de memoria.

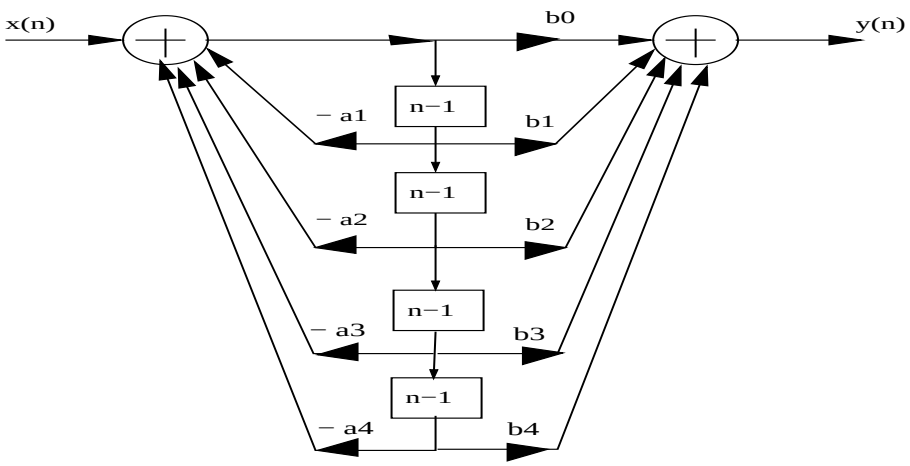


Figura 7.9: Filtro IIR, forma directa II canónica.

Filtro IIR Forma cascada

En esta forma, la ecuación (7.10) es factorizada en secciones de segundo orden, llamadas secciones *biquads* [33]. La función de transferencia del sistema es representada como el producto de funciones biquads. Cada función biquad es implementada en forma directa o canónica y la función completa del sistema como secciones *biquad* en cascada. Es decir, que se factorizan el numerador y denominador de la ecuación (7.17) para obtener factores de segundo orden con coeficientes reales, teniendo la ecuación a implementar:

$$H(z) = b_0 \prod_{k=1}^{K} \frac{1 + B_{k,1} z^{-1} + B_{k,2} z^{-2}}{1 + A_{k,1} z^{-1} + A_{k,2} z^{-2}} \tag{7.19}$$

Donde $K = \frac{N}{2}$, y $B_{k,1}$, $B_{k,2}$, $A_{k,1}$ y $A_{k,2}$ son números reales que representan los coeficientes de las secciones de segundo orden. Cada sección biquad se implementa en forma directa o canónica, y la entrada de la sección k-ésima es la salida de la sección $(k-1)$. Esta forma de implementación permite disminuir la acumulación de errores numéricos cuando los filtros se programan en aritmética de punto fijo.

Forma paralela

Es similar a la forma cascada, pero con la diferencia que la ecuación (7.17) se representa como una suma de fracciones parciales de segundo orden. De nuevo cada sección es implemen-

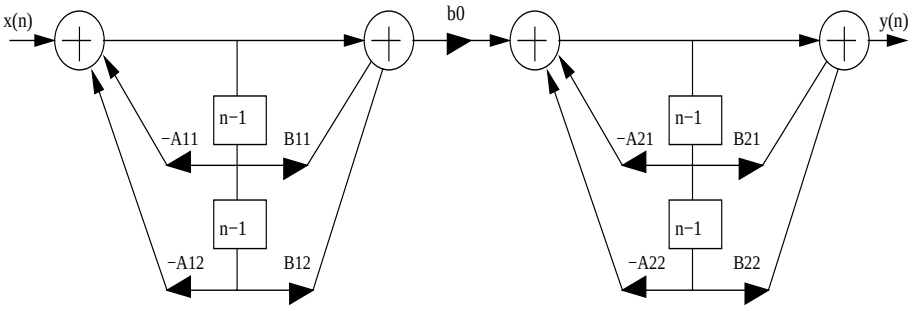


Figura 7.10: Filtro IIR en cascada.

tada en forma directa y la función total del sistema como una red paralela de las secciones. Desarrollando la ecuación (7.17) en fracciones parciales:

$$H(z) = \frac{B_0}{1 + a_1 z^{-1}} + \sum_{k=1}^{K} \frac{B_{k,0} + B_{k,1} z^{-1}}{1 + A_{k,1} z^{-1} + A_{k,2} z^{-2}} \tag{7.20}$$

Donde K es un entero dado por $K = \frac{N}{2}$, si K es impar se implementa el primer término, de lo contrario si K es par se implementan sólo términos de segundo orden. $B_{k,0}$, $B_{k,1}$, $A_{k,1}$ y $A_{k,2}$ son números reales que representan los coeficientes de las secciones de segundo orden. La entrada $x(n)$ del filtro está disponible para toda sección biquad, la salida de cada sección es sumada para formar la salida $y(n)$ del filtro. Una ilustración de esta estructura está en la figura 7.11.

7.3.4. Filtro digital de IIR de segundo orden

Un filtro IIR se puede analizar en dos secciones, una de realimentación y una similar a un filtro FIR con entrada X_0, como se ve en la figura 7.12.

La señal X_0 es una señal que es retardada y alimenta al filtro FIR y a la vez es realimentada al primer sumador. En seguida se ilustra un programa en lenguaje ensamblador correspondiente a la figura 7.12 con sus respectivas variables.

```
*   Programa que ejecuta un filtro IIR de segundo orden
*   para la forma canónica  de segundo orden (biquad)
*
```

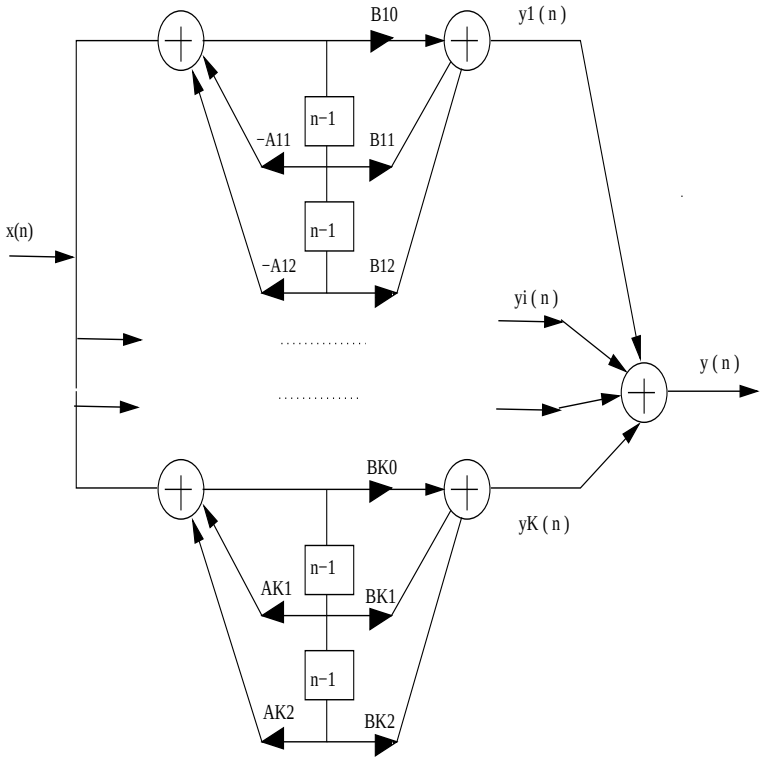


Figura 7.11: Filtro IIR en paralelo con K par.

```
*    AR4 apunta a los coeficientes del filtro, A2, A1, B2, B1 y B0
*    AR5 apunta a los datos del punto w(n-2)
*    AR6 apunta a los datos de entrada x(n)
*    AR7 apunta a localidad de salida y(n)

          LD   *AR6+,0,A          ; A = valor de entrada x(n)
          MAC   *AR4+,*AR5-,A     ; A = x(n) + w(n-2)*A2
          MAC   *AR4+,*AR5-,A     ; A = x(n) + w(n-2)*A2 + w(n-1)*A1
          STH   A,*AR5+           ; w(n)=A=x(n) + w(n-2)*A2 + w(n-1)*A1
          MAR   *AR5+
          MPY   *AR4+,*AR5-,A     ; A = w(n-2)*B2
          MAC   *AR4+,*AR5,A      ; A = w(n-2)*B2 + w(n-1)*B1
          DELAY  *AR5-            ; w(n-2)=w(n-1)
```

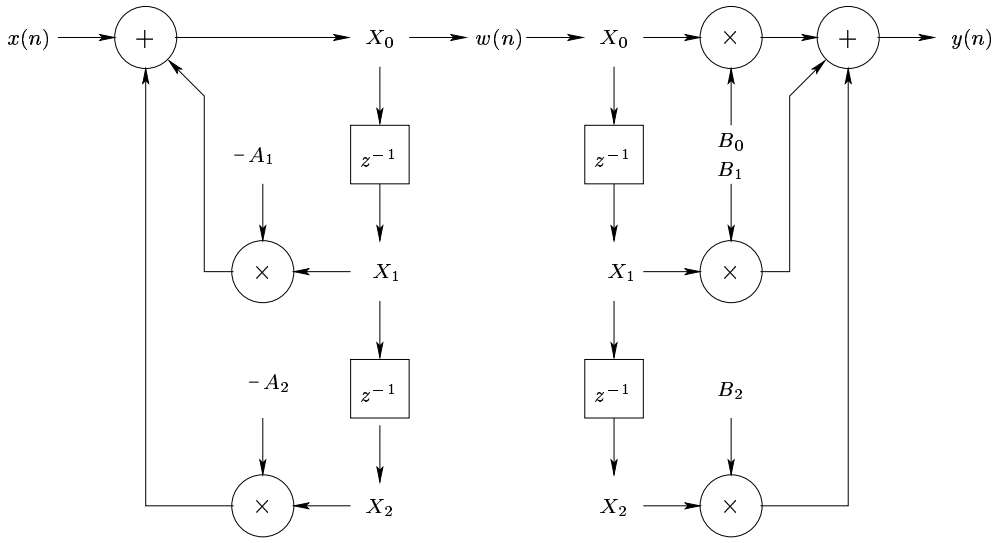


Figura 7.12: Implementación de un filtro IIR de segundo orden.

```
MAC   *AR4+,*AR5,A      ; A = w(n-2)*B2 + w(n-1)*B1 + w(n)*B0
DELAY  *AR5-            ; w(n-1)=w(n)
STH   *AR7+             ; Salida y(n)
```

Filtro paso bajas en tiempo real

En este ejemplo se muestra un programa en lenguaje ensamblador de un filtro pasobajas de respuesta finita FIR que tiene 80 coeficientes operando en tiempo real. Este filtro tal como está diseñado se puede aplicar a música, voz y otras señales que pueden adquirirse a través de la tarjeta DSK del DSP C54x.

Entrada: Voz o música, utilice un micrófono en el conector de entrada del Starter-kit.

Salida: Bocina o analizador de espectro.

Filtro FIR: Para la implantación del filtro paso bajas FIR se utiliza una ventana Blackman y 80 coeficientes. La frecuencia de corte del filtro paso bajas es de 1000 Hz.

Archivos:

- **LOWPASS.asm**. Código fuente del filtro paso bajas
- **LP_COEFF.asm**. Contiene los valores de los coeficientes del filtro
- **LP_AC01.asm**. Este archivo inicializa el convertidor AIC.
- **LP_VECS.asm**. Contiene la tabla de vectores de interrupción para lowpass.asm.

Operación de los archivos: En la primera parte del programa se realiza la de inicialización de algunas. variables, constantes y localidades de memoria. El archivo LP_AC01 inicializa la interface analógica. El archivo LP_COEFF contiene una referencia para la variable coeff y el archivo LP_VECS contiene la tabla de los vectores de interrupción. La etapa final obtiene las muestras del AIC (convertidor A/D y D/A) y las pasa a través del filtro paso baja.

AIC: Parámetros de trabajo. La frecuencia de muestreo es de 9259 Hz. Por lo tanto, la frecuencia de Nyquist es 4629.5 Hz. Se supone, un reloj maestro MCLK=10.0 MHz.

Programa principal

```
*************************************************************************
*                                                                       *
*    NOMBRE DE ARCHIVO: LOWPASS.asm                                     *
*                                                                       *
*   Este programa fue escrito para el TMS320C54x. Es un filtro          *
*   paso bajas de 80 coeficientes. El filtro puede ser probado usando   *
*   el generador de ruido. Frecuencia de corte para el filtro           *
*   es:                                                                 *
*                         Fcorte = 1000*Fm/9.6k                         *
*                                                                       *
*   donde Fm es la frecuencia de muestreo en Hz. La frecuencia          *
*   de muestreo Fm puede ser modificada mediante AC01init.asm           *
*                                                                       *
*************************************************************************

 ;  GENERADOR DE RUIDO RANDOM
 ;  -> Generador de ruido incluido para permitir al usuario oír
 ;     la diferencia en la salida cuando esta filtrada o verla en un
 ;     analizador de espectros.

          .title   "Filtro paso bajas"
          .mmregs
          .width   80
          .length  55

       .setsect ".text",0x1800,0   ; Estas directivas en ensamblador
       .setsect ".data",0x0200,1   ; especifican la dirección de las
       .setsect "vectors",0x0180,0 ; diferentes secciones del código.

       .sect "vectors"     ; la tabla de los vectores de interrupción
       .copy "lp_vecs.asm" ; se localizará en la dirección  0x0180
       .sect ".data"       ; la sección de datos se localizará en 0x200

seed     .word    07e6dh  ; semilla para la variable aleatoria
temp     .word    0
```

```
XN       .word    0,0,0,0,0,0,0,0,0,0  ; 80 posiciones para las  80
XN1      .word    0,0,0,0,0,0,0,0,0,0  ; etapas del filtro.
XN2      .word    0,0,0,0,0,0,0,0,0,0
XN3      .word    0,0,0,0,0,0,0,0,0,0
XN4      .word    0,0,0,0,0,0,0,0,0,0
XN5      .word    0,0,0,0,0,0,0,0,0,0
XN6      .word    0,0,0,0,0,0,0,0,0,0
XN7      .word    0,0,0,0,0,0,0,0,0
XNLAST   .word    0
OUTPUT   .word0


         .sect".text"
         .copy "lp_coeff.asm"
         .copy "lp_ac01.asm"

start: intm = 1          ; deshabilita  interrupciones mascarables
       dcall   AC01INIT  ; inicializa la interface analógica
       DP = #0
       nop
       pmst = #01a0h ; remapeo de vectores de interrupción loc. a 180h
       sp = #0ffah   ; apuntador de pila localizado en el kernel de
                     ; comunicación
       imr = #240h   ; desenmascara TDM RINT y HPIINT  (a la PC)
       intm = 0      ; habilita  interrupciones mascarables
WAIT:  goto   WAIT   ; espera recibir interrupciones


;----- Subrutina de atención de interrupción de recepción de dato ----

receive: DP = #seed    ; Apunta a la página  de memoria  XN
                       ; que fue definida para datos de entrada

;--------- Generador de ruido aleatorio    ------------------
         a = @seed << 1
         a = @seed ^ a
         @temp = a << 2
```

```
         a = @temp ^ a
         a = #8000h & a
         a = a + @seed << 16
         @seed = hi(a) << 1
         a = @seed << 11
         a = a & #0fffch << 15
         repeat(#12)
         a = a <<C -1

;----- Obtiene la  muestra actual y realiza  el filtro paso bajas-------

      b = DRR1                ; Almacena en el acumulador B el dato recibido
                              ;  del AIC
      @XN = A << 0            ; Se carga el valor obtenido de la
                              ; variable XN
      AR0 = #XNLAST           ; En AR0 se almacena la dirección
                              ; de ultimo elemento de retraso!
      A = #0                  ; Se limpia el acumulador  A
      repeat(#79)             ; Se repite la instrucción siguiente 80 veces.
         macd(*AR0-,h0,A)     ; Se calcula la salida del filtro FIR.

      @OUTPUT = hi(A) << 0    ; Se almacena el valor de salida
                              ; en la variable OUTPUT.
      A = @OUTPUT << 0        ; OUTPUT ==>Acumulador A
      A = #0FFFCh & A         ; Limpia dos bits  LSB's para AIC
      DXR1 = A                ; El valor se carga a registro de transmisión
      return_enable           ; Habilita interrupciones  y retorno
                              ; de interrupción.

;--------  Subrutina de  interrupción de transmisión
------------------ transmit:
        return_enable          ; habilita interrupciones  y retorno
                               ; de interrupción de transmisión
        nop
        nop
        nop
       .end
```

Programa par la inicialización y configuración del codec

```
;***********************************************************************
; Archivo: LP_AC01.ASM -> Inicialización de  AC01 *
;***********************************************************************

         .width   80
         .length  55
         .title "PROGRAMA DE INICIALIZACIÓN DEL CONVERTIDOR AC01"
         .mmregs

*************************************************************************
* Algunos registros del AC01 pueden ser  inicializados usando una constante
* condicional en ensamblador.  Dándole el valor apropiado a la constante
* REGISTER, el ensamblador incluye o ignora la inicialización del registro.
* La constante REGISTER  se fija  para incluir los siguientes registros
* del AC01:
*
* REGISTRO (número binario)
*
* 0000 0000 0000 0001-> inicializa el registro 1  (Registro A)
* 0000 0000 0000 0010-> inicializa el Registro 2  (Registro B)
* 0000 0000 0000 0100-> inicializa el registro 3  (Registro A')
* 0000 0000 0000 1000-> inicializa el registro 4  (Selector de ganancia)
* 0000 0000 0001 0000-> inicializa el registro 5  (Configuración analógica)
* 0000 0000 0010 0000-> inicializa el registro 6  (Configuración digital)
* 0000 0000 0100 0000-> inicializa el registro 7  (Sincronización del tramo)
* 0000 0000 1000 0000-> inicializa el registro 8  (Sincronización de número)
*
* Cualquier combinación puede ser inicializada agregando el número binario
* a la constante REGISTER. Por ejemplo, para inicializar los registros 4 y 5,
* REGISTER=18h. Cuando se llama a la subrutina esta cargará los valores REG4
* y REG5  dentro de los registros del AC01.
*
* El registro 4 siempre es inicializado para tener una ganancia de entrada de
* 6 dB. Esto establece una escala completa de 3 Vpp de entrada en la
* configuración del  AC01.
```

```
*
REGISTER .set   1bh
REG1     .set   124h
REG2     .set   20fh
REG3     .set   300h
REG4     .set   409h
REG5     .set   503h
REG6     .set   600h
REG7     .set   700h
REG8     .set   801h



AC01INIT:
        xf = 0              ; reinicializa el AC012set ac01
        intm = 1            ; deshabilita rutinas de interrupción
        tcr = #10h          ; para el temporizador
        imr = #280h         ; desenmascara  TXINT y HPIINT
        tspc = #0008h       ; detiene el puerto serial TDM
        tdxr = #0h          ; envía 0 como primer palabra a xmit
        tspc = #00c8h       ; reset and start TDM puerto serial
        xf = 1
;   ------------ Inicialización de los registros -------------

        .eval REGISTER & 1h,SELECT  ; si REG1, entonces incluye este código
        .if SELECT = 1h
        a = #REG1                   ; carga Acc A con el valor de REG1
        call REQ2                   ; llama la subrutina  REQ2
        .endif
        .eval REGISTER & 2h,SELECT  ; si REG2, entonces incluye este código
        .if SELECT = 2h
        a = #REG2
        call REQ2

        .endif

        .eval REGISTER & 4h,SELECT  ;si REG3, entonces incluye este código
```

```
        .if SELECT = 4h
        a = #REG3
    call REQ2
        .endif

        .eval REGISTER & 8h,SELECT  ;si REG4, entonces incluye este código
        .if SELECT = 8h
        a = #REG4
        call REQ2
        .endif

        .eval REGISTER & 10h,SELECT ;si REG5, entonces incluye este código
        .if SELECT = 10h
        a = #REG5
        call REQ2
        .endif

        .eval REGISTER & 20h,SELECT ;si REG6, entonces incluye este código
        .if SELECT = 20h
        a = #REG6
        call REQ2
        .endif

        .eval REGISTER & 40h,SELECT ;si REG7, entonces incluye este código
        .if SELECT = 40h
        a = #REG7
        call REQ2
        .endif

        .eval REGISTER & 80h,SELECT ;si REG8, entonces incluye este código
        .if SELECT = 80h
        a = #REG8
        call REQ2
        .endif
        return

REQ2
```

```
ifr = #080h       ; limpia banderas del registro IFR
tdxr = #03h

idle(1)           ; espera interrupción  xmit
tdxr = a          ; envía el valor del registro al puerto
ifr = #080        ; limpia banderas de IFR
idle(1)           ; espera interrupción  xmit
tdxr = #0h        ; envía un estado neutral an caso de la
                  ; ultima int
ifr = #080h       ; limpia banderas de IFR
idle(1)           ; espera un estado neutral para xmit
return            ; regreso de  subrutina
```

Definición de los vectores de Interrupción

```
*************************************************************************
 ; Archivo: LP_VECS.ASM -> Vectores de interrupción para el C54x
*************************************************************************
 ; Los  vectores de  esta tabla  pueden ser configurados  para procesar
 ; interrupciones   de  software   externas   e internas. El  Depurador
 ; del DSK plus usa 4 vectores de  interrupción. Estos son RESET, TRAP2
 ; INT2 y HPIINT. NO MODIFICAR ESTOS VECTORES  SI SE  USARA EL DEPURADOR
 ;
 ; Todas las direcciones no mencionadas son libres de usarse. Al programar
 ; siempre el bit HPIINT es no enmascarable  (IMR=200h)  para  permitir al
 ; kernel de comunicación y la interface con la PC. La INT2 debe
 ; ser normalmente enmascarada (IMR(bit2)=0) de esta manera el DSP no se
 ; interrumpirá sólo con HINT. HINT está unido a INT2 externamente.
 ;
         .title "Tabla de vectores"
         .mmregs
         .width   80
         .length  55

reset    goto  #80h ; 00, RESET, No modificar si se usa el depurador
         nop
         nop
nmi      return_enable  ; 04, externa no enmascarable
         nop
         nop
         nop
trap2    goto  #88h     ; 08, trap2, No modificar si se usa el depurador
         nop
         nop
         .space 52*16   ; 0C-3F, vector de software 18-30
int0     return_enable  ; 40, externa int0
         nop
         nop
         nop
int1     return_enable  ; 44, externa int1
```

```
         nop
         nop
         nop
int2
         return_enable  ; 48, externa int2
         nop
         nop
         nop
tint     return_enable  ; 4C, reloj interno TINT
         nop
         nop
         nop
brint    return_enable  ; 50, recepción RINT
         nop
         nop
         nop
bxint    return_enable  ; 54, transmisión XINT
         nop
         nop
         nop
trint    dgoto receive  ; 58, interrupción de recepción TDM
         nop
         nop
txint    return_enable  ; 5C, interrupción de transmisión TDM
         nop
         nop
         nop
int3     return_enable  ; 60, externa interrupción int3
         nop
         nop
         nop
hpiint   goto  #0e4h   ; 64, HPIint, No modificar si se usa el depurador
         nop
         nop
        .space  24*16   ; 68-7F, área reservada
```

Archivo de definición de los coeficientes para el filtro

```
*************************************************************************
* Archivo: LP_COEFF.ASM  -Coeficientes del filtro paso bajas.           *
*************************************************************************
; Coeficientes del filtro en formato Q15, fcut=1K.

h0  .word         0
h1  .word      -157
h2  .word      -261
h3  .word      -268
h4  .word      -170
h5  .word         0
h6  .word       180
h7  .word       301
h8  .word       310
h9  .word       198
h10 .word         0
h11 .word      -211
h12 .word      -354
h13 .word      -367
h14 .word      -236
h15 .word         0
h16 .word       255
h17 .word       431
h18 .word       451
h19 .word       292
h20 .word         0
h21 .word      -323
h22 .word      -551
h23 .word      -584
h24 .word      -383
h25 .word         0
h26 .word       438
h27 .word       763
h28 .word       827
h29 .word       557
```

```
h30 .word      0
h31 .word   -681
h32 .word  -1240
h33 .word  -1417
h34 .word  -1022
h35 .word      0
h36 .word   1533
h37 .word   3307
h38 .word   4960
h39 .word   6131
h40 .word   6131
h41 .word   4960
h42 .word   3307
h43 .word   1533
h44 .word      0
h45 .word  -1022
h46 .word  -1417
h47 .word  -1240
h48 .word   -681
h49 .word      0
h50 .word    557
h51 .word    827
h52 .word    763
h53 .word    438
h54 .word      0
h55 .word   -383
h56 .word   -584
h57 .word   -551
h58 .word   -323
h59 .word      0
h60 .word    292
h61 .word    451
h62 .word    431
h63 .word    255
h64 .word      0
h65 .word   -236
h66 .word   -367
```

```
h67 .word      -354
h68 .word      -211
h69 .word         0
h70 .word       198
h71 .word       310
h72 .word       301
h73 .word       180
h74 .word         0
h75 .word      -170
h76 .word      -268
h77 .word      -261
h78 .word      -157
h79 .word         0
```

Resumen

En este capítulo se ha descrito en forma breve las estructuras de los filtros digitales FIR e IIR, mostrando bloques de código de instrucciones del C54x que optimizan la implementación de las estructuras descritas. Asimismo, se incluye un filtro digital en tiempo real que hace amplio uso de la arquitectura e instrucciones del DSP. Cabe mencionar que con estos ejemplos el lector está en capacidad de diseñar e implementar aplicaciones similares y de mayor complejidad.

Capítulo 8

Aplicación: Sistema de Síntesis en Tiempo Real

En este capítulo se muestra una aplicación en tiempo real de un sistema de síntesis de voz describiendose desde el diseño y las estrategias tomadas hasta la implantación empleando el DSP TMS320C5402. Presentamos esta aplicación debido a que se puede considerar como un bloque modular de sistemas más complejos como un sistema de reconocimiento, transmisión, encriptamiento y almacenamiento de voz.

Se puede observar que los procesos que consideran a la señal de voz son la parte medular de los sistemas de comunicación, por lo que es necesario establecer aquellos procesos de voz que mejoren las tareas del almacenamiento, la velocidad de transmisión - recepción, la inteligibilidad (comprensión de la información), privacidad y la facilidad de uso con la calidad requerida en las aplicaciones en que tomen parte.

La reducción de la cantidad de información concretamente en una señal de voz, es un proceso indispensable para la manipulación de ésta en cualquier sistema que se desee implantar en tiempo real. Tal reducción justifica el empleo de estrategias para eliminar la redundancia que presentan las señales de voz; siendo la técnica de síntesis una opción eficiente para cumplir tal propósito; debido a que considera las propiedades físicas del sistema de producción de la señal de voz, caracterizándola con una calidad aceptable, y minimizando la tasa de bits a emplear sin perder la calidad de la voz al reconstruirla o descomprimirla. Por ejemplo, en lugar de transmitir 8000 muestras/s se podría reducir a menos de 500 parámetros, es decir, comprimir la información hasta en un 7.7 % [3].

8.1. Codificadores de Voz

Las técnicas de codificación de la señal de voz pretenden reducir el volumen de información necesario para almacenar o transmitir una señal de voz, de tal forma que la pérdida de calidad en la señal reconstruida con respecto a la señal sin comprimir sea mínima; además de mantener la inteligibilidad del mensaje y existir un compromiso de calidad contra la tasa de compresión, complejidad computacional, etc. Los métodos para la codificación de la señal de voz se dividen en dos categorías principales: Codificadores de Forma de Onda (dominio temporal y frecuencial) y Codificadores de Voz conocidos como *Vocoders*.

Los codificadores de *forma de onda* realizan eficientemente la compresión explotando las características temporales y/o espectrales de la señal de voz. Sin embargo, para obtener tal eficiencia, la tasa de bits para su transmisión y almacenamiento es muy alta. Por otro lado, en los *Vocoders*, la representación de la señal de voz es por medio de un conjunto de parámetros estimados con un modelo de producción de voz y una codificación en forma digital de éstos. Para la transmisión y almacenamiento de estos parámetros, los Vocoders resultan ser óptimos, en el sentido de usar una tasa de bits mínima en comparación con los codificadores de forma de onda.

Los Vocoders se basan en la representación de la señal de voz mediante un modelo todo polo del sistema del tracto vocal. En la producción de voz sintética para segmentos de voz con naturaleza voceada, la excitación es un tren de impulsos con un período igual al período fundamental del segmento. En el caso de segmentos no voceados, la excitación es una secuencia de ruido blanco (ver figura 8.3). En ambos casos la ganancia del segmento es estimada.

Los *Sonidos Voceados* (figura 8.1) son producidos como consecuencia de la oscilación de las cuerdas vocales forzando la apertura de la epiglotis. La apertura y cierre de las cuerdas vocales secciona el pulso de aire en pulsos cuasi - periódicos llamados pulsos glotales, aproximadamente con forma de onda triangular, con una frecuencia fundamental llamada *frecuencia de pitch* y frecuencias armónicas que disminuyen su amplitud a razón de 40 dB/decada [20]. El tracto vocal actúa como una cavidad resonante con una naturaleza de un filtro paso - bajas, proporcionando un espectro con una fuerte frecuencia fundamental y atenuando las frecuencias armónicas. El intervalo en el que vibran las cuerdas vocales depende de la presión del aire comprimido proveniente de los pulmones y la tensión de éstas, obteniendo así, la variación de la frecuencia fundamental y produciendo distintos sonidos voceados, donde el intervalo de pitch para hombres adultos es de 50 Hz hasta 250 Hz, y para una mujer adulta de 120 Hz hasta 500 Hz [4]. En la figura 8.1 se muestra un segmento de sonido voceado de

voz real masculina (256 muestras del fonema /e/ a una frecuencia de muestreo de $f_s = 8$ kHz).

En los *sonidos no voceados* (figura 8.2) las cuerdas vocales no vibran y son generados mediante la contracción de algunos puntos a lo largo del tracto vocal, forzando al aire a atravesar dicha contracción para producir cierta turbulencia, por tanto, la excitación de estos sonidos es de naturaleza ruidosa. En la figura 8.2 se muestra un segmento de sonido no voceado de voz real masculina (256 muestras del fonema /s/ a $f_s = 8$ kHz).

El Vocoder presentado como ejemplo de aplicación en tiempo real es un vocoder por codificación de predicción lineal (LPC), que a continuación describimos.

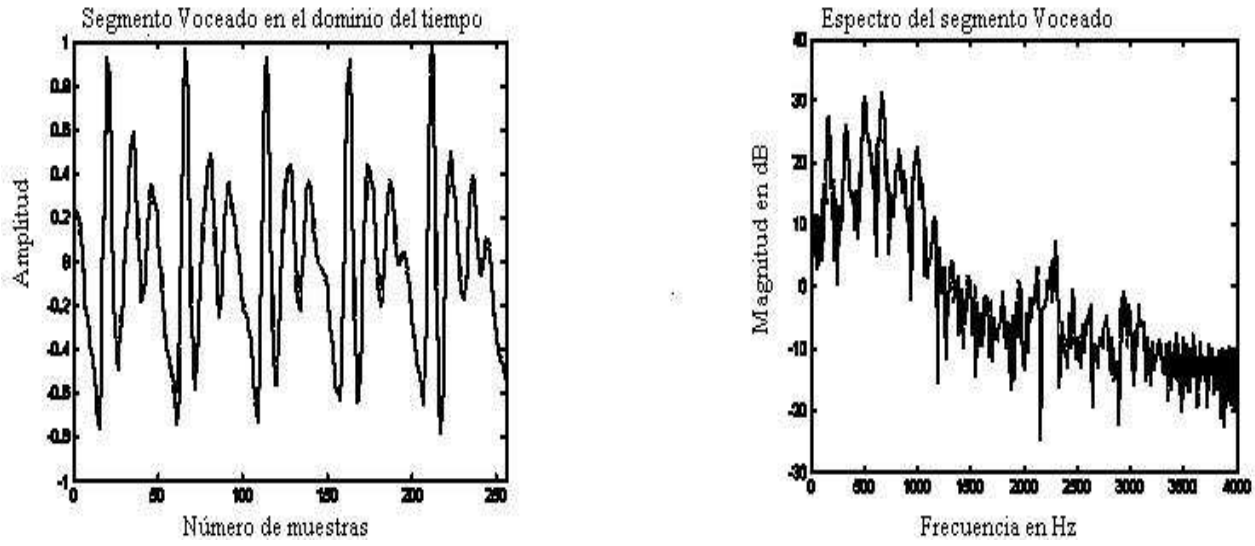


Figura 8.1: Representación de un segmento de Sonido Voceado en los dominios del tiempo y la frecuencia.

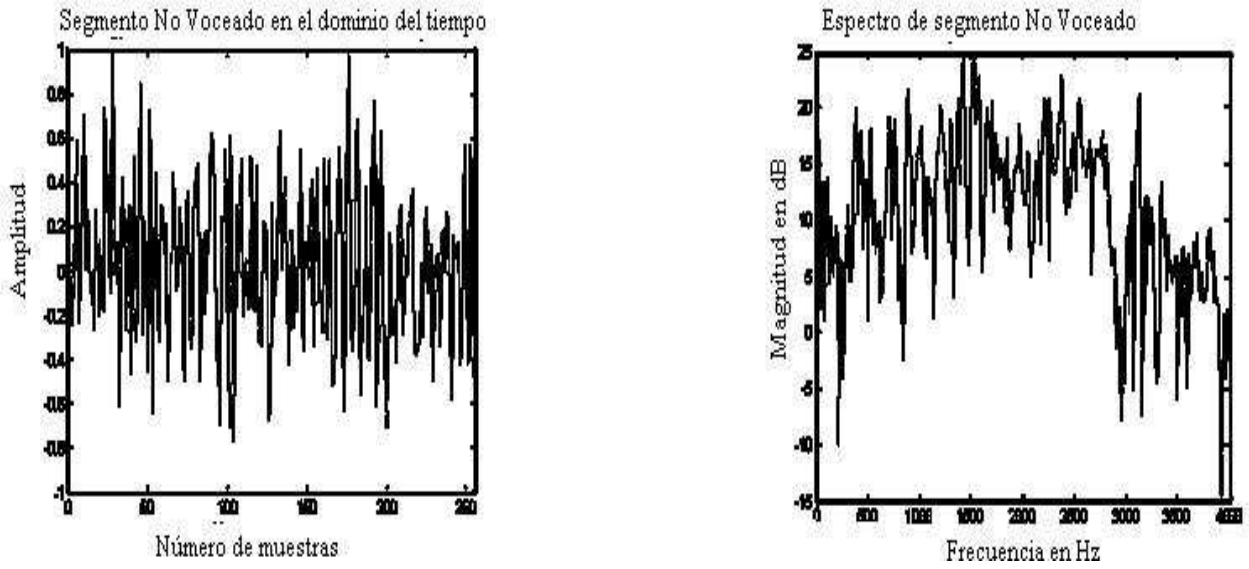


Figura 8.2: Representación de un segmento de Sonido No Voceado en los dominios del tiempo y la frecuencia.

8.2. Vocoder por codificación de predicción lineal

Es el tipo de Vocoder más utilizado debido a su simplicidad de implantación. Este Vocoder se basa en el modelo digital de la producción de voz (Modelo terminal - análogo), de tal manera que desacoplando los efectos producidos por el modelo del pulso glotal y el modelo de radiación debido a los labios, podemos simplificar tal modelo en un filtro todo polo conocido como modelo filtro - fuente como se muestra en la figura 8.3 [4]. Al reducir el modelo digital de la producción de la voz a un sistema todo polo resulta un sistema modelado por ecuaciones lineales cuyos parámetros son estimados por medio del método de predicción lineal [13], simplificando enormemente la solución de la modelización. Por tanto, la función de transferencia que caracteriza al modelo todo polo se muestra en la ecuación (8.1):

$$H(z) = \frac{G}{1 + \sum_{k=1}^{p} a_k z^{-k}} = \frac{G}{\prod_{k=1}^{p} (1 - p_k z^{-1})} \tag{8.1}$$

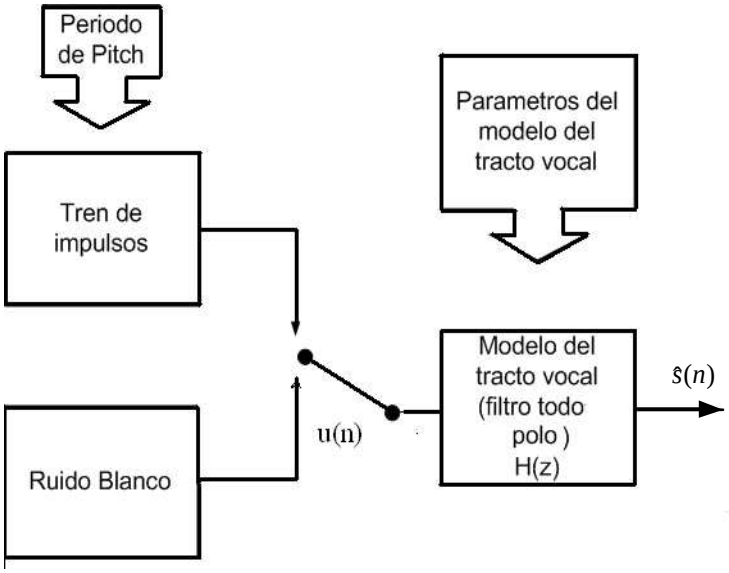


Figura 8.3: Modelo Todo Polo del tracto Vocal.

Donde G representa una ganancia, p_k los polos complejos que caracterizan al sistema y p el orden de la predicción. La ecuación en diferencias del modelo todo polo se presenta en la ecuación (8.2):

$$\widehat{s}(n) = -\sum_{k=1}^{p} a_k \widehat{s}(n-k) + u(n) \tag{8.2}$$

Donde $\widehat{s}(n)$ es la señal estimada de salida, $u(n)$ la entrada respectiva según la naturaleza del segmento y p el orden de la predicción lineal.

A continuación se presenta el método de predicción lineal y su respectiva solución que estima los parámetros necesarios para la modelización del tracto vocal mediante un filtro todo polo.

8.2.1. Predicción Lineal por el método de Autocorrelación

Una de las técnicas más poderosas en el análisis de señales de voz es el *método de predicción lineal*. Con este método es posible estimar los parámetros de la señal de voz que modelan al sistema de producción de la voz [27]. El proceso de análisis por predicción lineal consiste en aproximar las muestras a partir de una combinación lineal de muestras anteriores de la señal de voz. Al minimizar el promedio del error al cuadrado (en un intervalo de duración

finita) entre la señal original y la señal estimada se obtiene un conjunto único de coeficientes que ponderarán a la combinación lineal.

Mediante el método de la predicción lineal para la estimación de los parámetros del modelo todo polo consideramos segmentos de la señal de 10 a 30 ms [4], para garantizar que el segmento en estudio cumpla con las condiciones de estacionaridad [1] y ergódicidad[2].

Considerando el error de la predicción (figura 8.4 y 8.5):

$$e(n) = s(n) - \widehat{s}(n) \tag{8.3}$$

$$\widehat{s}(n) = -\sum_{k=0}^{p} a_k s(n-k) \tag{8.4}$$

$$e(n) = s(n) + \sum_{k=0}^{p} a_k s(n-k) \qquad a_0 = 1 \tag{8.5}$$

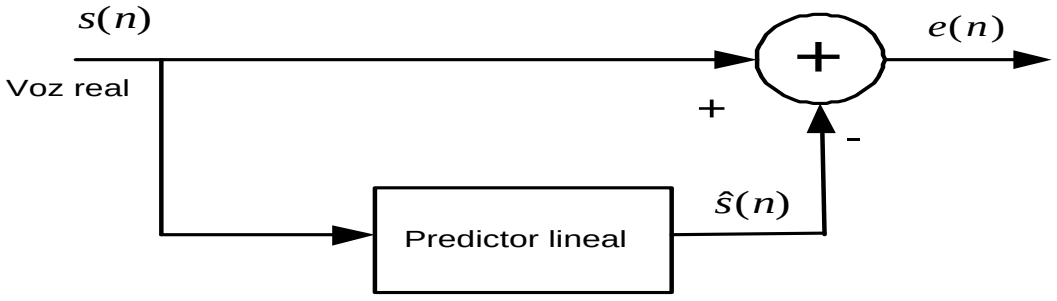


Figura 8.4: Error de predicción.

Donde $s(n)$ es la señal original, $\widehat{s}(n)$ es la señal estimada, $e(n)$ es la señal de error, $\{a_i\}$ el conjunto de coeficientes de la predicción y p el orden de la predicción. Mostrando la ecuación (8.5) en notación vectorial (ecuación 8.6):

[1] Un proceso aleatorio es estacionario si sus estimaciones estadísticas no varían con el tiempo [4].

[2] Un proceso aleatorio es ergódico si sus estimaciones estadísticas puede ser aproximadas por sus estimaciones temporales [4].

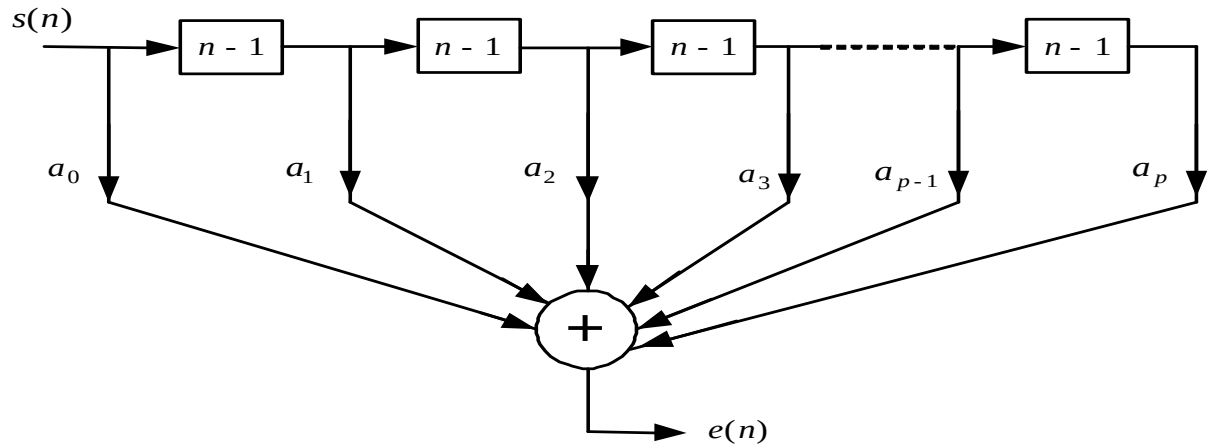


Figura 8.5: Filtro de predicción.

$$e(n) = s(n) + A_P^T s_{p_i} \tag{8.6}$$

$$A_P^T = \begin{bmatrix} a_1 & a_2 & a_3 & .... & a_p \end{bmatrix}$$

$$s_{p_i} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_p \end{bmatrix}$$

Aplicando el criterio del error cuadrático a la ecuación (8.6), resulta la ecuación (8.7):

$$\begin{aligned} e^2(n) &= \left(s(n) + A_P^T s_{p_i}\right)\left(s(n) + A_P^T s_{p_i}\right) \\ &= s^2(n) + s(n)A_P^T s_{p_i} + A_P^T s_{p_i} s(n) + A_P^T s_{p_i} A_p^T s_{p_i} \\ &= s^2(n) + 2s(n)A_p^T s_{p_i} + A_p^T s_{p_i} s_{p_i}^T A_p \end{aligned} \tag{8.7}$$

Considerando que $s(n)$ es una señal aleatoria y estacionaria, obteniendo la esperanza matemática $E\{\cdot\}$ de la ecuación (8.7):

$$\begin{aligned} E\left\{e^2(n)\right\} &= E\left\{s^2(n) + 2s(n)A_p^T s_{p_i} + A_p^T s_{p_i} s_{p_i}^T A_p\right\} \\ &= r_p(0) + 2A_p^T E\left\{s(n)s_{p_i}\right\} + A_p^T E\left\{s_{p_i} s_{p_i}^T\right\} A_p \end{aligned} \tag{8.8}$$

Expresando al vector de autocorrelación de la señal $s(n)$ como:

$$r_p(i) = E\left\{s(n)s_{p_i}\right\} = \left\{ \begin{array}{cccccc} r_1 & r_2 & r_3 & .... & r_p \end{array} \right\} \tag{8.9}$$

Y a la ecuación (8.10) como la matriz de autocorrelación, siendo ésta una matriz del tipo Toeplitz [13]:

$$R_p = E\left\{s_{p_i} s_{p_i}^T\right\} = \left[\begin{array}{ccccc} r_0 & r_1 & r_2 & ... & r_p \\ r_1 & r_0 & r_1 & ... & r_{p-1} \\ r_2 & r_1 & r_0 & .. & r_{p-2} \\ \vdots & & & & \vdots \\ r_p & .. & .. & ... & r_0 \end{array} \right] \tag{8.10}$$

Entonces, la esperanza matemática del error cuadrático se expresa como muestra la ecuación (8.11):

$$E\left\{e^2(n)\right\} = r_p(0) + 2A_p^T r_p + A_P^T R_p A_p \tag{8.11}$$

Aplicando el criterio de optimización sobre $E\{e^2(n)\}$ resulta la ecuación (8.12):

$$\begin{aligned} \frac{\partial E\left\{e^2(n)\right\}}{\partial A_p} &= 0 + 2r_p + 2R_p A_p = 0 \\ R_p A_p &= -r_p \qquad (8.12) \\ A_p &= -R_p^{-1} r_p \qquad (8.13) \end{aligned}$$

A la ecuación (8.13) se le denomina *ecuación Normal* dado que el error es perpendicular al espacio vectorial s_p. La ecuación (8.13) es la solución de minimizar el promedio del error cuadrático entre la señal $s(n)$ y la señal estimada $\widehat{s}(n)$ y de esta manera definimos el método de la autocorrelación para la resolución de la codificación por predicción lineal. Todavía queda encontrar el conjunto de valores de A_p que son los coeficientes que ponderan a la combinación lineal que define la predicción lineal, para ello, en la siguiente sección presentamos un algoritmo que resuelve lo anterior, llamado algoritmo de Levinson - Durbin.

8.2.2. Algoritmo de Levinson - Durbin

Para la solución de la ecuación normal (ecuación 8.13) es necesario invertir la matriz R_p, considerando que el cálculo del conjunto de valores de A_p se tendrá que hacer con el mínimo de operaciones para justificar su empleo en un sistema de síntesis de voz en tiempo Real.

El algoritmo de Levinson - Durbin es un procedimiento computacionalmente eficiente para resolver las ecuaciones normales (ecuación 8.13) para los coeficientes de predicción. Este algoritmo explota la simetría especial en la matriz de autocorrelación (ecuación 8.10) de tal modo que $R_p(i,j) = R_p(i+1, j+1)$, así la matriz de autocorrelación es una *matriz Toeplitz*. Además, $R_p(i,j) = R_p^*(j,i)$, la matriz es también *Hermitiana* [22]. La clave del algoritmo de Levinson - Durbin consiste en proceder recursivamente, empezando con un predictor de orden $p = 1$ y después incrementar el orden recursivamente, usando las soluciones de orden menor para obtener la solución al siguiente orden superior. Entonces, el algoritmo de Levinson - Durbin propone una solución al orden $(p+1)$ con base a la solución al orden p [13]. El organigrama del algoritmo de la recursión de Levinson - Durbin se muestra en el cuadro 8.1. Donde E es error cuadrático y podemos considerarlo como una ponderación de la energía de la señal con los coeficientes de la predicción.

1) Iniciar con:
a) $a_0(0) = 1 = k_0$
b) $\varepsilon_0 = r_0$
2) Para j=0,1,2,$\cdots, p-1$
a) $\gamma_j = r_p(j+1) + \sum_{i=1}^{j} a_j(i) r(j-i+1)$
b) $k_{j+1} = -\frac{\gamma_j}{\varepsilon_j}$
c) Para i=1,2,3,$\cdots, j$
$a_{j+1}(i) = a_j(i) + k_{j+1} a_j(j-i+1)$
d) $a_{j+1}(j+1) = k_{j+1}$
e) $\varepsilon_{j+1} = \varepsilon_j(1 - k_{j+1}^2)$
3) $\mathrm{E}^2(0) = \varepsilon_j$

Cuadro 8.1: Organigrama del algoritmo de Levinson - Durbin

8.3. Diseño del Sistema de Síntesis de Voz

En la figura 8.6 se presenta un esquema general del proceso de un sistema de síntesis que engloba un *bloque de análisis*, donde se estimarán los parámetros de la señal en cuestión (coeficientes LPC, naturaleza del segmento, estimación del pitch, ganancia del segmento) y un *bloque de síntesis*, que mediante los parámetros obtenidos en el proceso de análisis se reconstruirá a la señal. A continuación se presentan los subprocesos que constituyen a los procesos de análisis y síntesis, respectivamente.

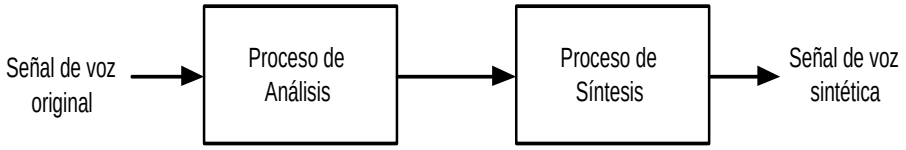


Figura 8.6: Sistema General de Síntesis de Voz.

- **Proceso de Análisis**:

  1. Estimación de la energía de la señal para decidir si el segmento es de silencio. Una forma de calcular la energía del segmento de voz en estudio es por medio de la ecuación (8.14):

$$E = \sum_{i=0}^{N-1} |s(n)|^2 \tag{8.14}$$

     Donde E es la energía del segmento, $s(n)$ es el segmento de voz y N es el número de muestras para cada segmento. La elección del valor del umbral para la determinación de la existencia de un segmento de silencio es dependiendo de las condiciones de ruido en que se realice la adquisición de la voz. Cabe mencionar que si el segmento es considerado como de silencio no será necesario estimar los parámetros de las siguientes etapas en el proceso de análisis.

  2. Aplicar un proceso de ventaneo de la señal de voz original eligiendo a la ventana de Hamming por ser la más utilizada para síntesis de voz debido a que el ancho de transición aproximado del lóbulo principal de la ventana es de $8\pi/N$ y la amplitud de la diferencia entre el primer y segundo lóbulo es de $-43\ dB$ [4]. La ecuación (8.15) muestra la definición de la ventana de Hamming:

$$w(n) = 0.54 - 0.46cos\left(\frac{2\pi n}{N-1}\right) \tag{8.15}$$

Donde N es la longitud de la ventana.

3. Decidir la naturaleza del segmento bajo estudio y el período de pitch, mediante *el método del recorte central* [21] que consiste en recortar las amplitudes de una ventana entre los niveles $\pm C_L$, es decir, solo consideramos las amplitudes de la ventana a partir de los niveles de umbral propuestos y posteriormente calculamos su función de autocorrelación para determinar la periodicidad de la señal. En forma analítica podemos definir a la función del recorte central como se muestra en la ecuación (8.16):

$$\begin{array}{ll} y(n) = s(n) - C_L & \text{Si } \ s(n) \geq C_L \\ y(n) = s(n) + C_L & \text{Si } \ s(n) \leq -C_L \\ y(n) = 0 & \text{Otro caso} \end{array} \tag{8.16}$$

Otra manera de definir a la función del recorte central, es considerando una forma más eficiente de recortar a la señal por razones de formato numérico[3]:

$$\begin{array}{lll} y(n) = & 1 & \text{Si } \ s(n) \geq C_L \\ y(n) = & -1 & \text{Si } \ s(n) \leq -C_L \\ y(n) = & 0 & \text{Otro caso} \end{array} \tag{8.17}$$

El efecto de aplicar un proceso de recorte es conservar solo la información de la periodicidad de la señal. El nivel de umbral de C_L es calculado para cada ventana si dividimos a la ventana en tres subsegmentos y empleamos la siguiente estrategia:

- Encontrar las amplitudes máximas del primer y tercer subsegmento (A_1 *y* A_3).
- Calcular el umbral C_L como se muestra en la ecuación (8.18):

$$C_L = K\ min(A_1, A_3) \tag{8.18}$$

Donde el operador $min()$ calcula el valor mínimo entre A_1 *y* A_3 y K es un parámetro de calibración que se encuentra en el intervalo de 0.6 a 0.8 [21].

[3]Evitando así desbordamientos al momento de realizar las multiplicaciones y acumulaciones en la función de autocorrelación.

Entonces, ya determinado el umbral C_L y obtenido la señal recortada, aplicamos la función de autocorrelación a la señal recortada. La función de autocorrelación se define como:

$$R(l) = \sum_{n=0}^{N-1} s(n)s(n+l) \tag{8.19}$$

Donde $s(n)$ es el segmento de voz recortado y N es el número de muestras del segmento y l el índice temporal de retraso.

En el caso de segmentos de sonidos voceados existe una periodicidad de la forma de onda en la señal, no así para ventanas no voceadas. Por tanto, la función de autocorrelación tiene la propiedad de ser periódica si la ventana en estudio lo es. Una ventaja de usar la función de autocorrelación es que presenta con mayor realce la propiedad de periodicidad de las señales. En consecuencia, la propiedad de periodicidad puede ser utilizada como criterio para la estimación de la naturaleza del segmento. Sin embargo, además de resaltar la periodicidad del segmento, también muestra información de la energía de la señal, R(0). Esta información adicional puede propiciar decisiones erróneas [21]. Entonces para la determinación de la naturaleza de la ventana se calcula el valor máximo de la función de autocorrelación comprendida entre las muestras 20 a la 200, y este valor es comparado con un valor umbral igual a $0.3R(0)$. Si el valor máximo es mayor que el valor umbral se decide como un segmento voceado, si es menor, el segmento será no voceado.

4. Aplicar al segmento (ya sea voceado o no voceado) un proceso de filtrado de preénfasis (filtro FIR de primer orden paso - altas). En la ecuación (8.20) definimos el filtro de preénfasis que solo posee un cero en α.

$$H_{pre}(z) = 1 - \alpha z^{-1} \tag{8.20}$$

Donde α es un valor cercano a 0.9. El valor de α comúnmente usado es de 0.9375. [21]. La ecuación en diferencias del filtro de preénfasis es:

$$y(n) = x(n) - \alpha x(n-1) \tag{8.21}$$

Donde $y(n)$ es la salida y $x(n)$ es la entrada al filtro.

5. Estimar los parámetros LPC y calcular la ganancia del segmento por medio del *algoritmo de Levinson - Durbin* [13].

- **Proceso de Síntesis**:

1. Lectura de parámetros.
2. Si el segmento en el proceso de análisis se decidió como de silencio generar una ventana de ceros.
3. Aplicar un Filtro todo polo (Modelo filtro - fuente) definido por los parámetros LPC y la ganancia, a la entrada de excitación según sea la naturaleza del segmento (voceado - tren de impulsos, no voceado - ruido blanco, figura 8.3. Entonces la función de transferencia del filtro todo polo está definida por la ecuación (8.22):

$$H(z) = \frac{G}{1 + \sum_{i=1}^{p} a_i z^{-1}} \tag{8.22}$$

o bien por su ecuación en diferencias:

$$\hat{y}(n) = Gx(n) - a_1\hat{y}(n-1) - a_2\hat{y}(n-2) - \ldots - a_p\hat{y}(n-p) \tag{8.23}$$

Donde G es la ganancia del segmento, $x(n)$ la entrada de excitación, $\hat{y}(n)$ es la voz sintética o estimada.

4. Proceso de filtrado de deénfasis. El filtro de deénfasis puede ser definido como un sistema IIR (de respuesta infinita al impulso), que se muestra en la ecuación (8.24):

$$H(z) = \frac{1}{1 - \alpha z^{-1}} \tag{8.24}$$

Donde α es el valor del polo en la función de transferencia. El filtro de deénfasis tiene un comportamiento paso - bajas, resaltando de esta forma las primeras frecuencias formantes. Se puede observar que el filtro de deénfasis es un filtro inverso del filtro de preénfasis (proceso de análisis) por lo que es usual elegir el mismo valor empleado por éste [4]. La ecuación en diferencias del filtro de deénfasis es:

$$y(n) = x(n) + \alpha y(n-1) \tag{8.25}$$

8.4. Implantación del Sistema de Síntesis de Voz en Tiempo Real

Para la implantación del sistema de síntesis de voz en tiempo real se emplea el DSP TMS320C5402 de punto fijo, donde los procesos de análisis y síntesis se integran en un sistema multitareas, por lo que se incluye una etapa de adquisición de la señal de voz por medio de un proceso de conversión analógico - digital. De igual manera, la voz sintética obtenida deberá ser presentada en el dominio analógico, esto, al emplear un convertidor digital - analógico. Así, en primera instancia, se observa que los procesos de adquisición de voz real y de entrega de voz sintética se realizan en conjunción con el procedimiento de análisis - síntesis. Lo anterior se logra con el uso de las interrupciones definidas por la arquitectura del DSP. En la figura 8.7 se muestra el esquema general del sistema de síntesis de voz en tiempo real que está constituido por:

- **El proceso de adquisición de voz real y entrega de voz sintética**: producidos por una llamada de atención a interrupción de recepción del puerto serie multicanal bufereado (periférico interno del DSP) [44], que está conectado a un convertidor A/D y D/A (doble convertidor TLC30AD50, Codec) que cuenta con una interfaz para micrófono y otra para altavoz.
- **El proceso de análisis**: aquí se estiman los parámetros de voz del segmento adquirido. Este proceso es ejecutado por el DSP y consta de los siguientes bloques:
  - *Proceso de Ventaneo*: Se realiza utilizando la ventana de Hamming sobre la señal con un traslape del 30 % de la totalidad del segmento. La longitud de la ventana es de N=240 muestras, es decir, adquirimos un segmento en 30 ms a una frecuencia de muestreo de $f_s = 8$ kHz.
  - *Decisión de la existencia de un segmento de Silencio*: Se calcula la energía de la ventana y se compara con un umbral de silencio que se ha calculado experimentalmente según las condiciones de ruido del entorno de adquisición.
  - *Decisión de la naturaleza de la ventana*: Mediante el método del recorte central se toma la decisión de la naturaleza de la ventana (segmento voceado o no voceado).
  - *Estimación del Pitch*: Si la ventana se decidió como voceada, se dispone a estimar el período de pitch.
  - *Filtrado de Preénfasis*: El segmento con ventana Hamming es filtrado para realzar las altas frecuencias.

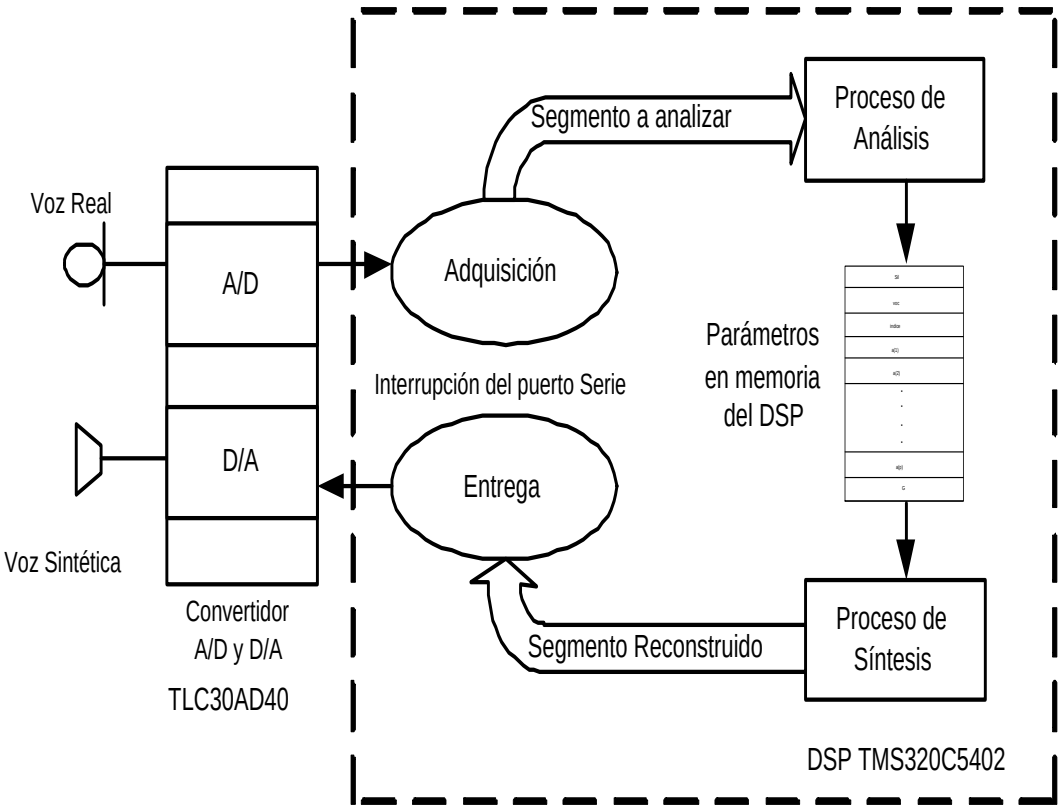


Figura 8.7: Esquema General del sistema de síntesis de voz en tiempo real.

- *Estimación de los coeficientes LPC*: Por medio del algoritmo de Levinson - Durbin se estima los parámetros LPC.
- *Cálculo de la Ganancia del segmento*: La ganancia es calculada por la recursión de Levinson - Durbin, siendo igual al error cuadrático evaluado en el orden p.

Los parámetros estimados se definen como un paquete de datos para su posible transmisión y/o almacenamiento mostrado en la figura (8.8). El tamaño del paquete de datos que contiene a los parámetros estimados está en función del orden de la predicción a utilizar, es decir, para nuestro caso el paquete consta de 14 parámetros (p=10).

- **El proceso de síntesis**: aquí se reconstruye la señal de voz por medio de los parámetros estimados definidos por el paquete de datos que se muestran en la figura (8.8). El proceso de síntesis (ejecutado por el DSP) engloba los siguientes bloques:

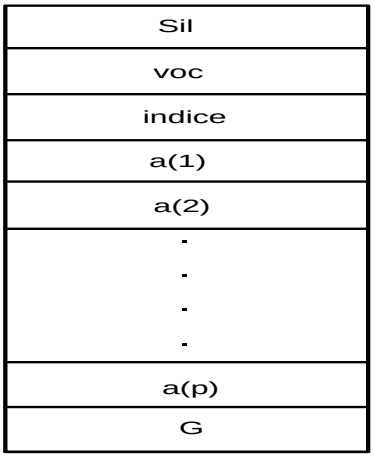


Figura 8.8: Definición del paquete de datos de los parámetros estimados en una ventana.

- *Generación de un segmento de 240 ceros:* si existe silencio.
- *Filtro Todo Polo*: Según la decisión de la naturaleza del segmento voceado o no voceado, se elige la excitación (tren de impulsos o ruido blanco) de un filtro todo polo definido por los parámetros LPC y la ganancia. El filtro Todo Polo reconstruye una señal sintética al modelar al sistema de producción de voz (tracto vocal).
- *Filtrado de Deénfasis*: la señal reconstruida por el filtro Todo Polo es filtrada para atenuar las altas frecuencias modelando la radiación producida por los labios.

La organización de los procesos a realizar por nuestro sistema de síntesis de voz en tiempo real opera como una estructura "pipeline", es decir, las tareas de adquisición, proceso de análisis - síntesis y de entrega de la voz sintética se ejecutan contemplando que mientras se adquiere un segmento, se procesa el anterior y, posteriormente se entrega la voz reconstruida y así sucesivamente.
En la figura 8.9 se muestra la organización de las tareas a ejecutarse.

El esquema de tiempos de la ejecución de las tareas (figura 8.10) presenta una señal de voz real de la palabra rojo (voz masculina) muestreada a una frecuencia de f_s= 8 kHz.

Se observa que la primera tarea a ejecutarse es la adquisición de un segmento de tamaño de 240 muestras, el segmento 1 se almacena en un arreglo en memoria DARAM (RAM de doble acceso) del DSP. Al llenarse este arreglo, mediante una bandera de sincronía, se dispone a ejecutarse los procesos de análisis - síntesis para el segmento 1, estos procesos no deberán

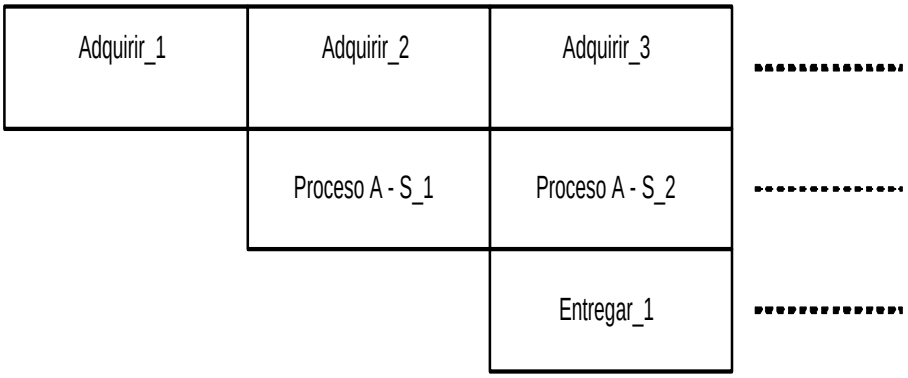


Figura 8.9: Organización de las tareas a ejecutarse.

de sobrepasar los 30 ms (tiempo de la adquisición para cada ventana) en sus respectivos cálculos. Sin embargo, por medio de la interrupción de recepción del puerto serie del DSP, la adquisición del siguiente segmento (segmento 2) se realiza en conjunción con los procesos de análisis - síntesis para el segmento 1 (figura 8.10). De igual manera, al adquirir el segmento 3, se levanta la bandera de sincronía, ejecutándose los procesos de análisis - síntesis para el segmento 2, mientras que se entrega el segmento reconstruido del segmento 1 (por medio de la interrupción de recepción del puerto serie). Las tareas de adquisición, procesos de análisis - síntesis y entrega de voz sintética se ejecutan de manera indefinida por ser un sistema en tiempo real.

En la figura 8.11 se muestra el diagrama de flujo de las tareas de adquisición, entrega y los procesos de análisis - síntesis, para su implantación en el DSP C5402. Los procedimientos presentados en el diagrama de flujo de la figura 8.11 se describen a continuación:

- ***Definición e inicialización de variables***: Se definen e inicializan las variables y arreglos necesarios para la ejecución de las tareas de adquisición de voz real, entrega de la señal reconstruida y los procesos de análisis y síntesis.
- ***Configuración e inicialización del Codec***: Es necesario la inicialización del Convertidor A/D y D/A y su configuración que consiste en abrir el puerto donde está conectado el codec (puerto serie) y definir la frecuencia de muestreo, así como las ganancias de entrada y salida de éste.
- ***Configuración de Interrupciones***: Se habilitan las interrupciones a emplear, para nuestro diseño, se utiliza la interrupción de recepción por hardware del puerto serie.

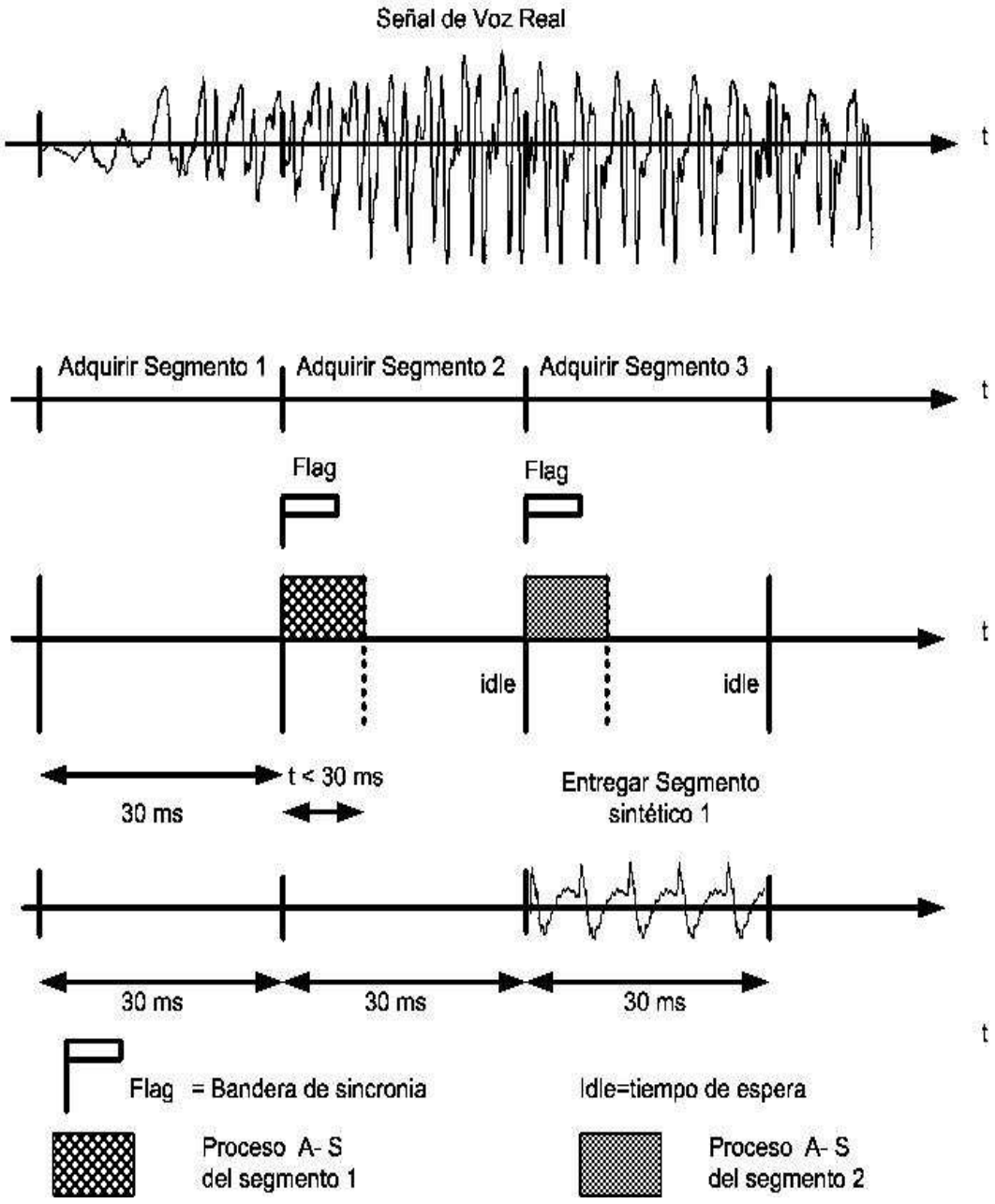


Figura 8.10: Esquema de tiempos de la ejecución de las tareas.

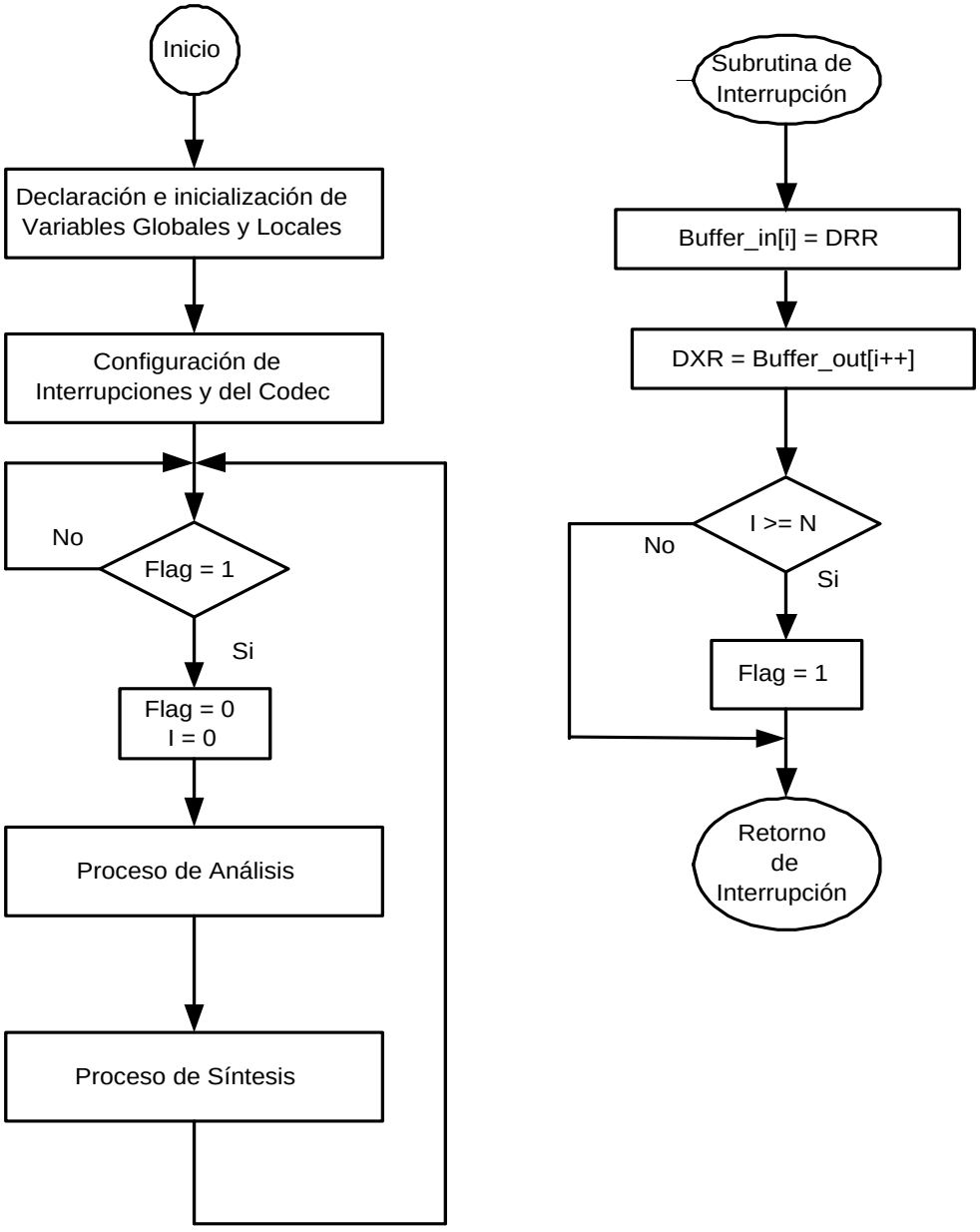


Figura 8.11: Diagrama de Flujo de las tareas de adquisición, entrega y los procesos de análisis - síntesis.

- ***Bandera de sincronía***: Esta variable (flag) funge como bandera para determinar si el arreglo está lleno (adquisición) dependiendo de la longitud del segmento que sea empleada.
- ***Proceso de Análisis***: Se ejecutan los bloques correspondientes al proceso de análisis.
- ***Proceso de Síntesis***: Se ejecutan los bloques correspondientes al proceso de síntesis.
- ***Llamada de atención a Interrupción***: Mediante una subrutina de interrupción se adquiere la señal real, donde los valores adquiridos se almacenan en un registro de recepción del puerto serie (DRR), también se entrega la señal sintética almacenando éstos valores en el registro de transmisión del puerto serie (DXR). Estos registros pasan sus respectivos valores a las etapas de conversión A/D y D/A. Por medio de la bandera de sincronía (flag) se establece si el arreglo utilizado para almacenar al segmento adquirido está lleno.

8.5. Implantación del Sistema en el DSP TMS320C5402

El proyecto que se diseño en el DSP TMS320C5402 para la implantación del sistema de síntesis de voz debe incluir las librerías; rts.lib, dsk5402.lib, drv5402.lib, 54xdsp.lib, así como un archivo con extensión .cmd. y un archivo escrito en ensamblador donde se definen los vectores de interrupción. Los códigos fuente de dicho proyecto hacen uso de aritmética de punto flotante y punto fijo utilizando el lenguaje C. Además, utilizamos lenguaje optimizado, es decir, se hacen llamadas a funciones escritas en ensamblador definidas por la librería dsplib.h, como también instrucciones intrinsicas. Presentamos el código principal en lenguaje C que ejemplifica la implantación del sistema de síntesis de voz (sistema.c).

Programa principal

```
/* sistema.c */

#include "librerias.h"    // librerías a utilizar
#include "variables.h"    // definición de las variables a emplear
#include "funciones.h"    // definición de las funciones utilizadas
#include "interrupcion.h" // definición de la subrutina de
                          // interrupción
```

```
 void main(void) {          // función principal
  GLOBAL_INT_DISABLE;       // deshabilitadas las interrupciones
  brd_init(100);            // inicialización de la tarjeta DSK en
                            // tiempo real a 100 MHz

  // inicialización y configuración del convertidor A/D y D/A
  hHandset = codec_open(HANDSET_CODEC);
  codec_dac_mode(hHandset, CODEC_DAC_15BIT);
  codec_adc_mode(hHandset, CODEC_ADC_15BIT);
  codec_ain_gain(hHandset, CODEC_AIN_0dB);
  codec_aout_gain(hHandset, CODEC_AOUT_MINUS_0dB);
  codec_sample_rate(hHandset,SR_8000);

  *pmst = 0x00A0;         // mapemos los vectores de interrupción
  *imr |= 0x0400;         // mascara habilitada para la interrupción
                          // del codec
  *ifr = *ifr;            // limpiamos banderas de interrupción
  GLOBAL_INT_ENABLE;      // habilitamos interrupciones

   m_d=m_d1=0;            // limpiamos banderas de sincronía
   flag=flag1=0;

while(1) {                // bucle infinito
  while(flag==0){};       // espera hasta que se adquiera 240
                          // muestras
  flag=0;                 // se limpia bandera


  /************Proceso de análisis*************************************/

  for (i=0; i<N; i++)           // se guarda la trama a analizar
         buff1[i]=datain[i];
  silencio(buff1);              // se calcula si es trama de silencio
  if (sil==0)                   // si no es trama de silencio se procesa
  {
   adquirir(buff1,buff2);       // se realiza el proceso de ventaneo
   pitch(buff2);                // se calcula la naturaleza del segmento
```

```
                                // y se estima el período de pitch
   preenfasis(buff2,buff1);     // filtrado de preénfasis


   // calculo de la función de autocorrelación

   for (i=0;i<N;i++)
       buff1[i]=buff1[i]>>1;
   acorr(buff1,r,N,p+1,raw);
   for (i=0;i<p+1;i++)
       rxx[i]=((float)(r[i]<<2));

   levinson(rxx,ai);            //se estiman los parámetros LPC


  /************Proceso de síntesis*************************************/
  excitacion(exc);        // se genera la entrada de excitación
                          // dependiendo de la naturaleza del segmento

  tracto(exc,buff2,ai,gain); // filtro todo polo
  deenfasis(buff2,buff1);    // filtrado de Deénfasis
  for (i=0; i<N; i++) dataout[i]=buff1[i]<<1; //se guarda la trama
                                              //sintética
  flag1=1;                   //se alza bandera de sincronía
  m_d1=0;
  }

  // si la trama fuese de silencio se genera un segmento de ceros

  else
   {
     for (i=0; i<N; i++) dataout[i]=0;
   }
 }  // fin while(1)
}   // fin main()
```

Archivo "librerias.h"

```
 /* librerias.h */

 // librerías necesarias para utilizar el codec
 #include <type.h>
 #include <board.h>
 #include <codec.h>

 // librería de funciones dsplib
#include <dsplib.h>


 // archivos que definen la ventana de hamming y una secuencia de
 // ruido. Estas variables se definen como constantes.

  #include "hamm240.dat"
  #include "ruido240f.dat"
```

Archivo "variables.h"

```
/* variables.h */

 //definición de la variables a emplear

 #define N  240
 #define Ns  60
 #define p   10

 #define GLOBAL_INT_ENABLE     asm( " rsbx intm ")
 #define GLOBAL_INT_DISABLE    asm( " ssbx intm ")


 // asignación de puntero a puntero para utilizar los registros
 // mapeados en memoria del DSP
```

```
 volatile short *imr = (short *) 0x00;
 volatile short *ifr = (short *) 0x01;
 volatile short *pmst= (short *) 0x1d;
 volatile short *drr = (short *) 0x41;
 volatile short *dxr = (short *) 0x43;

// variable necesaria para el codec HANDLE hHandset;

// indices y banderas utilizadas
short i,j,m_d,m_d1,flag,flag1;

// arreglos enteros a utilizar
short buff1[N],buff2[N],datain[N], dataout[N];
short sil,voc,indice; short overlap[Ns];

// formato punto fijo q15
short K=26214;       // K=0.8
short u_max=9830;    // u_max=0.3
short pre_de=30720;  // pre_de=0.9375
short CL,maximo,umbral_v;
short ipk[2];
short segm2[N],segm3[N],r[p+1];
long  Es=2e9;
long  es;

// arreglos en formato de punto flotante
float rxx[p+1],ai[p+1],gain,exc[N], rx[p+1];
```

Archivo "funciones.h"

```
/* funciones.h */

// función que decide si el segmento es de silencio o no
void silencio(short *in)
 {
   es=0;
```

```
   for (i=0; i<N; i++)  es=_smac(es,in[i],in[i]);
   if (Es >= es) sil=1;
   else sil=0;
 }

// función que realiza el ventaneo
void adquirir(short *in, short *out)
{
  for (i=0; i<Ns; i++) out[i]=overlap[i];
  for (i=Ns,j=0; i<N; i++,j++) out[i]=in[j];
  for (i=N-Ns,j=0; i<N; i++,j++) overlap[j]=in[i];
  for (i=0; i<N; i++)  out[i]=_smpy(hamm[i],out[i]);
}



// función que estima la naturaleza del segmento y calcula el
// período de pitch

void pitch(short *in)
  {
   short segm1[80];

   ipk[0]=maxval(in,80);
   for (i=160, j=0; i<N; i++,j++) segm1[j]=in[i];
   ipk[1]=maxval(segm1,80);
   CL=minval(ipk,2);
   CL=(K*CL)>>15;

   for (i=0; i<N; i++)
     {
       if (in[i] >= CL) segm2[i]=1000;
       else if (in[i] <= -CL) segm2[i]=-1000;
       else segm2[i]=0;
     }

   acorr(segm2,segm3,N,N,raw);
```

```
   for (j=0; j<180; j++) segm2[j]=segm3[j+20];
   maximo=maxval(segm2,180);
   umbral_v=_smpy(u_max,segm3[0]);

   if (umbral_v < maximo)
     {
      voc=1;
      for (i=0; i<180; i++)
        {
          if(segm2[i]==maximo)
           {
            indice=i;
            break;
           }
          else
            indice=indice+1;
        }
     indice=indice+20;
    }
  else
     {
      voc=0;
      indice=0;
     }
 }


// filtro de preénfasis

void preenfasis(short *in,short *out)
 {
   out[0]=in[0];
   for (i=1; i<N; i++) out[i]=in[i] - _smpy(pre_de,in[i-1]);
 }
```

```
// algoritmo de Levinson - Durbin

void levinson(float *r, float *aj1)
 {
   float aj[p+1],kj, gama,e,sum;
   short i,j;
   e=r[0];
   aj1[0]=1;

   for (j=0; j<=p-1; j++)
     {
      sum=0;
      for (i=1; i<=j; i++)
         sum+=aj[i]*r[j-i+1];

      gama=r[j+1]+sum;
      kj=-gama/e;

      for (i=1; i<=j; i++)
        aj1[i]=aj[i]+kj*aj[j-i+1];

      aj1[j+1]=kj;
      e=e*(1-kj*kj);

      for (i=0; i<=p; i++)
        aj[i]=aj1[i];
 }
   gain=e*0.000001 ;
}


// función de genera las señales de excitación

void excitacion(float *in)
  {
    if (voc==1)
      {
```

```
        for (i=0; i<N; i++)
           in[i]=0;

        for (i=0; i<N; i=i+indice)
           in[i]=1;
      }
    else
      {
        for (i=0; i<N; i++)
           in[i]=ruido[i]*0.10;
       }
  }


// filtro todo polo

 void tracto(float *entr, short *sal, float *ais, float Gain)
    {
      float y_ant[p];
      float yn,temp;

      for (i=0; i<p; i++)
        y_ant[i]=0;

      for (i=0; i<N; i++)
        {
         yn=0;
         temp=entr[i];
         for(j=0; j<p; j++)
             yn += ais[j+1]*y_ant[j];
         sal[i] = (short)(Gain*temp - yn);
         for(j=p-1; j>=1; j--)
             y_ant[j] = y_ant[j-1];
         y_ant[0]=(float)sal[i];
        }
    }
```

```
/ /filtro de deénfasis

 void deenfasis(short *in,short *out)
   {
     out[0]=in[0];
     for (i=1; i<N; i++)
       out[i]= in[i] + _smpy(pre_de,out[i-1]);
   }
```

Archivo "interrupcion.h"

```
// subrutina de atención a interrupción, producida por la recepción de
// muestras en el puerto serial bufereado 1 que está conectado al codec

interrupt void recibir(void)
 {
   if (flag1==1)
     *dxr = dataout[m_d1++];
   else
      *dxr=0;

   if(m_d1>N)
     {
       m_d1=0;
       flag1=0;
     }

   datain[m_d++] = *drr;
   if (m_d>N)
     {
      m_d=0;
      flag=1;
     }
  }
```

Definición de los espacios de memoria a emplear

El archivo de ligado de comandos "memoria.cmd" que define los espacios de memoria utilizados en este proyecto es el siguiente:

```
/* memoria.cmd */

MEMORY
 {
   PAGE 0:

     VECS: org=0080h, len=0080h  /* vectores de interrupción */
     P   : org=0100h, len=3ff0h  /* espacio programa         */

   PAGE 1:

     D   : org=02500h,len=1500h  /* espacio dato             */
  }

  SECTIONS
   {
      vectors : load = VECS PAGE 0
     .text    > P           PAGE 0
     .bss     > D           PAGE 1
     .stack   > D           PAGE 1
  }
```

Definición de los vectores de interrupción

El archivo en ensamblador donde se definen los vectores de interrupción se muestra a continuación:

```
*   vectores.asm

;definición de los vectores de interrupción

    .mmregs
    .sect    " vectors"  ;sección definida en el archivo .cmd
```

```
    .ref     _c_int00    ;referencia al punto de entrada
    .ref     _recibir    ;referencia a la subrutina de interrupción

reset:  B   _c_int00
        nop
        nop
nmi:    rete  ; interrupción no mascarable
        nop
        nop
        nop
sint17: rete  ;interrupción por software
        nop
        nop
        nop
sint18: rete  ;interrupción por software
        nop
        nop
        nop
sint19: rete  ;interrupción por software
        nop
        nop
        nop
sint20: rete  ;interrupción por software
        nop
        nop
        nop
sint21: rete  ;interrupción por software
        nop
        nop
        nop
sint22: rete  ;interrupción por software
        nop
        nop
        nop
sint23: rete  ;interrupción por software
        nop
        nop
```

```
        nop
sint24: rete  ;interrupción por software
        nop
        nop
        nop
sint25: rete  ;interrupción por software
        nop
        nop
        nop
sint26: rete  ;interrupción por software
        nop
        nop
        nop
sint27: rete  ;interrupción por software
        nop
        nop
        nop
sint28: rete  ;interrupción por software
        nop
        nop
        nop
sint29: rete  ;interrupción por software
        nop
        nop
        nop
sint30: rete ;interrupción por software
        nop
        nop
        nop
int0:   rete ;interrupción externa 0
        nop
        nop
        nop
int1:   rete ;interrupción externa 1
        nop
        nop
        nop
```

```
int2:   rete ;interrupción externa 2
        nop
        nop
        nop
tint0:  rete ;interrupción timer 0
        nop
        nop
        nop
brint0: rete ;int puerto serial 0 de recepción
        nop
        nop
        nop
bxint0: rete ;int puerto serial 0 de transmisión
        nop
        nop
        nop
dmac0:  rete ;int canal DMA 0
        nop
        nop
        nop
tint1:  rete ;int timer 1
        nop
        nop
        nop
int3:   rete ;int externa 3
        nop
        nop
        nop
hpint:  rete ; int puerto HPI
        nop
        nop
        nop
brint1: b _recibir  ; cuando se produce una interrupción debida a
        nop         ; la recepción del puerto serial 1 se ejecuta la función
        nop         ; recibir()
bxint1: rete  ;int puerto serial 1 de transmisión
        nop
```

```
        nop
        nop
dmac4:  rete  ;int canal DMA 4
        nop
        nop
        nop
dmac5:  rete ;int canal DMA 5
        nop
        nop
        nop
       .end
```

8.6. Requerimientos del Sistema

Con las consideraciones anteriormente presentadas mostramos la disposición de memoria requerida para el diseño realizado del sistema de síntesis de voz en tiempo real (cuadro 8.2), esto, con base al mapa de memoria del DSP C5402 [44].

Bloque	Memoria dato	memoria programa	Porcentaje (%)
rts.lib	34	2007	12.46
dsk5402.lib	60	5660	34.91
drv5402.lib	30	75	0.64
pila	1024	-	6.25
Constantes	488	-	2.98
Código	1323	1553	17.55
Total	2959	9295	74.79

Cuadro 8.2: Memoria dato y programa requerida en el diseño.

El bloque de constantes se refiere a los arreglos asignados a la ventana de Hamming, al ruido blanco y valores contantes necesarios en los procesos de análisis y síntesis. En cuanto al bloque de código (memoria programa), este alberga las funciones y subrutinas de los procesos de análisis, síntesis y subrutina de la llamada a atención de interrupción, mientras que en la memoria dato requerida en el bloque de código se definen las variables y arreglos temporales necesarios. En el cuadro 8.3 se presenta los requerimientos de memoria programa desglosados para las funciones y subrutinas que integran al código del diseño.

Subproceso	Memoria programa
Proceso de ventaneo y traslape	140
Proceso de umbral de silencio	82
Decisión de la naturaleza y estimación del pitch	248
Filtro de preénfasis	56
Recursión de Levinson - Durbin	252
Filtro todo polo	184
Generador de tren de impulsos o lectura del ruido	95
Filtro de deénfasis	56
Subrutina de interrupción	57
Función de autocorrelación	79

Cuadro 8.3: Memoria programa requerida para cada proceso.

8.7. Algunos Resultados

Se presenta una prueba para comparar el desempeño del sistema de síntesis de voz empleando el DSP con respecto al resultado obtenido al emplear aritmética de punto flotante. La señal de voz real usada fue la palabra "rojo" muestreada a una frecuencia de f_s= 8 kHz. En la figura 8.12 se presenta la simulación tanto en punto flotante como en punto fijo, observando la gran similitud en el dominio temporal con relación a la señal original. En el cuadro 8.4 se muestran los tiempos de ejecución para cada subproceso que integran a los procesos de análisis y síntesis, para un segmento de voz de $N = 240$ muestras. El ciclo de instrucción del C5402 es de 10 ns por lo que si muestreamos a una f_s=8000 Hz, el tiempo en adquirir 240 muestras es de 30 ms. En el cuadro 8.4 se observa que los procesos de análisis y síntesis tienen un tiempo de ejecución de 1.0473 ms y 3.5474 ms, respectivamente. Así, los procesos de análisis - síntesis se ejecutan en un 15.31 % del tiempo de adquisición para N=240 muestras. Lo anterior representa que no habrá conflictos entre los tiempos de cálculo de los procesos de análisis - síntesis entre ventanas sucesivas.

Resumen

En este ejemplo de aplicación se presentó las estrategias y el diseño para un sistema de síntesis de voz en tiempo real considerando su implantación en una arquitectura de punto fijo

Subproceso	Tiempo (ms)
proceso de ventaneo	0.0762
proceso de estimación de pitch	0.527
Filtrado de preénfasis	0.0835
Vector de Autocorrelación	0.0356
Recursión de Levinson - Durbin	0.325
Generador de tren de impulsos y lectura de ruido	0.051
Filtro todo polo	3.42
Filtrado de deénfasis	0.0764

Cuadro 8.4: Tiempos de ejecución para cada subproceso en el DSP.

(TMS320C5402). De los resultados obtenidos se observa que la comparación entre aritmética de punto flotante y fija, no difieren en gran manera, sólo existe error del 6.6 % debido a cuestiones de precisión numérica, que no afecta la calidad de la voz al momento de reconstruirse. La compresión de la señal de voz original utilizando el método de síntesis, se alcanzó hasta en un 7.7 %, es decir, de 8000 muestras de voz adquiridas en un segundo, sólo se requerirán 616 parámetros/s para su posible transmisión y/o almacenamiento.
El diseño se efectuó en tiempo real con tiempos de ejecución de un 15.31 % del tiempo de adquisición para N=240 muestras, garantizando un desempeño óptimo y ser válida nuestra implantación como un bloque modular que puede ser partícipe en sistemas de mayor complejidad.
Con la arquitectura del DSP TMS320C5402 se tiene la opción de integrar nuestro sistema de síntesis de voz a sistemas de reconocimiento, seguridad o almacenamiento masivo puesto que el 85 % del tiempo de ejecución por ventana está libre (25.41 ms). Por tanto, no existen conflictos en tiempos, carga computacional y precisión numérica para implantar lo anterior.

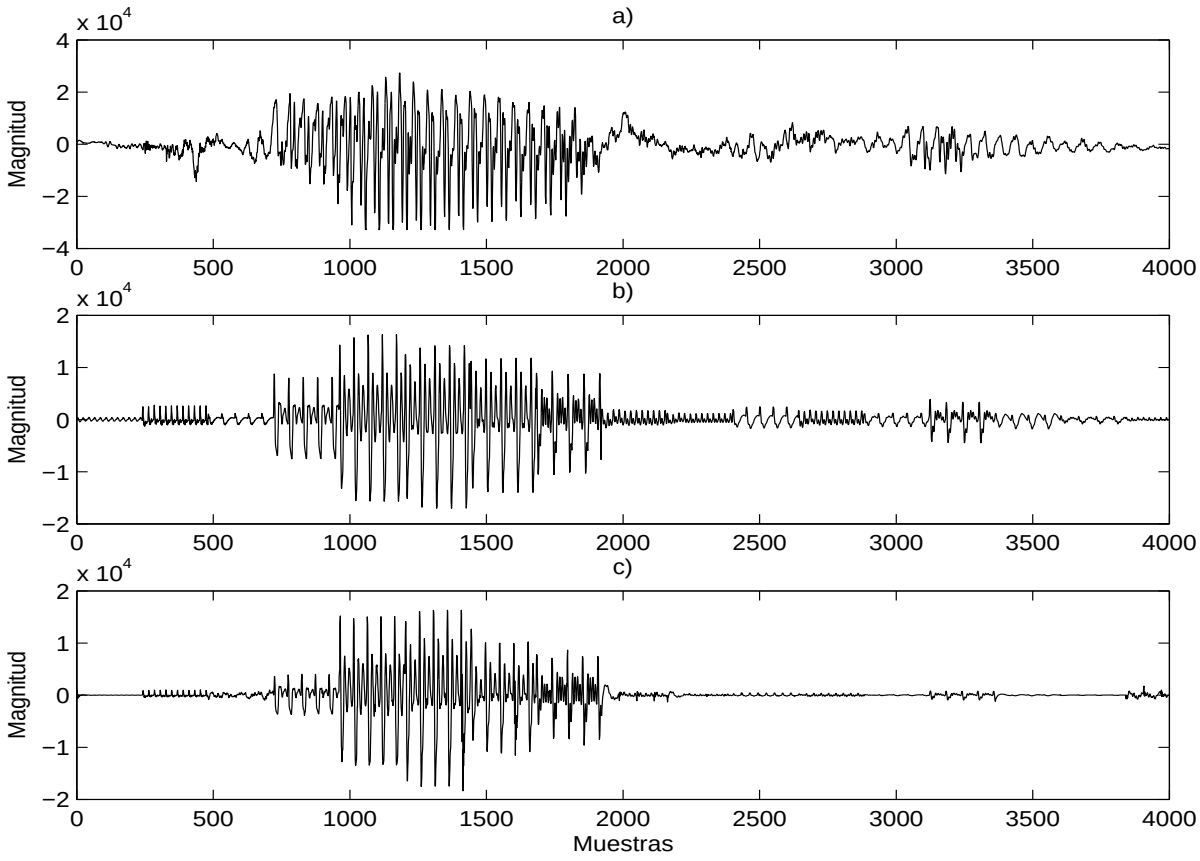


Figura 8.12: Simulación entre aritmética de punto flotante y fijo. a) señal original, b) señal sintetizada en punto flotante, c) señal sintetizada en punto fijo.

Bibliografía

[1] Alcántara S. R. & Escobar S. L. *Dynamic Range and Scaling Evaluation in Adaptive Filtering Algorithms for DSP fixed-point implementation.* International Symposium on Information Theory and its Applications (ISITA98). México, octuber 14-16, 1998.

[2] Alcántara S. R. & Escobar S. L. *A comparative TMS DSP Fixed-Point Implementation of FRLS Adaptive Filtering Algorithms Family.* The International Conference on Signal Processing Applications and Technology (ICSPAT). November 1-4, 1999. Orlando, Florida USA.

[3] Molero A. M. & Barajas P. G., "Síntesis de Voz en Tiempo Real empleando una arquitectura DSP", Tesis de Licenciatura, Facultad de Ingeniería, UNAM 2004.

[4] Deller John R., Proakis John & Hansen John "Discrete-Time Processing of Speech Signals, MacMillan, EUA 1993.

[5] Escobar S. L. & Rodríguez S. R. *Diseño y construcción de Arquitectura para Procesamiento Digital de Imágenes.* F.I. UNAM. 1992. Tesis de Licenciatura.

[6] Escobar S. L. *Algoritmos de Filtrado Adaptable: Implementación, Evaluación, Comparación y Aplicaciones en Telecomunicaciones.* F.I. UNAM. 1997. Tesis de Maestría.

[7] Escobar S. L. & Pasalaguas A. *Síntesis de voz en tiempo real utilizando una arquitectura de punto flotante.* Congreso de Electrónica. Instituto Tecnológico de Chihuahua. Chihuahua, México, octubre de 1997.

[8] Escobar S. L. *Arquitecturas de DSPs, familia TMS320 y el TMS20C50.* Facultad de Ingeniería, UNAM, México D. F., agosto del 2000.

[9] Escobar S. L. & Alcántara S. R. *Fixed Point Arithmetic using Digital Signal Procesors*. International Conference on Telecommunications (ICT200), Acapulco, México 22-25 de mayo del 2000.

[10] Escobar S. L. *Manual del laboratorio de Procesamiento Digital de Señales*. Facultad de Ingeniería, UNAM, México D. F., abril del 2000.

[11] Escobar S. L. *Laboratorio DSPs, familias TMS320C5x y TMS320C54x*. Facultad de Ingeniería, UNAM, México D. F., junio del 2002.

[12] Furui Sadaoki "Digital Speech Processing, Synthesis, and Recognition ". Marcel Dekker Ed. EUA 2001.

[13] Hayes H. Monson. "Statistical Digital Signal Processing and Modeling", John Wiley & Sons, Inc. EUA 1996.

[14] Hamming R. W. *Digital filters*. Prentice Hall, New Jersey 1983.

[15] Lee. E. A. *Arquitectura programable DSP*. IEEE ASSP MAGAZINE octubre de 1988 y enero 1989.

[16] Makhoul J. "Linear Prediction: A tutorial Review"Proc. IEEE, Vol 63, April 1975: 561-580.

[17] Markel, J. D. & Gray A. H. "Linear Prediction of Speech", Springer - Verlag, EUA 1976.

[18] Martínez J. & Escobar S. L. & Rodríguez S. R. *Arquitectura para Procesamiento digital de Imágenes. Congreso de Electrónica*. Instituto Tecnológico de Chihuahua, Chihuahua México, octubre de 1993.

[19] Oppenheim A. V. & Schafer R. W. *Digital Signal Processing*. Prentice-Hall, Englewood Cliffs, New Jersey 1975.

[20] Owens F. J. "Signal Processing of Speech", Mc Graw Hill Inc, EUA 1993.

[21] Papamichalis E. Panos "Practical Approaches to Speech Coding", Prentice - Hall and Texas Instrument Digital Signal Processor Series.

[22] Proakis John & Manolakis Dimitris "Tratamiento Digital de Señales: Principios, algoritmos y aplicaciones", Prentice Hall, 1998

[23] PRADO J. & ALCÁNTARA S. R. *A Fast Square-Rooting Algorithm Using a Digital Signal Processor*. Proceedings of IEEE, USA, Vol. 75, No.2, pg. 262-264, Feb. 1987.

[24] PROAKIS J.G. & MANOLAKIS. *Digital Signal Processing*. Macmillan. Singapure 1992.

[25] PSENICKA B. *Apuntes de Procesamiento Digital de Señales*. Facultad de Ingeniería, UNAM, México D.F., 1996.

[26] PSENICKA B. Y ESCOBAR S. L. *Procesamiento Digital de Señales, segunda parte, Microcontroladores y realización de los filtros digitales con TMS320Cxx*. Facultad de Ingeniería, UNAM, México D.F., julio de 1998.

[27] RABINER LAWRENCE & SCHAFER RONALD W. "Digital Processing of Speech Signals"Prentice-Hall Inc. EUA 1978.

[28] SCILAB GROUP. *Signal Processing with Scilab*. INRIA Meta2 proyect/ENPC Cergrene, París France. scilab@inria.fr.

[29] TEXAS INSTRUMENTS *Texas Instruments. Digital Signal Processing Applications with the TMS320 Family, Theory, Algorithms, and implementations*. Vol.1,2 y 3. USA 1990.

[30] TEXAS INSTRUMENTS.*TMS30C1x, User's Guide*. USA 1990.

[31] TEXAS INSTRUMENTS.*TMS30C2x, User's Guide*. USA 1991.

[32] TEXAS INSTRUMENTS. *TMS30C3x, User's Guide*. USA 1992.

[33] TEXAS INSTRUMENTS. *TMS30C5x, User's Guide*. USA 1997.

[34] TEXAS INSTRUMENTS. *TMS320C54x Reference Set, Vols 1: CPU and Peripherals, 2: Mnemonic Instruction Set, 3: Algebraic Instruction Set & 4: Applications Guide*. USA 1997.

[35] TMS320C5402 DSK Help (SPRH075A) Copyright© 1998-1999 Texas Instruments

[36] TEXAS INSTRUMENTS. Code Composer User's Guide. Literature number SPRU296.

[37] TEXAS INSTRUMENTS. Code Composer Studio User's Guide. literature number SPRU328.

[38] TEXAS INSTRUMENTS, "Optimized DSP Library for C Programmers on the TMS320C54x", (SPRA480A), Application Report.

[39] TEXAS INSTRUMENTS, "TMS320C54x DSP Reference Set, Volume 2: Mnemonic Instruction Set", literature number SPRU172.

[40] TEXAS INSTRUMENTS, "TMS320C54x DSP Reference Set, Volume 3: Algebraic Instruction Set", literature number SPRU179.

[41] TEXAS INSTRUMENTS, "TMS320C54x Assembly Language Tools User's Guide",literature number SPRU103.

[42] TEXAS INSTRUMENTS, "TMS320C54x Optimizing C Compiler User's Guide", literature number SPRU102.

[43] TEXAS INSTRUMENTS, "TMS320C54x C Source Debugger User's Guide", literature number SPRU099.

[44] TEXAS INSTRUMENT, TMS320C54x DSP Reference Set, Volume 1: CPU and Peripherals (literature number SPRU131).

[45] TEXAS INSTRUMENTS, TMS320C54x DSP Reference Set, Volume 5: Enhanced Peripherals (literature number SPRU302).

[46] TEXAS INSTRUMENTS, TMS320C54x DSP Reference Set, Volume 5: Enhanced Peripherals (literature number SPRU302).

[47] WIDROW & STEARNS. *Adaptive signal processing*. Prentice-Hall, New Jersey 1985.