Tema 4
Programacion y Aplicaciones de los Procesadores Digitales de Señal

4.1. Programación del TMS320C2x

La programación de la familia TMS320C2x está fuertemente influenciada por su objetivo de implementar algoritmos de procesado digital de señales en tiempo real, lo que lleva asociada una optimización en cuanto a velocidad de ejecución de instrucciones individuales y gran capacidad de E/S.

La *pipeline* implementada en el procesador permite ejecutar la mayoría de las instrucciones (97 de 133) en un solo ciclo de instrucción. Las instrucciones que requieren ciclos adicionales comprenden saltos, llamadas y retornos (las cuales implican una ruptura en la *pipeline*), instrucciones con código de 2 palabras, E/S e instrucciones que permiten operaciones paralelas del procesador. No obstante, algunas de ellas se ejecutan en un solo ciclo cuando se utilizan conjuntamente con el contador de repetición.

4.1.1. Modos de direccionamiento de memoria

Tal y como se vio en el capítulo anterior, el TMS320C2x dispone de estructuras *hardware* que le permiten implementar diferentes modos de direccionamiento. Podemos distinguir tres tipos principales:

Direccionamiento directo

Aunque ésta es la denominación de Texas Instruments, en realidad se trata de un direccionamiento directo por página de memoria. En este modo, el campo de dirección de la palabra de instrucción contiene los 7 bits bajos de la dirección de memoria de datos (*data address memory*: dma). Este campo se concatena con los 9 bits del registro puntero de página de memoria de datos (DP) para formar la dirección completa de 16 bits. El DP apunta a una de las 512 posibles páginas de 128 palabras cada una, y los 7 bits de la instrucción especifican la posición dentro de la página (*offset*).

JUAN F. GUERRERO MARTÍNEZ
JOSÉ V. FRANCÉS VILLORA
Curso 2010-2011

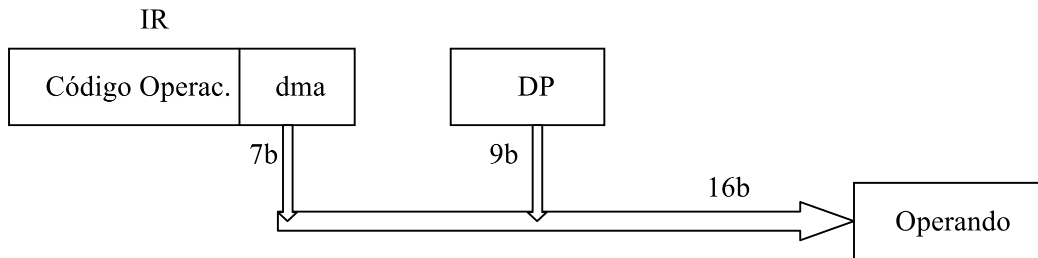


Figura 4.1. Direccionamiento directo del TMS320C2x.

La sintaxis y el formato de direccionamiento directo son:

Instrucción dma [,shift]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Código operación								0	dma						

Los bits 8 a 15 contienen el código de operación (*opcode*). B7=0 define el modo de direccionamiento como directo. Los bits 6 a 0 contienen la dirección de memoria. Opcionalmente puede controlar también el funcionamiento del desplazador de escalado, provocando un desplazamiento (*shift*) de n bits a la izquierda durante la carga del operando en la ALU.

Como ejemplo de direccionamiento directo, la siguiente figura muestra la ejecución de la instrucción ADD X,2 que suma al acumulador el contenido de la variable X en memoria, desplazándola a la izquierda 2 bits:

$$ACC = ACC + X \cdot 2^2 = 1 + 2 \cdot 2^2 = 9$$

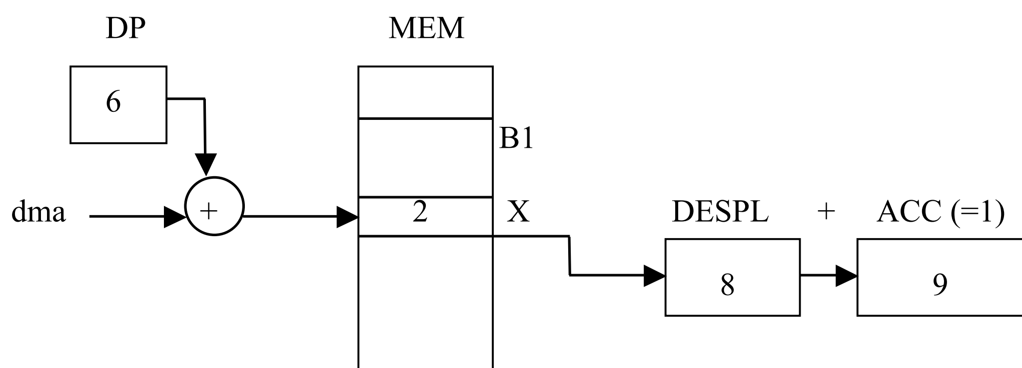


Figura 4.2. Ejemplo de direccionamiento directo.

Direccionamiento indirecto

También aquí se trata en realidad de un direccionamiento indirecto por registro, utilizando los registros auxiliares (AR). En este modo puede accederse a cualquier posición del espacio de memoria de 64 K mediante las direcciones de 16 bits contenidas en los AR. Para seleccionar un AR determinado, se carga el puntero de registro auxiliar (ARP) con un valor de 0 a 7. Un campo del IR permite modificar el valor del registro ARP. El AR actual seleccionado por el ARP direcciona la memoria de datos.

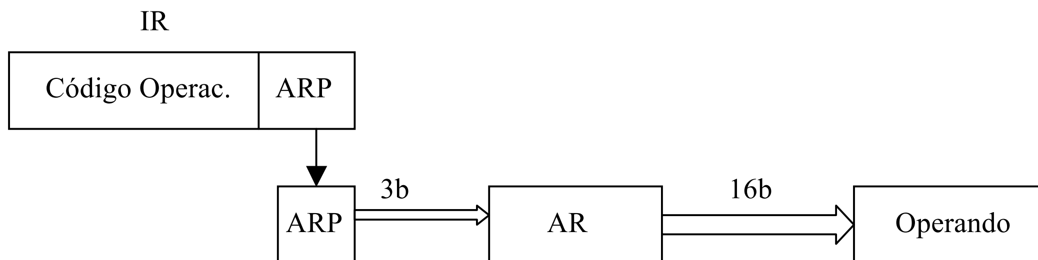


Figura 4.3. Direccionamiento indirecto del TMS320C2x.

El DSP soporta dos tipos de direccionamiento indirecto:

- Direccionamiento indirecto regular con incremento o decremento.
- Direccionamiento indirecto con indexado basado en el contenido de AR0:
    - Indexado sumando o restando su contenido.
    - Indexado sumando o restando su contenido con propagación de acarreo invertida.

La sintaxis y el formato del direccionamiento indirecto son:

Instrucción ind [,shift [,next ARP]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Código operación								1	idv	inc	dec	nar	Y		

Los bits 8 a 15 contienen el código de operación. B7=1 define el modo de direccionamiento como indirecto. Los bits 6 a 0 contienen los bits de control del direccionamiento indirecto, codificándose el campo *ind* de la instrucción. Opcionalmente se puede controlar el desplazamiento en bits, así como la actualización con un nuevo valor del registro ARP después de la ejecución de la instrucción, permitiendo cambiar el registro activo AR en el mismo ciclo sin necesidad de una nueva instrucción. El significado de estos bits es:

B6: define modo indexado o no.
B5: incremento.
B4: decremento
(B5-B4 definen conjuntamente modo reversión acarreo)
B3: nuevo AR. En este caso, el nuevo valor de ARP está en el campo Y.

Los siguientes signos se utilizan en direccionamiento indirecto para especificar el campo *ind* de la instrucción:

*	El contenido de AR(ARP) se utiliza como dma.
*-	El contenido de AR(ARP) se utiliza como dma, y es decrementado después del acceso.
*+	El contenido de AR(ARP) se utiliza como dma, y es incrementado después del acceso.
*0-	El contenido de AR(ARP) se utiliza como dma, y se le resta el contenido de AR0 después del acceso.
*0+	El contenido de AR(ARP) se utiliza como dma, y se le suma el contenido de AR0 después del acceso.
*BR0-	El contenido de AR(ARP) se utiliza como dma, y se le resta el contenido de AR0, con propagación de acarreo invertida, después del acceso.
*BR0+	El contenido de AR(ARP) se utiliza como dma, y se le suma el contenido de AR0, con propagación de acarreo invertida, después del acceso.

Para comprender la utilidad de cambiar ARP, volvamos al ejemplo de la instrucción ADD. Aunque generalmente los AR se utilizan como punteros de memoria, pueden utilizarse también como contadores de bucle. Esto es necesario cuando los bucles deben contener más de una instrucción, deben ejecutarse más de 256 veces (el contador *hardware* de bucle es de 8 bits) o cuando hay varios bucles anidados.

En este caso, supongamos que AR1 se utiliza para contener el puntero de acceso a memoria, con valor inicial Comienzo_Memoria, y AR2 es un contador de bucle, con valor incial N. Supongamos también inicialmente ARP=1. La siguiente secuencia de programa suma el contenido (sin desplazamiento) de las N posiciones de memoria a partir de la dirección Comienzo_Memoria.

```
BUCLE   ADD     *+,0,2        ; ARP=2
        BANZ    BUCLE,*-,1    ;ARP=1
```

Por tanto, y sin instrucciones accesorias que consumirían ciclos de reloj, se conmuta en cada momento al registro auxiliar necesario.

Ejemplos:

1. Componer una instrucción en ensamblador del TMS320C2x que realice:
   ACC = ACC + 16·X
   DIR(X) = DIR(X) + 1 (actualizar puntero de direcciones).

ADD *+,4 Suma al acc el contenido de la posic. de memoria definida por el contenido del AR actual (X). Este dato se desplaza a la izquierda 4 bits antes de ser sumado. El AR se incrementa.

2. Componer una instrucción en ensamblador del TMS320C2x que realice:
   ACC = ACC + 16·X
   DIR(X) = DIR(X) - 20 (actualizar puntero de direcciones: acceso indexado).

ADD *0-,4 Suma al acc el contenido de la posic. de memoria definida por el contenido del AR actual. Este dato se desplaza a la izquierda 4 bits antes de ser sumado. El contenido de AR se decrementa en el contenido de AR0 (20).

Direccionamiento inmediato

En este caso, la instrucción (una o más palabras) contiene el valor del operando inmediato. El TMS320C2x soporta este caso con una sola palabra (*short*) o dos palabras (*long*). La extensión K o LK definen estos dos tipos en los nemónicos de las operaciones.

La sintaxis y el formato del direccionamiento inmediato corto son:

Instrucción constante

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Código operación								constante							

Un ejemplo sería la operación RPTK 99, que ejecuta la siguiente instrucción 100 veces.

Para direccionamiento inmediato largo, la constante es de 16 bits, y sigue al código de operación. La constante puede usarse opcionalmente como entero con o sin signo. Un ejemplo sería ADLK 16384 que suma al ACC el valor de la constante.

4.1.2. Conjunto de instrucciones

Podemos agrupar las instrucciones de la familia TMS320C2x en los siguientes grupos:

ACUMULADOR Y ALU

- Carga y almacenamiento (LAC, SACH, SACK, ZALH, ZALR, etc.)
- Desplazamientos y rotaciones del acumulador (ROL, ROR, SFL, SFR).
- Aritméticas (valor absoluto ABS, sumas ADD, restas SUB, negaciones NEG, etc.).
- Lógicas (AND, OR, XOR).

MULTIPLICADOR

- Carga y almacenamiento de P desde memoria (LPH,SPH,SPL).
- Carga y acumulación de P en ACC (APAC, SPAC, PAC)
- Carga de T y acumulación (LT,LTA,LTD,LTP,LTS)
- Multiplicación (MAC, MACD, MPYA, MPYS, etc.)
- Elevar al cuadrado (SQRA,SQRS).

REGISTROS AUXILIARES Y DP

- Modificación de ARs (MAR, ADRK, SBRK, etc.)
- Carga y almacenamiento (LAR, LARK, LRLK, SAR, etc.)
- Carga ARP (LARP)
- Carga DP (LDP, LDPK)

SALTOS Y LLAMADAS A SUBRUTINAS

- Saltos condicionales e incondicionales.
- Llamadas y retonos de subrutinas (CALA,CALL,RET)

MEMORIA Y E/S

- Movimiento de bloques (BLKD, BLKP, DMOV, TBLR, TBLW)
- E/S (IN, OUT)
- Puerto serie (RFSM, RTXM, ...)

CONTROL

- Registros estado (LST, LST1, SST, SST1)
- Control *flags* (SOVM, SSXM, ...)
- Interrupciones (DINT, EINT, TRAP)
- Control bucle (RPT, RPTK)
- Pila (POP, PUSH)
- Configuración B0 (CNFD, CNFP)
- Test bits (BIT, BITT, STC ...)
- Modo bajo consumo (IDLE)

El conjunto de instrucciones del TMS320C26 es prácticamente el mismo que el del TMS320CC25. La única diferencia estriba en la configuración de los bloques internos de memoria RAM como memoria de programa mediante las instrucciones CONF i, donde i puede tomar los valores 0, 1 ó 3.

Las siguientes tablas contienen, para cada instrucción, el mnemónico, su sintaxis, una breve descripción y las operaciones realizadas durante su ejecución. En aquellas instrucciones que admiten direccionamiento directo e indirecto, se dan las sintaxis correspondientes a ambos modos, con el directo en primer lugar. Los corchetes indican que el parámetro encerrado es opcional. Los símbolos utilizados son:

Símbolo	Descripción	Símbolo	Descripción
ACC	Registro acumulador	ST0	Registro de estado 0
ACCH	Parte alta registro acumulador	ST1	Registro de estado 1
ACCL	Parte baja registro acumulador	XF	Salida de estado
P	Registro P	BIO	Entrada de control de salto
Shifted P	Registro P desplazado por desplazador de salida del multiplicador	TXM	Bit de modo de transmisión
T	Registro T	FSM	Bit de modo de sincronización
AR	Registro auxiliar	FO	Bit de formato de transmisión
ARP	Registro puntero registros auxiliares	CNF	Bits de control de configuración de memoria
ARB	Registro buffer de ARP	INTM	Bit de máscara de interrupción
PC	Registro contador programa	C	Bit de acarreo
RTPC	Contador de repetición de bucle	OV	Bit de desbordamiento
DP	Registro puntero página datos	HM	Bit de modo Hold
dma	Dirección de memoria de datos	OVM	Bit de modo saturación
pma	Dirección de memoria de programa	SXM	Bit de extensión de signo
shift	Valor de desplazamiento	TC	Bit de control de test
Next ARP	Siguiente valor de ARP		

Por ejemplo, en el caso de la instrucción ADD, al contenido previo del acumulador se suma el contenido de la dirección de memoria de datos desplazada, y el resultado se guarda en el ACC. Tenemos:

Instr.	Sintaxis	Descripción	Operación
ADD	ADD dma[,shift] ADD ind[,shift[,next ARP]]	Sumar al ACC con desplazamiento	$(ACC)+[(dma)\cdot 2^{shift}] \rightarrow ACC$

- <u>Opciones de sintaxis</u>:
  - Direccionamiento directo:
    - ADD dma
    - ADD dma, shift
  - Direccionamiento indirecto:
    - ADD ind
    - ADD ind, shift
    - ADD ind, shift, next ARP
- <u>Operación</u>:

- Símbolos con paréntesis: contenidos de registros o memoria.
- Símbolos sin paréntesis: direccioness de registros o memoria.

→ : indica destino del resultado de la operación.

ACUMULADOR Y ALU			
Instr.	**Sintaxis**	**Descripción**	**Operación**
ABS	ABS	Valor absoluto del ACC	$\|ACC\| \rightarrow ACC$
ADD	ADD dma[,shift] ADD ind[,shift[,next ARP]]	Sumar al ACC con desplazamiento	$(ACC)+[(dma)\cdot 2^{shift}] \rightarrow ACC$
ADDC	ADDC dma ADDC ind[,next ARP]	Sumar al ACC con acarreo	$(ACC)+(dma)+(C) \rightarrow ACC$
ADDH	ADDH dma ADDH ind[,next ARP]	Sumar al ACC alto	$(ACC)+[(dma)\cdot 2^{16}] \rightarrow ACC$
ADDK	ADDK constante	Sumar al ACC constante en formato inmediato corto	(ACC)+ constante 8 bits→ ACC
ADDS	ADDS dma ADDS ind[,next ARP]	Sumar al ACC sin extensión de signo	$(ACC)+(dma) \rightarrow ACC$
ADDT	ADDT dma ADDT ind[,next ARP]	Sumar al ACC con desplazamiento especificado por registro T	$(ACC)+[(dma)\cdot 2^{(Treg)}] \rightarrow ACC$
ADLK	ADLK constante[,shift]	Sumar al ACC constante en formato inmediato largo con desplazamiento	(ACC)+ [constante 16 bits $\cdot 2^{shift}$] → ACC
AND	AND dma AND ind[,next ARP]	AND con ACC	(ACC(15-0)) AND (dma) → ACCL. ACCH=0.
ANDK	ANDK constante[,shift]	AND inmediato con ACC, con desplazamiento	(ACC(30-0)) AND [constante 16 bits $\cdot 2^{shift}$] → ACC(30-0). 0 → ACC(31)=0.
CMPL	CMPL	Complementar ACC	NOT(ACC) → ACC
LAC	LAC dma[,shift] LAC ind[,shift[,next ARP]]	Cargar ACC con desplazamiento	$(dma)\cdot 2^{shift} \rightarrow ACC$
LACK	LACK constante	Cargar ACC con constante en formato inmediato corto	constante 8 bits→ ACC
LACT	LACT dma LACT ind[,next ARP]	Cargar ACC con desplazamiento especificado por registro T	$(dma)\cdot 2^{(Treg)} \rightarrow ACC$
LALK	LALK constante[,shift]	Cargar ACC con constante en formato inmediato largo con desplazamiento	(constante 16 bits) $\cdot 2^{shift}$→ ACC
NEG	NEG	Negar ACC	-(ACC) → ACC
NORM	NORM ind	Normaliza ACC (convierte formato coma fija a coma flotante)	Si (ACC) = 0, 1 → TC. Si no, { si (ACC(31))XOR(ACC(30))=0 0 → TC, (ACC)·2 → ACC. si no, 1 → TC.}
OR	OR dma OR ind[,next ARP]	OR con ACC	(ACC(15-0)) OR (dma) → ACC(15-0).
ORK	ORK constante[,shift]	OR inmediato con ACC, con	(ACC(30-0)) OR [constante 16

		desplazamiento	bits $\cdot 2^{shift}$] → ACC(30-0).
ROL	ROL	Rotación ACC a izquierda	(ACC(30-0)) → ACC(31-1). (C)→ACC(0). ACC(31)→(C).
ROR	ROR	Rotación ACC a derecha	(ACC(31-1)) → ACC(30-0). ACC(0) → (C).
SACH	SACH dma[,shift] SACH ind[,shift[,next ARP]]	Almacenar ACC alto con desplazamiento	[(ACC)$\cdot 2^{shift}$] → dma
SACL	SACL dma[,shift] SACL ind[,shift[,next ARP]]	Almacenar ACC bajo con desplazamiento	[(ACCL)$\cdot 2^{shift}$] → dma
SBLK	SBLK constante[,shift]	Restar al ACC constante en formato inmediato largo con desplazamiento	(ACC) - [constante 16 bits $\cdot 2^{shift}$] → ACC
SFL	SFL	Desplazar ACC a izquierda	(ACC(30-0)) → ACC(31-1). 0 → ACC(0).
SFR	SFR	Desplazar ACC a derecha	(ACC(31-1)) → ACC(30-0). (ACC(31)) → ACC(31).
SUB	SUB dma[,shift] SUB ind[,shift[,next ARP]]	Restar del ACC con desplazamiento	(ACC)-[(dma)$\cdot 2^{shift}$] → ACC
SUBB	SUBB dma SUBB ind[,next ARP]	Restar del ACC llevando	(ACC)-(dma)-NOT(C) → ACC
SUBC	SUBC dma SUBC ind[,next ARP]	Resta condicional del ACC	Si ((ACC)-[(dma)$\cdot 2^{shift}$]) ≥ 0 ((ACC)-[(dma)$\cdot 2^{shift}$])·2→ ACC si no (ACC)·2→ ACC.
SUBH	SUBH dma SUBH ind[,next ARP]	Restar del ACC alto	(ACC)-[(dma)$\cdot 2^{16}$] → ACC
SUBK	SUBK constante	Restar del ACC constante en formato inmediato corto	(ACC) - constante 8 bits→ ACC
SUBS	SUBH dma SUBH ind[,next ARP]	Sumar al ACC sin extensión de signo	(ACC)+(dma) → ACC
SUBT	SUBT dma SUBT ind[,next ARP]	Restar del ACC con desplazamiento especificado por registro T	(ACC)-[(dma)$\cdot 2^{(Treg)}$] → ACC
XOR	XOR dma XOR ind[,next ARP]	OR exclusiva con ACC	(ACC(15-0)) XOR (dma) → ACC(15-0).
XORK	XORK constante[,shift]	XOR inmediato con ACC, con desplazamiento	(ACC(30-0)) XOR [constante 16 bits $\cdot 2^{shift}$] → ACC(30-0).
ZAC	ZAC	Poner a cero ACC	0 → ACC
ZALH	ZALH dma ZALH ind[,next ARP]	Poner a cero ACC bajo y cargar ACC alto	(dma)$\cdot 2^{16}$→ ACCH ACCL=0.
ZALR	ZALR dma ZALR ind[,next ARP]	Poner a cero ACC bajo y cargar ACC alto con redondeo	(dma) → ACCH 8000h → ACCL
ZALS	ZALS dma ZALS ind[,next ARP]	Poner a cero ACC y cargar ACC bajo sin extensión de signo	(dma) → ACCL ACCH=0.

MULTIPLICADOR			
Instr.	**Sintaxis**	**Descripción**	**Operación**
APAC	APAC	Sumar P al ACC	(ACC)+(shifted P) → ACC
LPH	LPH dma LPH ind[,next ARP]]	Cargar P alto	(dma) → P(31-16)
LT	LT dma LT ind[,next ARP]]	Cargar registro T	(dma) → T
LTA	LTA dma LTA ind[,next ARP]]	Cargar registro T y acumular producto previo	(dma) → T (ACC)+(shifted P) → ACC
LTD	LTD dma LTD ind[,next ARP]]	Cargar registro T, acumular producto previo y mover datos	(dma) → T (ACC)+(shifted P) → ACC (dma) → dma + 1
LTP	LTP dma LTP ind[,next ARP]]	Cargar registro T y cargar P en ACC	(dma) → T (shifted P) → ACC
LTS	LTS dma LTS ind[,next ARP]]	Cargar registro T y restar producto previo	(dma) → T (ACC)-(shifted P) → ACC
MAC	MAC pma, dma MAC pma, ind[,next ARP]]	Multiplicar y acumular producto previo	(ACC)+(shifted P) → ACC (pma) · (dma) → P
MACD	MACD pma, dma MACD pma, ind[,next ARP]]	Multiplicar, acumular producto previo y mover datos	(ACC)+(shifted P) → ACC (pma) · (dma) → P (dma) → dma + 1
MPY	MPY dma MPY ind[,next ARP]]	Multiplicar por registro T	(T) · (dma) → P
MPYA	MPYA dma MPYA ind[,next ARP]]	Multiplicar por registro T y acumular producto previo	(ACC) + (shifted P) → ACC (T) · (dma) → P
MPYK	MPYK constante	Multiplicar registro T por constante en formato inmediato corto	(T) · constante 13 bits → P
MPYS	MPYS dma MPYS ind[,next ARP]]	Multiplicar por registro T y restar producto previo	(ACC) - (shifted P) → ACC (T) · (dma) → P
MPYU	MPYU dma MPYU ind[,next ARP]]	Multiplicar por registro T sin extensión de signo	(sin signo): (T) · (dma) → P
PAC	PAC	Cargar P en ACC	(shifted P) → ACC
SPAC	SPAC	Restar registro P del ACC	(ACC)-(shifted P) → ACC
SPH	SPH dma SPH ind[,next ARP]]	Guardar registro P alto	(shifted P(31-16)) → dma
SPL	SPL dma SPL ind[,next ARP]]	Guardar registro P bajo	(shifted P(15-0)) → dma
SPM	SPM constante	Define funcionamiento del desplazador de P	constante 2 bits → PM
SQRA	SQRA dma SQRA ind[,next ARP]]	Elevar al cuadrado y acumular producto previo	(ACC) + (shifted P) → ACC (dma) · (dma) → P
SQRS	SQRA dma SQRA ind[,next ARP]]	Elevar al cuadrado y restar producto previo	(ACC) - (shifted P) → ACC (dma) · (dma) → P

REGISTROS AUXILIARES Y DP			
Instr.	**Sintaxis**	**Descripción**	**Operación**

ADRK	ADRK constante	Sumar a AR activo constante en formato inmediato corto	(AR) + constante 8 bits→ AR
CMPR	CMPR constante	Comparar AR con AR0	Comparación (AR,AR0) → TC
LAR	LAR AR,dma LAR AR,ind[,next ARP]	Cargar AR	(dma) → AR
LARK	LARK AR, constante	Cargar AR con constante en formato inmediato corto	constante 8 bits→ AR
LARP	LARP constante	Cargar registro puntero de AR	constante 3 bits→ ARP. (ARP) → ARB.
LDP	LDP dma LDP ind[,next ARP]	Cargar registro puntero de página de memoria de datos	(dma) → DP
LDPK	LDPK constante	Cargar registro puntero de página de memoria de datos en modo inmediato	constante 9 bits→ DP
LRLK	LRLK AR, constante	Cargar AR con constante en formato inmediato largo	constante 16 bits→ AR
MAR	MAR dma MAR ind[,next ARP]	Modificar contenido AR (actua como NOP en direccionamiento directo)	Modificar (AR(ARP)) según especificación *ind*
SAR	SAR AR, dma SAR AR, ind[,next ARP]	Almacenar contenido AR en memoria	(AR) → dma
SBRK	SBRK constante	Restar a AR activo constante en formato inmediato corto	(AR) - constante 8 bits→ AR

SALTOS Y LLAMADAS A SUBRUTINAS			
Instr.	**Sintaxis**	**Descripción**	**Operación**
B	B pma [,ind[,next ARP]]	Salto incondicional	pma → PC
BACC	BACC	Salto a dirección dada por el ACC	(ACC(15-0)) → PC
BANZ	BANZ pma [,ind[,next ARP]]	Salto si AR no cero	Si (AR(ARP)) ≠ 0, pma → PC. si no, (PC) + 2 → PC
BBNZ	BBNZ pma [,ind[,next ARP]]	Salto si bit TC no cero	Si (TC) = 1, pma → PC. si no, (PC) + 2 → PC
BBZ	BBZ pma [,ind[,next ARP]]	Salto si bit TC cero	Si (TC) = 0, pma → PC. si no, (PC) + 2 → PC
BC	BC pma [,ind[,next ARP]]	Salto si acarreo	Si (C) = 1, pma → PC. si no, (PC) + 2 → PC
BGEZ	BGEZ pma [,ind[,next ARP]]	Salto si ACC ≥ 0	Si (ACC) ≥ 0, pma → PC. si no, (PC) + 2 → PC
BGZ	BGZ pma [,ind[,next ARP]]	Salto si ACC > 0	Si (ACC) > 0, pma → PC. si no, (PC) + 2 → PC
BIOZ	BIOZ pma [,ind[,next ARP]]	Salto si estado I/O = 0	Si NOT(BIO) = 0, pma → PC. si no, (PC) + 2 → PC
BLEZ	BLEZ pma [,ind[,next ARP]]	Salto si ACC ≤ 0	Si (ACC) ≤ 0, pma → PC. si no, (PC) + 2 → PC

BLZ	BLZ pma, ind[,next ARP]]	Salto si ACC < 0	Si (ACC) < 0, pma → PC. si no, (PC) + 2 → PC
BNC	BNC pma, ind[,next ARP]]	Salto si no acarreo	Si (C) = 0, pma → PC. si no, (PC) + 2 → PC
BNV	BNV pma, ind[,next ARP]]	Salto si no desbordamiento	Si (OV) ≠ 0, pma → PC. si no, (PC) + 2 → PC
BNZ	BNZ pma, ind[,next ARP]]	Salto si ACC ≠ 0	Si (ACC) ≠ 0, pma → PC. si no, (PC) + 2 → PC
BV	BV pma, ind[,next ARP]]	Salto si desbordamiento	Si (OV) = 0, pma → PC. si no, (PC) + 2 → PC
BZ	BZ pma, ind[,next ARP]]	Salto si ACC = 0	Si (ACC) = 0, pma → PC. si no, (PC) + 2 → PC
CALA	CALA	Llamada indirecta a subrutina	(PC) + 1 → pila (ACC(15-0)) → PC
CALL	CALL pma, ind[,next ARP]]	Llamada a subrutina	(PC) + 2 → pila pma → PC
RET	RET	Retorno de subrutina	(pila) → PC

MEMORIA Y E/S			
Instr.	**Sintaxis**	**Descripción**	**Operación**
BLKD	BLKD dma1, dma2 BLKD dma1, ind[,next ARP]]	Mover bloque en memoria de datos	(dma1, direccionada por PC) → dma2
BLKP	BLKP pma, dma BLKP pma, ind[,next ARP]]	Mover bloque de memoria programa a memoria datos	(pma, direccionada por PC) → dma
DMOV	DMOV dma, ind[,next ARP]]	Mover dato en memoria de datos	(dma) → dma + 1
FORT	FORT constante	Determina formato puerto serie	constante 1 bit → FO
IN	IN dma, puerto IN ind, puerto[,next ARP]]	Leer dato desde puerto	(puerto) → dma
OUT	OUT dma, puerto OUT ind, puerto[,next ARP]]	Sacar dato por puerto	(dma) → puerto
RFSM	RFSM	Reset modo sincronización puerto serie	0 → FSM
RTXM	RTXM	Reset modo transmisión puerto serie	0 → TXM
RXF	RXF	Reset flag externo	0 → XF
SFSM	SFSM	Set modo sincronización puerto serie	1 → FSM
STXM	STXM	Set modo transmisión puerto serie	1 → TXM
SXF	SXF	Set flag externo	1 → XF
TBLR	TBLR dma TBLR ind[,next ARP]]	Lectura tabla	(pma, obtenida de ACCL) → dma
TBLW	TBLW dma TBLW ind[,next ARP]]	Escritura tabla	(dma) → pma, obtenida de ACCL

CONTROL			
Instr.	**Sintaxis**	**Descripción**	**Operación**
BIT	BIT dma, código bit BIT ind, código bit[,next ARP]]	Test bit	(bit dma en posición (código bit)) → TC
BITT	BITT dma, ind[,next ARP]]	Test bit especificado por registro T	(bit dma en posición (T)) → TC
CNFD	CNFD	Configurar B0 como memoria datos *	0 → CNF
CNFP	CNFP	Configurar B0 como memoria programa *	1 → CNF
CONF	CONF constante	Configurar bloques memoria **	constante 2 bits → CNF
DINT	DINT	Desactivar interrupción	1 → INTM
EINT	EINT	Activar interrupción	0 → INTM
IDLE	IDLE	Activar modo bajo consumo hasta interrupción	(PC) + 1 → PC Modo bajo consumo
LST	LST dma LST ind[,next ARP]]	Cargar registro estado ST0	(dma) → ST0
LST1	LST1 dma LST1 ind[,next ARP]]	Cargar registro estado ST1	(dma) → ST1
NOP	NOP	No operación	(PC) + 1 → PC
POP	POP	Carga ACCL con contenido cima pila	(pila) → ACC(15-0)
POPD	POPD dma POPD ind[,next ARP]]	Descarga contenido cima pila en memoria	(pila) → dma
PSHD	PSHD dma PSHD ind[,next ARP]]	Carga dato memoria en cima pila	(dma) → pila
PUSH	PUSH	Carga cima pila con contenido ACCL	(ACC(15-0)) → pila
RC	RC	Reset bit acarreo	0 → C
RHM	RHM	Reset modo *hold*	0 → HM
ROVM	ROVM	Reset modo desbordamiento	0 → OVM
RPT	RPT dma RPT ind[,next ARP]]	Cargar contador bucle con valor en memoria	(dma) → RPTC
RPTK	RPTK constante	Cargar contador bucle con constante en formato inmediato corto	Constante 8 bits → RPTC
RSXM	RSXM	Reset modo extensión signo	0 → SXM
RTC	RTC	Reset bit control/test	0 → TC
SC	SC	Set bit acarreo	1 → C
SHM	SHM	Set modo *hold*	1 → HM
SOVM	SOVM	Set modo desbordamiento	1 → OVM
SST	SST dma SST ind[,next ARP]]	Guardar registro estado ST0 en memoria	ST0 → dma
SST1	SST1 dma SST1 ind[,next ARP]]	Guardar registro estado ST1 en memoria	ST1 → dma
SSXM	SSXM	Set modo extensión signo	1 → SXM

STC	STC	Set bit control/test	1 → TC
TRAP	TRAP	Interrupción *software*	(PC) + 1 → pila 30 → PC

*: en TMS320C25
**: en TMS320C26

Tabla 4.1. Conjunto de instrucciones del TMS320C2x.

Paralelismo del conjunto de instrucciones

La principal diferencia entre el conjunto de instrucciones de un DSP y de un microprocesador convencional es la posibilidad de que algunas instrucciones permitan controlar el funcionamiento paralelo de las distintas unidades funcionales internas. Para mostrar este hecho y la dualidad entre *hardware* y software, vamos a mostrar varios niveles de complejidad:

1. Hemos visto cómo instrucciones como ADD pueden controlar el Acumulador, la ALU y el desplazador simultáneamente, y los mecanismos *hardware* implicados (lazos de realimentación, etc.)
2. Otra unidad funcional es el multiplicador. En su nivel más simple, habría también instrucciones de carga del registro T (LT) o P (LPH, SPH, SPL) y multiplicaciones (MPY).
3. Podemos combinar el multiplicador con la ALU sumando (APAC), restando (SPAC) o cargando (PAC) P en el ACC. Esto implica también al desplazador de salida del multiplicador.
4. En un nivel superior, mientras se carga el registro T o se realiza una multiplicación, se pueden realizar las operaciones descritas en 3, puesto que corresponden a caminos *hardware* diferentes y no hay conflictos. Fusionar las instrucciones del grupo 2 y 3 permite ahorrar tiempo (LTA, LTS, LTP, MPYA, MPYS).
5. En las operaciones comentadas en 4, se accede a un operando en cada instrucción, por lo que realizar una multiplicación y suma implica 2 instrucciones (LTA+MPY, etc.) Es posible realizar esto accediendo a 2 operandos en un ciclo de reloj siempre que cada uno de ellos esté en una memoria distinta (programa-datos) (MAC).
6. En el nivel más complejo, se puede controlar el multiplicador, la ALU, los desplazadores y memoria (LTD y MACD). Esto es particularmente útil en el caso de implementación de filtros digitales, en los que además del sumatorio de productos se debe actualizar el buffer de entrada (gestión de memoria a través del generador de direcciones).

Existen diferentes combinaciones de instrucciones que producen el mismo resultado (por ejemplo, LTA-MPY = MAC; LTD-MPY = MACD). Esto permite una

mayor flexibilidad de programación y permite mayor eficacia de ejecución según el caso, como veremos más adelante.

MACD: Multiplicar y acumular con movimiento de datos.

La instrucción MACD sirve como ejemplo del paralelismo diseñado en el conjunto de instrucciones y la arquitectura del TMS320C2x. La ejecución de MACD implica los siguientes pasos:

1. El contenido del registro P (32 bits) es desplazado (escalado) por un desplazador de salida. (DESPLAZADOR)

2. La ALU (32 bits) acumula el resultado del desplazamiento del registro P con el contenido actual del ACC de 32 bits. (ALU)

3. Se activan apropiadamente los bits de acarreo y desbordamiento. (ALU)

4. El contenido (16 bits) de una posición de memoria de datos (generalmente direccionada indirectamente a través de un registro auxiliar) se carga en el registro T. (ARAU)

5. El contenido (16 bits) de una posición de memoria de programa (direccionada a través del PFC -*prefetch counter*-), se carga en el multiplicador. (CONTROL)

6. Se realiza una multiplicación de 16x16 bits, proporcionando un resultado de 32 bits que se carga en el registro P para ser acumulado en el siguiente ciclo. (MULTIPLICADOR)

7. El contenido (16 bits) de una posición de memoria de datos se copia en la siguiente posición. (ARAU)

8. El contenido (16 bits) del AR actual se modifica adecuadamente para direccionar la memoria de datos en el siguiente ciclo. (ARAU)

9. El contenido (16 bits) del PFC se incrementa para direccionar la siguiente posición de la memoria de programa. (CONTROL)

10. El contador de repetición se decrementa. (CONTROL)

En este caso, uno de los datos se obtiene de la memoria de datos mientras el otro se toma de la memoria de programa. Si se utiliza memoria de datos interna, se puede ejecutar en un solo ciclo de instrucción. Además, el desplazamiento de los datos en el buffer sólo se puede dar en este caso.

El siguiente ejemplo muestra el funcionamiento de la MACD:

```
MACD        pma, ind [,next ARP]
```

1. Definir dirección de memoria de datos a través de un AR:

```
LARP   1          ;AR actual =1
LRLK   1,300h     ;inicializa AR: apunta a RAMB1
```

2. Definir el funcionamiento del desplazador de salida del multiplicador, implicado en la acumulación de P y ACC.

```
SPM    0          ;sin desplazamiento del registro P
```

3. Definir pma y el modo de actualización de AR. Como la actualización de la memoria se hace copiando el contenido de la posición actual en la siguiente, AR debe siempre inicializarse al final del buffer y recorrerlo hacia atrás.

```
MACD        020h,*-     ;PFC= dir. Memoria de programa.
```

Antes de ejecutar MACD:

PFC=20h   AR1=300h

Después de ejecutar MACD:

ACC=(ACC)+P   P=(20h) x (300h)

PFC=21h   AR1=299h   (301h)=(300h)

Esta instrucción suele utilizarse en la implementación de filtros FIR. En este caso:

$$y(n) = \sum_{k=0}^{N} h(k)x(n-k)$$

Los coeficientes del filtro (h(k)) se sitúan en memoria de programa y se direccionan con PFC. Los datos están en un buffer de entrada direccionado por AR. Para realizar el sumatorio de N+1 términos debe inicializarse el contador de bucle:

```
RPTK  N          ;repetir N+1 veces (contando el cero)
```

La acumulación se produce antes de efectuar el producto, por lo que al terminar el bucle el último producto permanece en P.

Movimiento de bloques

El TMS320C2x soporta seis instrucciones de movimiento de bloques de datos y programa, y transferencias de datos a través de los puertos de E/S.

La instrucción BLKD mueve un bloque dentro de la memoria de datos, y la BLKP lo hace desde la memoria de programa a la de datos. La transferencia de bloques

entre espacios de memoria de datos y programa pueden implementarse mediante TBLR y TBLW (*table read* y *table write*). Las instrucciones IN y OUT permiten transferencias entre memoria de datos y espacio de E/S.

4.1.3. Ejemplos de programación

Ejemplo 1

Implementar una rutina que realice la operación:

$$Z = \sum_{i=1}^{10} X(i)Y(i)$$

¡Error!No se le ha dado un nombre al marcador.

Para poder realizar el producto de cada término y la suma con los anteriores en un sólo ciclo de reloj, deberemos localizar los datos x(i) e y(i) en memorias separadas, lo que permitirá acceder por buses distintos simultáneamente. Para ello, situaremos los x(i) en el B0 y los y(i) en el B1, y posteriormente configuraremos B0 como memoria de programa. A partir de este momento, B0 se direccionará con el PFC, mientras que B1 lo será con alguno de los AR. Las direcciones respectivas de comienzo, como se especifican en los mapas de memoria para el caso MC, serán 0FF00h y 300h. Situaremos la variable de salida Z a continuación del último Y(i).

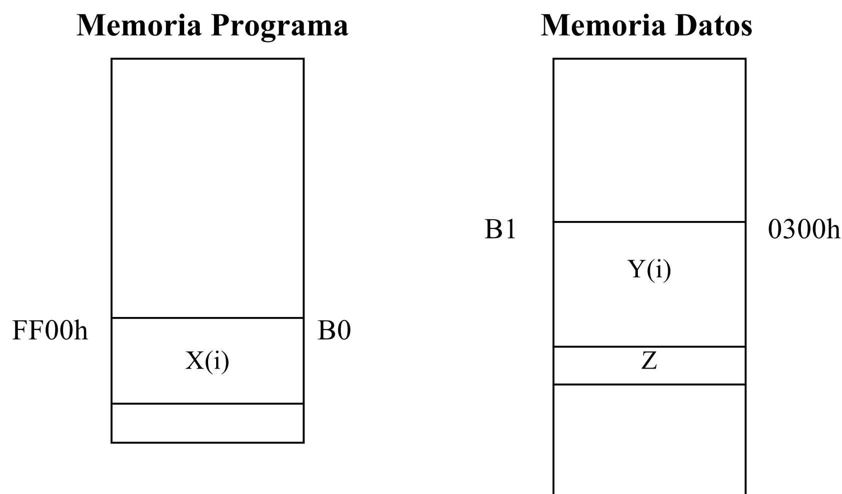


Figura 4.4. Configuración de memoria para el Ejemplo 1.

El algoritmo será el siguiente:

JUAN F. GUERRERO MARTÍNEZ
JOSÉ V. FRANCÉS VILLORA

```
SERIE
        Inicializar B0 como PROGMEM.
        Inicializar AR=B1.
        Inicializar ACC y P.

        Para i=1 hasta N
                ACC=ACC+x(i)*y(i)
                AR=AR+1
        Z=ACC.
```

En ensamblador del TMS320C25:

```
SERIE   CNFP                ;B0:PROGMEM
        LARP  AR1           ;inicializa ARP para direccionamiento
                            ; indirecto
        LRLK  AR1,300h      ;inicializa AR1 al comienzo de B1
        ZAC                 ;ACC=0
        MPYK 0h             ;P=0
        RPTK  N-1           ;bucle desde N-1 hasta 0
        MAC   0FF00h,*+     ;acumula producto. previo y multiplica.
                            ; Inicializa pma y define modificación AR1
        APAC                ;acumula último producto
        SACH *              ;AR1 direcciona Z. Z=ACC
```

En este caso, hemos utilizado MAC conjuntamente con el contador de bucle para implementar la rutina. La estructura de *pipeline* para esta combinación permite realizar la operación de multiplicación y suma en un solo ciclo. No obstante, esta *pipeline* implica un cierto tiempo para cargar las condiciones iniciales, por lo que cuando la velocidad es el parámetro importante en la aplicación en lugar de la cantidad de memoria, puede resultar conveniente sustituir instrucciones MAC por pares LTA-MPY. Vamos a analizar ambas formas de implementar el algoritmo en función de la velocidad de ejecución y de la cantidad de memoria necesaria.

- **Utilizando MAC**

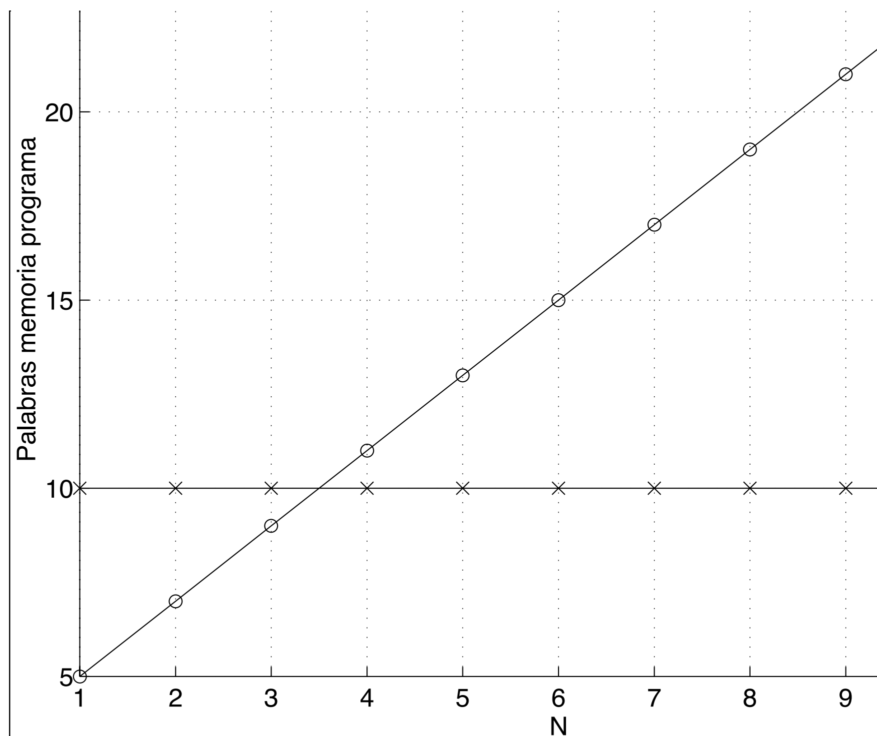
	Ciclos reloj	Posiciones memoria
CNFP	1	1
LARP AR1	1	1
LRLK AR1,300h	2	2
ZAC	1	1
MPYK 0h	1	1
RPTK N-1	1	1
MAC 0FF00h,*+	3+N	2
APAC	1	1

	Ciclos reloj	Posiciones memoria
TOTAL:	11+N	10

- **Utilizando LTA-MPY**

		Ciclos reloj		Posiciones memoria
LDPK 6		1		1
ZAC		1		1
LT X1		1		1
MPY Y1		1		1
.				
.	2N		2N	
.				
LT XN		1		1
MPY YN		1		1
APAC		1		1
TOTAL:		3+2N		3+2N

En las siguientes gráficas se puede ver el tiempo de ejecución y la memoria de programa en función del número de sumas y productos. Como puede verse, el método LTA-MPY es más rápido para valores inferiores a 8 términos, y ocupa menos memoria para menos de 4 términos.



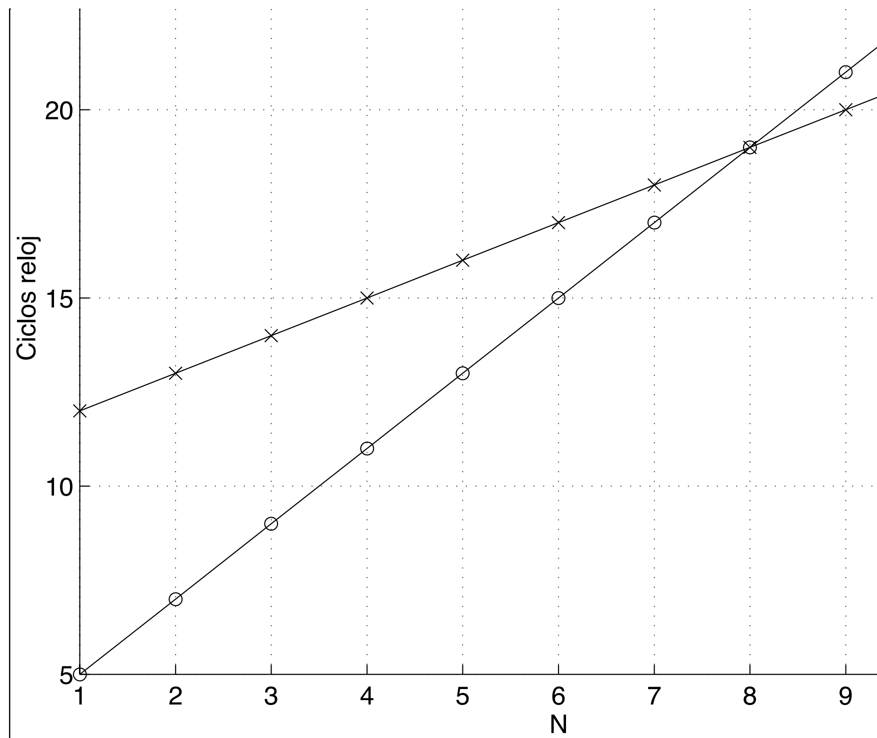


Figura 4.5. Tiempo de ejecución y memoria de programa en función del número de sumas y productos. (o: LTA-MPY; x: MAC).

Ejemplo 2

Implementación de un filtro digital FIR de orden 40.

La expresión general para un filtro digital viene dada por la ecuación:

$$y(n) = \sum_{k=0}^{N} a_k y(n-k) + \sum_{k=0}^{N} b_k x(n-k)$$

A partir de esta ecuación, podemos diseñar los dos tipos de filtros digitales: FIR (o Respuesta Finita al Impulso) e IIR (Respuesta Infinita al Impulso). Para el caso que nos ocupa, filtro FIR, los coeficientes a_k valen 0 y la ecuación se reduce a:

$$y(n) = \sum_{k=0}^{N} b_k x(n-k) b_k = \sum_{k=0}^{N} h(k)x(n-k)$$

donde N es el orden, (N+1) es la longitud y h(k) es la correspondiente respuesta al impulso del filtro digital. Esta ecuación puede representarse por la red o estructura que se muestra en la siguiente figura. Dicha estructura se denomina directa puesto que puede identificarse directamente con la ecuación, sustituyendo los retardos implicados en las muestras por los términos z^{-1}.

La ecuación que debemos implementar para nuestro caso será:

$$y(n) = x(n-40)h(40) + x(n-39)h(39) + \ldots + x(n-2)h(2) + x(n-1)h(1) + x(n-0)h(0)$$

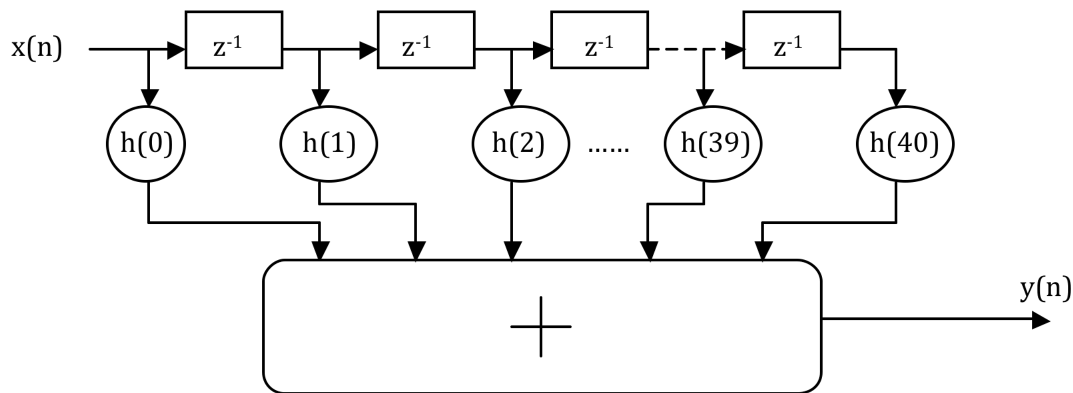


Figura 4.6. Estructura del filtro FIR de orden 40.

De las dos formas de implementar las sumas y productos con el desplazamiento de las muestras de entrada comentadas (utilizar el par de instrucciones LTD/MPY o la instrucción MACD), la implementación de filtros FIR suele utilizar la segunda opción. Esto se debe principalmente a que el orden de un filtro FIR es generalmente elevado. La utilización de MACD implica disponer de uno de los términos producto (usualmente los coeficientes del filtro) en memoria de programa y el otro en memoria de datos para poder realizar un acceso simultáneo. Situaremos las muestras de entrada (x(n)) en el bloque B1 y los coeficientes (h(k)) en la EPROM interna situada en memoria de programa. La configuración de memoria para el presente programa será:

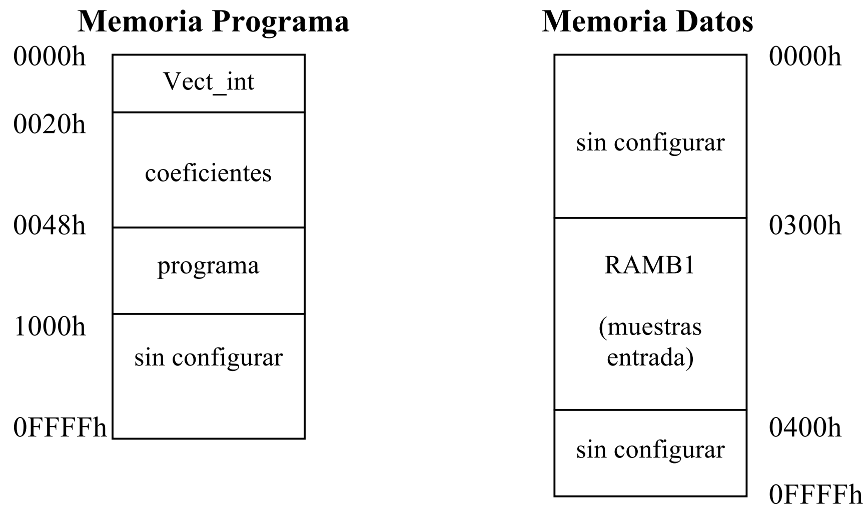


Figura 4.7. Configuración de memoria para el ejemplo 2.

La rutina que implementa este filtro será:

```
         LARP   AR1          ;AR1 registro auxiliar actual
IN_DATO  IN     XN,PA0       ;lee x(n) del puerto 0
         LRLK   AR1,300h+N   ;AR1->B1+N
         MPYK   0            ;inicializa P
         ZAC                 ;inicializa ACC
         RPTK   N            ;repite N+1 veces
         MACD   20h,*-       ;dir coef: 20h, se decrementa AR1
         APAC                ;suma último producto
         SACH   YN,1         ;almacena salida con desplazamiento
                             ;izqda. para eliminar signo
                             ;redundante
         OUT    YN,PA1       ;salida filtro por puerto 1
         B      IN_DATO      ;lee siguiente muestra
```

Ejemplo 3

Implementación de un filtro digital IIR de orden 2.

La expresión general para un filtro digital IIR viene dada por la ecuación:

$$y(n) = \sum_{k=0}^{N} b_k x(n-k) b_k = \sum_{k=0}^{N} h(k)x(n-k)$$

de lo que se deduce que la salida del filtro es una suma, con distintos pesos, de los valores de entrada y salida actuales y pasados. Existen diversas estructuras para implementar esta ecuación. Vamos a utilizar la forma directa canónica (o forma directa II, ver figura), que tiene un número mínimo de retrasos, por lo que requiere el valor mínimo de registros de almacenamiento. Esta estructura es ventajosa respecto a la minimización de la memoria de datos utilizada.

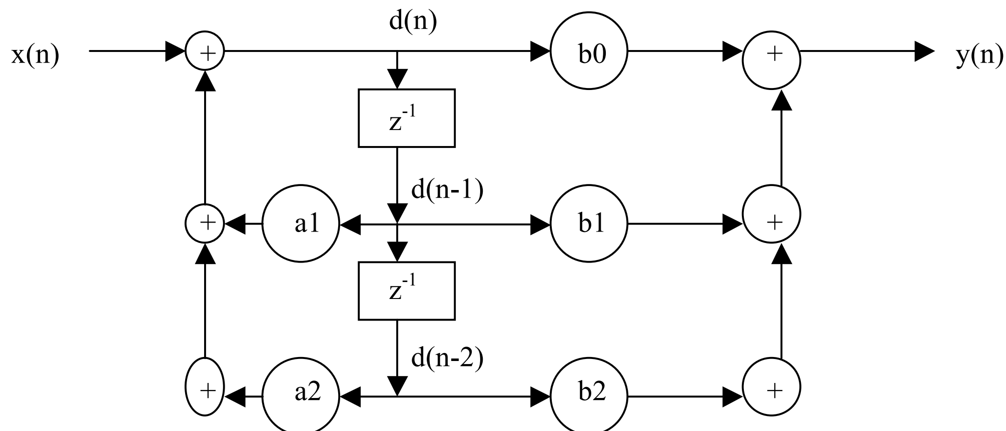


Figura 4.8. Estructura directa II para el filtro IIR de orden 2.

Como se comentó anteriormente, debido a las operaciones de inicialización de la instrucción MACD, ésta no es adecuada cuando se repite pocas veces. En el caso de implementación de filtros IIR, el diseño se suele hacer con cascada de células de segundo orden, por lo que el método más apropiado es el de pares de instrucciones LTD-MPY, que son más eficientes para órdenes pequeños.

Las ecuaciones en diferencias vienen dadas por:

$$d(n) = x(n) + a_1 \cdot d(n-1) + a_2 \cdot d(n-2)$$
$$y(n) = b_0 \cdot d(n) + b_1 \cdot d(n-1) + b_2 \cdot d(n-2)$$

En este caso, los d(n) corresponde a los valores en los distintos retrasos de la red. Estos retrasos, así como los coeficientes, la variable de entrada y la de salida se almacenan en memoria según:

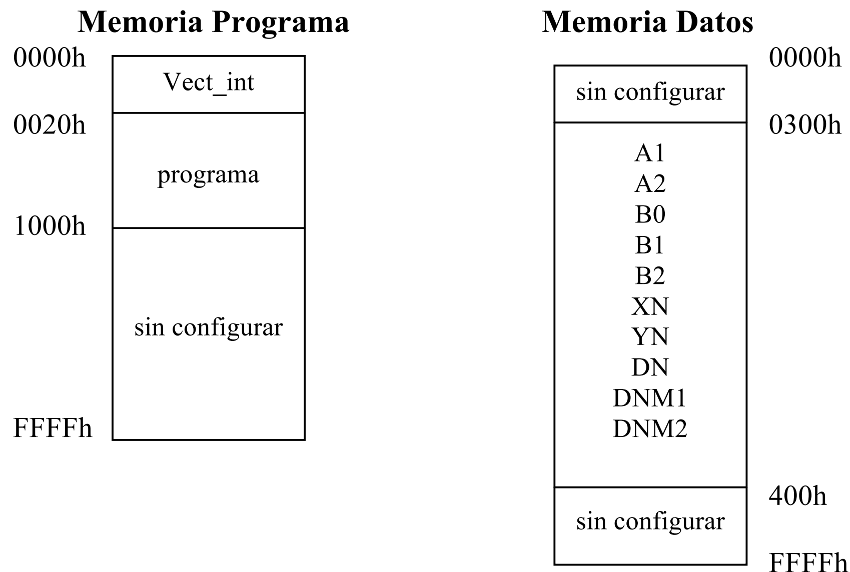
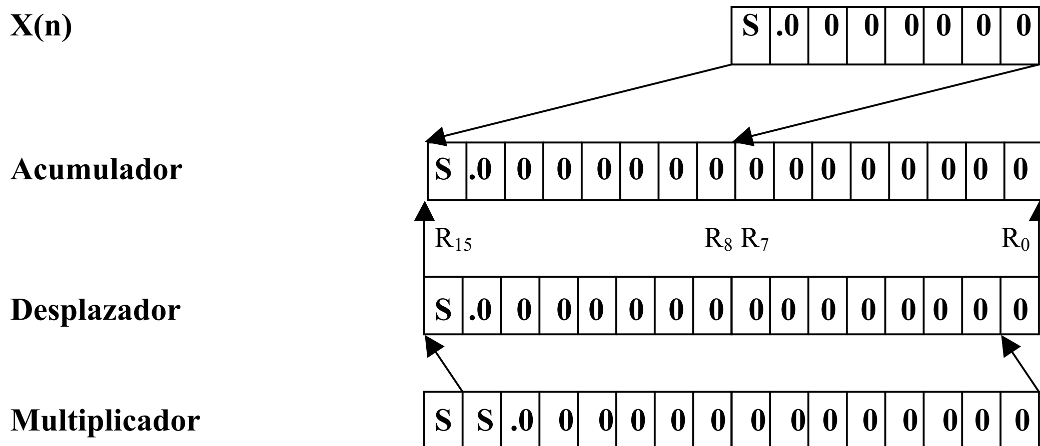


Figura 4.9. Configuración de memoria para el Ejemplo 3.

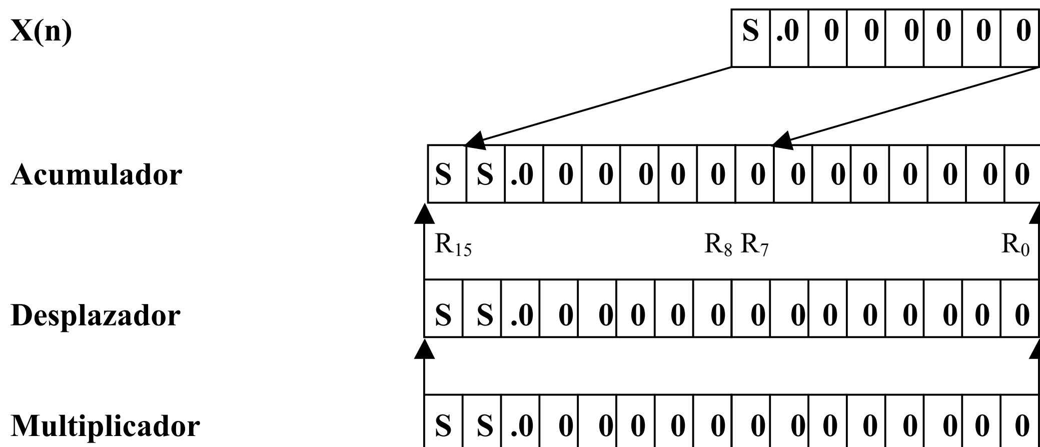
En cada paso del algoritmo, se realiza un producto y el resultado del producto previo se suma al acumulador. También se desplazan los valores de los retrasos a la siguiente posición de memoria, preparándolos para el siguiente paso.

Un aspecto a tener en cuenta es el que se da en la primera ecuación de la estructura. Aquí se realiza una suma entre x(n), que es un número binario en formato Q15 almacenado en memoria, y los productos entre los coeficientes a_i y los retrasos de la estructura, que proceden del registro P y por tanto tienen doble precisión (Q30). Por tanto, antes de realizar la suma, hay que alinear correctamente ambos números, de manera que la coma y los bits de signo coincidan. Existen dos opciones:

1. Cargar x(n) en la parte alta del acumulador (ACCH) y programar el desplazador de salida del multiplicador para eliminar el signo redundante antes de realizar la suma. De este modo, tendríamos:



2. Desplazar x(n) 15 bits a la izquierda al cargarlo en el acumulador (esto crea un segundo bit de signo) y no desplazar la salida del multiplicador. De este modo, ambos operandos presentan signo redundante y las posiciones de la coma coinciden (el signo redundante se debe eliminar en el resultado final). Con esta opción, tendremos:



La rutina que implementa este filtro será:

```
      LDPK  6          ; inicializa página de datos 6 (bloque B1)
IIR   IN    XN,PA0     ;lee dato entrada de Puerto 0
      LAC   XN,15      ;ACCH=x(n)
      LT    DNM1       ;T=d(n-1)
      MPY   A1         ;P=a1·d(n-1)
      LTA   DNM2       ;ACC=x(n)+a1·d(n-1) | T=d(n-2)
```

```
MPY   A2          ; P=a2·d(n-2)
APAC              ; ACC=x(n)+a1·d(n-1)+a2·d(n-2)
SACH  DN,1        ; d(n)=x(n)+a1·d(n-1)+a2·d(n-2)
ZAC               ; inicializa ACC
MPY   B2          ; T=d(n-2) (se mantiene) | P=b2·d(n-2)
LTD   DNM1        ; T=d(n-1) | ACC=b2·d(n-2) | DNM1 -> DNM2
MPY   B1          ; P=b1·d(n-1)
LTD   DN          ; T=d(n) | ACC=b2·d(n-2)+b1·d(n-1)
                  ; DN → DNM2
MPY   B0          ; P=b0·d(n)
APAC              ; ACC=b2·d(n-2)+b1·d(n-1)+b0·d(n)
SACH  YN,1        ; y(n)=b2·d(n-2)+b1·d(n-1)+b0·d(n)
                  ; desplaza 1 bit a la izquierda el resultado
                  ; para eliminar el signo redundante
OUT   YN,PA1      ; salida filtro por Puerto 1
B     IIR         ; toma nueva muestra
```

4.2. Aplicaciones de los DSP

Los campos de aplicación de los procesadores digitales de señal se han ido ampliando según los sistemas electrónicos analógicos han ido sustituyéndose por digitales. Actualmente podemos encontrar DSP en sistemas de comunicaciones, control, radar o electrónica de consumo, entre otras, y es tarea del diseñador seleccionar adecuadamente el procesador adecuado para cada caso según criterios de coste, prestaciones, consumo, integración, etc.

Aunque no de forma exhaustiva, podemos clasificar las aplicaciones en tres grandes grupos:

- Sistemas de bajo coste. Se caracterizan por el gran volumen de unidades desarrolladas, lo cual implica que el coste es el factor determinante. Como ejemplos tenemos *modems*, teléfonos móviles o controladores de disco. En sistemas alimentados por baterías, el consumo (y también el peso) son factores a considerar. Suelen hacerse desarrollos propietarios (específicos para cada sistema) tanto de *hardware* como de *software*.

- Aplicaciones de altas prestaciones. En este caso, se trata de sistemas con necesidades especiales que procesan grandes cantidades de datos y con algoritmos complejos, como ocurre en el caso de sonar y exploración sísmica. Suelen implicar un bajo número de unidades y necesitar arquitecturas multiprocesador, por lo que los desarrollos se realizan generalmente utilizando sistemas ya desarrollados que se interconectan adecuadamente, así como el uso de librerías software de soporte a la programación.

- Aplicaciones multimedia para ordenadores personales. Actualmente, este tipo de ordenadores va incorporando cada vez en mayor medida aplicaciones multimedia como síntesis de voz o música, *modems*, compresión de imágenes, etc. La utilización de un DSP en sistemas de este tipo permiten bajo coste, alta integración a la vez que altas prestaciones. Aunque los procesadores principales cada vez incluyen más funciones para el procesado digital, las prestaciones que proporciona un DSP hacen que sea la mejor opción, al menos actualmente y, previsiblemente, durante los próximos años.

Dentro de estos tres grupos, podemos destacar los siguientes campos de aplicación:

- Procesado digital de señal: filtrado digital y adaptativo, convolución, correlación, FFT, generación de ondas.

- Procesado de voz: vocoders, reconocimiento del habla, verificación, síntesis, conversión texto-voz.

- Procesado de imágenes: visión de robots, compresión y transmisión de imágenes, reconocimiento de patrones, procesado homomórfico, animación.

- Instrumentación: análisis espectral, generación de funciones, reconocimiento de patrones, procesado sísmico, análisis de transitorios.

- Control: control de discos, robots, impresoras láser, control de motores.

- Telecomunicaciones: cancelación de ecos, ADPCM, repetidores de línea, multiplexado de canales, *modems*, ecualizadores adaptativos, codificación/ decodificación DTMF, encriptación de datos, FAX, teléfonos móviles, videoconferencia.

- Automoción: análisis de vibraciones, control de conducción adaptativa, posicionamiento global, comando por voz, radio digital.

- Consumo: detectores de radar, audio/TV digital, sintetizadores de música, juguetes.

- Industrial: robótica, control numérico, seguridad de acceso, monitores de líneas de tensión.

- Aplicaciones médicas: prótesis auditivas, monitorización de pacientes, equipos de ultrasonidos, herramientas de diagnóstico, prótesis, monitores fetales.

- Aplicaciones militares: comunicaciones seguras, radar, sonar, procesado de imágenes, navegación, guiado de misiles, *modems* de RF.

4.3. Ejemplos de desarrollo

En esta sección, se presentará el desarrollo de algunos algoritmos en ensamblador del TMS320C2x que ilustran técnicas de programación de los mismos. Se tratarán dos tipos de implementación, utilizando el simulador y la placa DSK descritos en el capítulo anterior.

4.3.1. Filtrado Digital

Un problema asociado a la implementación de un filtro digital, y en general a la de algoritmos con DSP o con cualquier procesador, es la posibilidad de que se produzca desbordamiento en el resultado final o en los intermedios debido al tamaño finito de los registros. Dicha posibilidad es mayor cuando se utiliza aritmética en coma fija. El producto de dos números no suelen producir desbordamiento (la salida del multiplicador es de doble precisión en muchos DSP), pero las sumas pueden hacerlo a pesar de la existencia de bits de guarda, como ya se comentó. La solución es escalar los operandos previamente.

Supongamos un filtro digital IIR de segundo orden. Su función de transferencia viene dada por:

$$H(z)=\frac{a(0)+a(1)Z^{-1}+a(2)Z^{-1}}{1+b(1)Z^{-1}+b(2)Z^{-1}}$$

Si escalamos por un factor S, tendremos:

$$[y(n)/S]=[a(0)/S_1][x(n)/S_2]+$$

$$[a(1)/S_1][x(n-1)/S_2]+[a(2)/S_1][x(n-2)/S_2]$$

$$-b(1)y(n-1)-b(2)y(n-2)$$

El factor de escala $S = S_1 \cdot S_2$ se ha descompuesto en dos términos para minimizar los errores de cuantización que se introducen al escalar. Si todo el escalado se agrupa sobre los coeficientes, se pierde resolución en éstos y las características del filtro sufren una dispersión, pudiendo volverse inestable. Si se agrupan en la entrada y salida, se pierde rango dinámico. La distribución debe seguir la norma más restrictiva, y los coeficientes deben escalarse lo menos posible. Por ejemplo, si S=64, los coeficientes se escalarían por 4 y los datos por 16 (16·4=64).

Para estimar el factor de escala, existen diversas definiciones. Vamos a tener en cuenta la respuesta del filtro al impulso unidad. En este caso, la escala vendría dada por:

$$S=\sum_{n=0}^{N-1}|h(n)|$$

Para el caso estacionario, sería:

$$S=\mathrm{Max}|H(e^{2\pi jft})|$$

El primer caso es el más restrictivo. El segundo sólo asegura que no existirá desbordamiento en respuesta estacionaria, pero puede producirse en transitoria.

Filtro IIR

Vamos a implementar un filtro IIR de 2º orden con estructura transpuesta (Figura 4.1) especificada por las ecuaciones:

$$y(n) = a_0 \cdot x(n) + d_1(n)$$

$$d_1(n) = d_2(n) + a_1 \cdot x(n) - b_1 \cdot y(n)$$

$$d_2(n) = a_2 \cdot x(n) - b_2 \cdot y(n)$$

Los pasos a seguir serán:

1. Simular la respuesta transitoria y estacionaria del filtro en los nodos suma.

2. Calcular el valor del factor de escalado necesario para evitar desbordamiento.

3. Obtener el valor de los coeficientes escalados en formato Q15 con su equivalente en decimal.

4. Programar el filtro en ensamblador.

Suponiendo una frecuencia de muestreo de 10 kHz, y un filtro pasa-banda con frecuencia de pico en 1600 Hz, los valores de los coeficientes iniciales son:

a_0= 2.0585 a_1= 0.4262 a_2= -1.6324
b_0= -0.8524 b_1= 0.7047

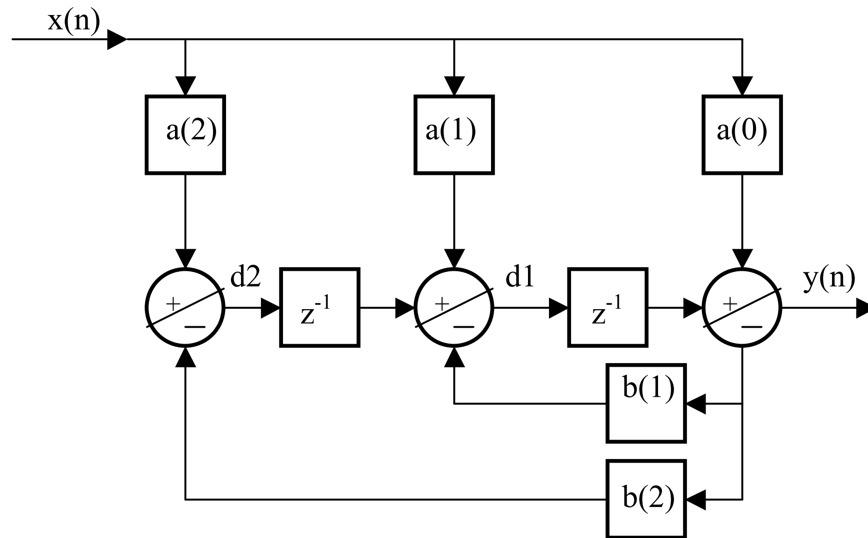


Figura 4.10. Estructura transpuesta del filtro IIR de 2º orden.

Para simular la respuesta transitoria y estacionaria del filtro en los nodos suma, deberemos calcular las siguientes funciones de transferencia:

$$H(z)=\frac{Y(z)}{X(z)}$$

$$H1(z)=\frac{D1(z)}{X(z)}$$

$$H2(z)=\frac{D2(z)}{X(z)}$$

para los dos métodos de escalado. Para el primero, se utilizará como entrada un impulso unidad. Puesto que se trata de un filtro IIR, tiene respuesta infinita a un impulso unidad, por lo que para calcular el factor de escalado, se deberán tomar muestras hasta que la salida del filtro no varíe apreciablemente.

Para el segundo caso, hay que utilizar una entrada coseno y el mismo número de muestras usadas para el caso anterior, realizando un barrido de frecuencias hasta la frecuencia de Nyquist.

De los valores obtenidos, seleccionaremos el caso más desfavorable, y tomaremos la potencia de dos más cercana por exceso.

Después de obtener los factores de escala para los casos transitorio (16.7) y estacionario (12.5), seleccionamos el correspondiente al primero por ser el más restrictivo. Este valor está comprendido entre $2^4=16$ y $2^5=32$. Aunque estrictamente

debería tomarse la potencia de dos por exceso, en este caso el valor es muy próximo a 16, por lo que la probabilidad de que se produzca desbordamiento tomando dicha potencia es muy pequeña, y se escalan menos los resultados por lo que se pierde menos resolución. Por ello, tomaremos como factor de escalado 16, y lo dividiremos en un factor para coeficientes (S_1) de 2 y un factor para datos (S_2) de 8.

Los coeficientes escalados y en formato Q15 serán:

$a_0/2 = 16864$ $\quad a_1 = 6983$ $\quad a_2 = -26744$
$-b_0 = 27930$ $\quad -b_1 = -23093$

El valor de a_0 se guarda dividido por dos ya que es mayor que la unidad y no puede representarse directamente en formato Q15. Los valores de b_0 y b_1 se almacenan negados para simplificar la programación, utilizando así sólo sumas en las ecuaciones del filtro.

Otro aspecto a considerar es la programación de interrupciones. Los DSP utilizan valores de entrada que son las muestras de una señal analógica previamente convertida por un conversor A/D. La frecuencia de muestreo suele obtenerse mediante la programación del temporizador interno, que genera una interrupción cada periodo de muestreo. Por tanto, la rutina de servicio de interrupción del temporizador se corresponde generalmente con el procesado a realizar entre muestra y muestra.

La frecuencia de reloj del temporizador es MASTERCLK/4, siendo MASTERCLK la frecuencia de reloj del procesador, como se vio en el capítulo 4 al hablar de la familia TMS320C2x. Por tanto, el valor a programar en el temporizador vendrá dado por:

$$Kt = MASTERCLK / (4 \cdot Fm).$$

donde:

Kt: constante del temporizador.

Fm: frecuencia de muestreo.

MASTERCLK: 50 MHz para el TMS320C25.

En este caso, la frecuencia de muestreo es de 10 kHz, por lo que:

$$Kt = 50\ MHz / (4 \cdot 10\ kHz) = 1250$$

Una vez determinado el valor de Kt, deben inicializarse los vectores de interrupción. En estos ejemplos de aplicaciones utilizaremos el vector de *reset* y el del temporizador, localizados respectivamente en las direcciones 0h y 18h de memoria de programa. El primero permite saltar al programa principal, que inicializará el

procesador, y también las interrupciones. El segundo saltará a la rutina de servicio correspondiente.

La definición de los vectores de interrupción puede hacerse mediante el siguiente segmento de código:

```
*       TABLA DE VECTORES DE INTERRUPCION

        .asect  "vectores",0h       ;RESET
        B       INICIO              ;salta a inicialización del procesador

        .space  160h                ; TINT
        B       GEN_AM              ; salta a la rutina de servicio
```

La directiva *.asect "vectores",0h* define la sección inicializada *"vectores"* con dirección absoluta 0h, y ensambla el vector de *reset* en esta posición. Puesto que el vector del temporizador debe ensamblarse en la misma sección, se utiliza la directiva *.space 160h*, que produce un desplazamiento en el ensamblado determinado por el parámetro en número de bits.

La inicialización de la interrupción del temporizador puede hacerse con el siguiente segmento de código:

```
MASCARA .set    8       ;activa sólo la interrupción del temporizador
TIM     .set    2       ;registro temporizador (página 0)
PRD     .set    3       ;registro periodo (página 0)
IMR     .set    4       ;registro máscara interrupción (página 0)


        DINT            ;bloquea interrupciones hasta terminar inicialización
        LDPK    0       ;direccionamiento directo
        LACK    MASCARA
        SACL    IMR     ;activa interrupción temporizador
        LALK    Kt
        SACL    PRD     ;carga constante del temporizador
        SACL    TIM     ;en los registros asociados
```

A continuación, se muestra el programa en ensamblador que implementa el filtro IIR propuesto.

```
************************************************
* FILTRO IIR 2º ORDEN: ESTRUCTURA TRANSPUESTA
*
*  Entrada datos: PA1
*  Salida datos: PA2
```

```
*  .bss en RAMB1
*
**********************************************
*
NDATOS:          .set    1000        ; número de datos a generar
Kt               .set    1250        ;Kt=50 MHz/(4*10 kHz)=1250
MASCARA          .set    8           ;activa sólo la interrupción del temporizador
TIM              .set    2           ;registro temporizador (página 0)
PRD              .set    3           ;registro periodo (página 0)
IMR              .set    4           ;registro máscara interrupción (página 0)
RAMB1            .set    300h        ;dirección comienzo RAMB1 en memoria de
datos

* CONSTANTES FILTRO EN MEMORIA DATOS
        .bss    A0,1        ; A0/2
        .bss    A1,1        ; A1
        .bss    A2,1        ; A2
        .bss    B0,1        ; B0
        .bss    B1,1        ; B1

* VARIABLES
        .bss    X,1
        .bss    D1,1
        .bss    Y,1
        .bss    D2,1

*       TABLA DE VECTORES DE INTERRUPCION

        .asect  "vectores",0h       ;RESET
        B       INICIO              ;salta a inicialización del procesador

        .space  160h                ; TINT
        B       IIRT                ; salta a la rutina de servicio

        .text

* CONSTANTES DEL FILTRO EN MEMORIA DE PROGRAMA
FCOEF:
        .word   16864       ; A0/2
        .word   6983        ; A1
        .word   -26744      ; A2
        .word   27930       ; -B0
        .word   -23093      ; -B1


*       SUBRUTINA DE SERVICIO DE INTERRUPCION DEL TEMPORIZADOR

IIRT            DINT
                IN      X,PA1
                LAC     X,13
                SACH    X           ;X/8
```


JUAN F. GUERRERO MARTÍNEZ
JOSÉ V. FRANCÉS VILLORA
Curso 2010-2011

```
                LT     X
                MPY    A0              ;P=A0*X
                ZALH   D1              ;ACC=D1
                APAC                   ;ACC= D1+ (A0*X)/2
                MPYA   A1              ;ACC=D1+A0*X : P=A1*X
                SACH   Y
                ZALH   D2              ;ACC=D2
                LTA    Y               ;ACC=D2+A1*X
                MPY    B1              ;ACC=D2+A1*X : P=B1*Y
                MPYA   B2              ; ACC=D2+A1*X+B1*Y : P=B2*Y
                SACH   D1
                LTP    X               ;ACC=B2*Y
                MPY    A2              ;P=A2*X
                APAC                   ;ACC=A2*X+B2*Y
                SACH   D2
                OUT    Y,PA2
                EINT
                RET

* INICIALIZACION DEL PROCESADOR

INICIO:         DINT                   ;bloquea interrupciones hasta terminar inicialización
                LDPK   0               ;direccionamiento directo
                LACK   MASCARA
                SACL   IMR             ;activa interrupción temporizador
                LALK   Kt
                SACL   PRD             ;carga constante del temporizador
                SACL   TIM             ;en los registros asociados
                SSXM                   ;activa modo extensión de signo
                SPM    1               ;elimina signo redundante multiplicador

* INICIALIZACION MEMORIA DATOS
                LDPK   6               ;variables en RAMB1
                LARP   AR0
                LRLK   AR0,RAMB1
                RPTK   4               ;copia coeficientes
                BLKP FCOEF,*+          ;desde ROM
                ZAC
                RPTK   3               ;inicializa variables
                SACH   *+

                LRLK   AR0,NDATOS
                EINT

IIR:            IDLE
                BANZ   IIR,*-
```

4.3.2. Filtrado adaptativo

Los filtros permiten seleccionar determinadas características de la señal de entrada, eliminando las no deseadas. Los coeficientes del filtro determinan su comportamiento, y puesto que el valor de dichos coeficientes se calcula 'a priori', este comportamiento es fijo. No obstante, en algunos casos se desea una salida determinada pero no pueden calcularse los coeficientes del filtro previamente ya que la señal de entrada no es totalmente predecible. Para estos casos, se utilizan técnicas de filtrado adaptativo.

Un filtro adaptativo es un filtro con coeficientes que son modificados por un algoritmo adaptativo para optimizar la respuesta del filtro de acuerdo con un criterio previo. En general, consta de dos partes diferenciadas: un filtro y un algoritmo adaptativo.

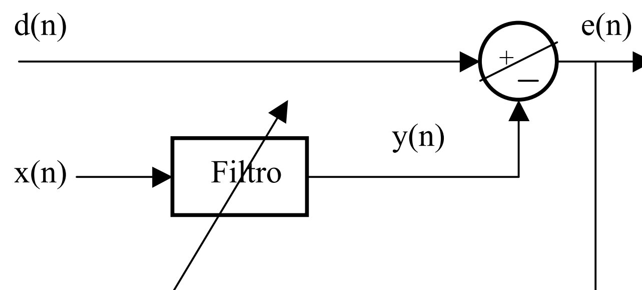


Figura 4.11. Estructura general de un filtro adaptativo.

La señal de entrada al filtro digital, x, es filtrada para producir una salida y. El algoritmo adaptativo ajusta los coeficientes del filtro para minimizar el error, e, entre la salida del filtro, y, y la respuesta deseada del filtro, d.

Los filtros adaptativos pueden usarse en diversas aplicaciones con diferentes configuraciones de entrada y salida. En este caso, nos vamos a centrar en una aplicación de tiempo real para cancelar el ruido de red de 50 Hz. Como algoritmo adaptativo vamos a utilizar el más usual, denominado de Media Cuadrática Mínima (LMS), ya que resulta más simple de diseñar.

En este caso, la señal d está compuesta por la señal útil, s, y una señal de ruido superpuesta, r. El filtro adaptativo modifica la entrada x, que corresponde a la misma interferencia r pero tomada como referencia con valores de amplitud y fase en general diferentes de los de r, y obtiene una salida y que es una estimación del ruido presente en d. En la medida en que esta estimación se aproxime más al valor r, el error de salida, e, será menor (idealmente, el mínimo se obtiene cuando y=r y se anula completamente la interferencia en d).

La estructura del filtro que vamos a implementar es:

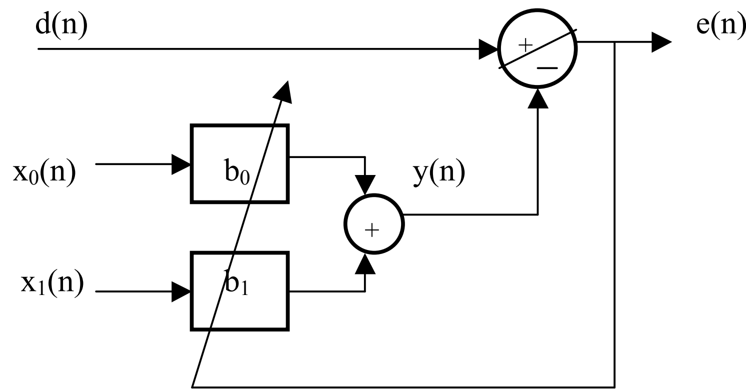


Figura 4.12. Filtro adaptativo para cancelación de la frecuencia de red.

En este caso, el filtro tiene dos coeficientes que multiplican a dos señales de entrada: la señal de red de 50 Hz y la misma señal con un desfase de 90º. Esto permite poder ajustar los dos grados de libertad que tiene el sistema: amplitud y fase.

Las ecuaciones del sistema son:

$$y(n) = b_0 x_0(n) + b_1 x_1(n)$$
$$e(n) = d(n) - y(n)$$

$$b_0(n+1) = b_0(n) + \beta e(n) x_0(n)$$
$$b_1(n+1) = b_1(n) + \beta e(n) x_1(n)$$

donde las dos últimas ecuaciones representan la implementación del algoritmo LMS. En ellas, los valores de los coeficientes se modifican en cada iteración en función de una constante β, denominada constante de adaptación, del valor del error previo y de la muestra de entrada de la señal al filtro.

El TMS320C2x es un procesador de 16 bits en coma fija. Cada muestra de la señal de entrada se representa mediante formato Q15. El intervalo de cuantización vendrá dado por:

$$d = 2^{-15}$$

y se denomina también anchura de cuantización, puesto que los número están cuantizados en pasos de valor d.

Los productos de las multiplicaciones entre datos y coeficientes en el filtro deben ser redondeados o truncados para poder ser almacenados en memoria o en registros de la CPU. Sin embargo, el DSP tiene un acumulador de precisión 32 bits, por

lo que productos de 16x16 bits no precisan ser redondeados hasta que son almacenados en memoria.

La condición a exigir para que los coeficientes del filtro se modifiquen en cada iteración es que el término corrector sea mayor que la mínima resolución:

$$|\beta e(n)x(n)| > \frac{\delta}{2}$$

Puesto que el algoritmo adaptativo está diseñado para minimizar el valor cuadrático medio de la señal de error, e(n) decrece con el tiempo. Si β es demasiado pequeño, la mayor parte del tiempo no se producirá modificación de los coeficientes, por lo que éstos no alcanzarán el valor óptimo y el valor cuadrático medio no convergerá al mínimo.

Las condiciones de convergencia serán:

$$\frac{\delta^2_{\min}}{4\sigma^2_x \varepsilon_{\min}} = \beta^2_{\min}$$

donde s_x^2 representa la potencia de la señal de entrada al filtro y e_{min} es el error cuadrático medio mínimo en el estado estacionario.

$$\beta_{\min} < \beta < \frac{1}{N\sigma^2_x}$$

Para un algoritmo implementado con el TMS320C2x, la longitud de palabra es de 16 bits, por lo que la constante mínima de adaptación, β_{min}, debe tener un valor mayor que cero.

El siguiente programa en ensamblador implementa el filtro adaptativo para eliminación de la interferencia de red. Se utiliza redondeo para minimizar el ruido introducido durante las multiplicaciones y sumas.

```
**********************************************
*       FILTRO ADAPTATIVO
*       QNOTCH FRECUENCIA 50 Hz
**********************************************
*
*  Entradas:
*       PA0: seno de 50Hz, x0(n)
*       PA1: coseno de 50Hz, x1(n)
*       PA2: entrada, d(n)
*  Salidas:
*       PA3: salida filtro adaptativo, err(n)
*  Frecuencia de muestreo: 1 kHz.
```

```
*   .bss en RAMB1
*
*********************************************

NDATOS      .set    2400        ;número datos a generar
Kt          .set    12500       ;Kt=50 MHz/(4*1 KHz)=12500
MASCARA     .set    8           ;activa sólo la interrupción del temporizador
TIM         .set    2           ;registro temporizador (página 0)
PRD         .set    3           ;registro periodo (página 0)
IMR         .set    4           ;registro máscara interrupción (página 0)
RAMB1       .set    300h        ;dirección comienzo RAMB1 en memoria de datos

            .bss    BETA,1      ;constante adaptacion
            .bss    B0,1        ;coeficiente b0 filtro
            .bss    B1,1        ; coeficiente b1 filtro
            .bss    UNO,1       ;factor redondeo
            .bss    X0,1        ;entrada seno
            .bss    X1,1        ;entrada coseno
            .bss    D,1         ;entrada
            .bss    ERR,1       ;señal error: salida
            .bss    B_ERR,1     ;funcion error
            .bss    Y,1         ;salida filtro


*       TABLA DE VECTORES DE INTERRUPCION

        .asect  "vectores",0h   ;RESET
        B       INICIO          ;salta a inicialización del procesador

        .space  160h            ; TINT
        B       ADPFIR          ; salta a la rutina de servicio


        .text

*   VALORES INICIALES DE LAS VARIABLES

V_FIL       .word   7000            ;BETA
            .word   10              ;B0
            .word   10              ;B1
            .word   1               ;UNO



*       SUBRUTINA DE SERVICIO DE INTERRUPCION DEL TEMPORIZADOR

ADPFIR
            DINT
            IN      X0,PA0          ;lee siguiente muestra
            IN      X1,PA1
            IN      D,PA2
```

```
                LAC     UNO,15          ;Redondeo del bit 15 del ACC
                LT      B0
                MPY     X0
                LTA     B1              ;A=B0*X0,
                MPY     X1              ;P=B1*X1
                APAC                    ;A=B0*X0+B1*X1
                SACH    Y

                NEG                     ;calcula ERR=D-Y
                ADDH    D
                SACH    ERR

                LT      BETA
                MPY     ERR
                LTP     X0              ;T=X0, ACC=BETA*ERR

                ADD     UNO,15
                SACH    B_ERR
                MPY     B_ERR           ;P=BETA*ERR*X0
                ZALR    B0              ;A=0,  AH=B0 con redondeo
                APAC                    ;ACC=B0+BETA*ERR*X0
                SACH B0

                LT      X1
                MPY     B_ERR           ;P=BETA*ERR*X1
                ZALR    B1              ;A=0,  AH=B1 con redondeo
                APAC                    ;ACC=B1+BETA*ERR*X1
                SACH B1
                OUT     ERR,PA3
                EINT
                RET


*       INICIALIZACION DEL PROCESADOR

INICIO:         DINT                    ;bloquea interrupciones hasta terminar inicialización
                LDPK    0               ;direccionamiento directo
                LACK    MASCARA
                SACL    IMR             ;activa interrupción temporizador
                LALK    Kt
                SACL    PRD             ;carga constante del temporizador
                SACL    TIM             ;en los registros asociados
                SSXM                    ;activa modo extensión de signo
                SPM     1               ;elimina signo redundante multiplicador
                SOVM                    ;activa aritmética saturada

                LDPK    6               ;DP=6
                LARP    AR0
                LRLK    AR0,RAMB1
                RPTK    3               ;inicializa vars.
```

```
        BLKP   V_FIL,*+      ;desde ROM

        LRLK   AR0,NDATOS
        EINT

BUCLE   IDLE
        BANZ   BUCLE,*-
```

4.3.3. Correlación

La correlación permite encontrar similitudes entre dos conjuntos de datos (correlación cruzada) o entre distintos instantes temporales de la misma señal (autocorrelación). Algunas aplicaciones de la correlación son:

- Sonar, radar.
- Identificación de sistemas.
- Análisis de vibraciones.
- Respuesta evocada en neurofisiología.
- Análisis del ritmo cardíaco.
- Radio-astronomía.
- Modelización del tracto vocal y reconocimiento de palabras en habla.
- Detección y corrección de errores en audio digital.
- Reducción de errores y ruido en comunicaciones digitales.

En este caso, vamos a aplicar la autocorrelación para detectar un determinado patrón temporal de la señal inmersa en ruido. Para ello, compararemos dicho patrón con una señal de entrada en la cual se repite éste cada cierto tiempo. La función autocorrelación indicará el instante temporal en que se produce el patrón. El estimador utilizado para el cálculo es:

$$r_{xx}(m) = \frac{1}{N} \sum_{n=0}^{N-1} x(n)x(n+m)$$

donde N es la anchura del patrón, n barre el patrón y m varía desde *lag*=0 (autocorrelación del patrón consigo mismo) hasta *lag_máximo*. En este caso, puesto que trabajaremos con un fichero de entrada al simulador, *lag_máximo* coincidirá con la longitud del fichero.

Puesto que la expresión del estimador implica una división por N, y el procesador no dispone de instrucciones de división, elegiremos la longitud del patrón igual a una potencia de 2, ya que en este caso la división se reduce a desplazamientos a derechas. Por tanto, tomaremos N = 64 muestras (2^6).

El *buffer* de entrada, de igual longitud que el patrón y situado en el bloque B1 de memoria de datos, se inicializa y se copia en el bloque B0, que posteriormente se

configura como memoria de programa para poder utilizar instrucciones MAC. El *buffer* se actualiza desplazando su contenido y añadiendo una nueva muestra (FIFO). A cada nueva muestra, se realiza la correlación con el patrón.

```
**********************************************
*       AUTOCORRELACION
*       DETECCION DE PATRONES
*
*   Entrada:
*       PA1: señal con ruido

*   Salida:
*       PA2: función autocorrelación
*
*   Frecuencia de muestreo: 1 kHz.
*   La zona de variables estará en RAMB1.
*
*   AR1: puntero buffer
*   AR7: contador datos fichero
*
**********************************************

NDATOS      .set    1000        ;datos a generar
N           .set    64          ;longitud patrón
PATRON      .set    0FF00h      ;dirección comienzo patrón (B0)
BUFFER      .set    300h        ;buffer entrada (B1)
Kt          .set    12500       ;Kt=50 MHz/(4*1 KHz)
MASCARA     .set    8           ;activa sólo la interrupción del temporizador
TIM         .set    2           ;registro temporizador (página 0)
PRD         .set    3           ;registro periodo (página 0)
IMR         .set    4           ;registro máscara interrupción (página 0)

            .bss    DAT_IN,N+1
            .bss    Y_COR,1             ;variable salida correlac.


*       TABLA DE VECTORES DE INTERRUPCION

            .asect  "vectores",0h       ;RESET
            B       INICIO              ;salta a inicialización del procesador

            .space  160h                ; TINT
            B       ACORR               ; salta a la rutina de servicio


            .text

*       SUBRUTINA DE SERVICIO DE INTERRUPCION DEL TEMPORIZADOR
```

```
ACORR       DINT
            LRLK  AR1,BUFFER+N-1
            ZAC
            MPYK  0
            RPTK  N-1
            MACD  PATRON,*-
            APAC
            RPTK  5                 ;divide por N (=2^6)
            SFR
            SACH  Y_COR
            OUT   Y_COR,PA2         ;almacena valor correlación
            IN    DAT_IN,PA1        ;lee siguiente muestra
            EINT
            RET

*       INICIALIZACION DEL PROCESADOR

INICIO:     DINT              ;bloquea interrupciones hasta terminar inicialización
            LDPK  0           ;direccionamiento directo
            LACK  MASCARA
            SACL  IMR         ;activa interrupción temporizador
            LALK  Kt
            SACL  PRD         ;carga constante del temporizador
            SACL  TIM         ;en los registros asociados
            SSXM              ;activa modo extensión de signo
            SPM   1           ;elimina signo redundante multiplicador
            SOVM              ;activa aritmética saturada

            LDPK  6
            LRLK  AR7,NDATOS-1
            LRLK  AR1,BUFFER+N-1
            LARP  AR1
            RPTK  N-1               ;lee patrón
            IN    *-,PA1
            LRLK  AR1,200h+N-1      ;comienzo B0 como DATMEM
            RPTK  N-1               ;copia patrón en B0
            BLKD  BUFFER,*-
            CNFP                    ;patrón en PROGMEM
            EINT

BUCLE       IDLE
            LARP  AR7
            BANZ  BUCLE,*-,1
```

En la siguiente figura se muestra la señal de entrada y la correspondiente función autocorrelación de salida. Los máximos de dicha función corresponden a los instantes de coincidencia de la señal de entrada con el patrón, proporcionando una referencia temporal.

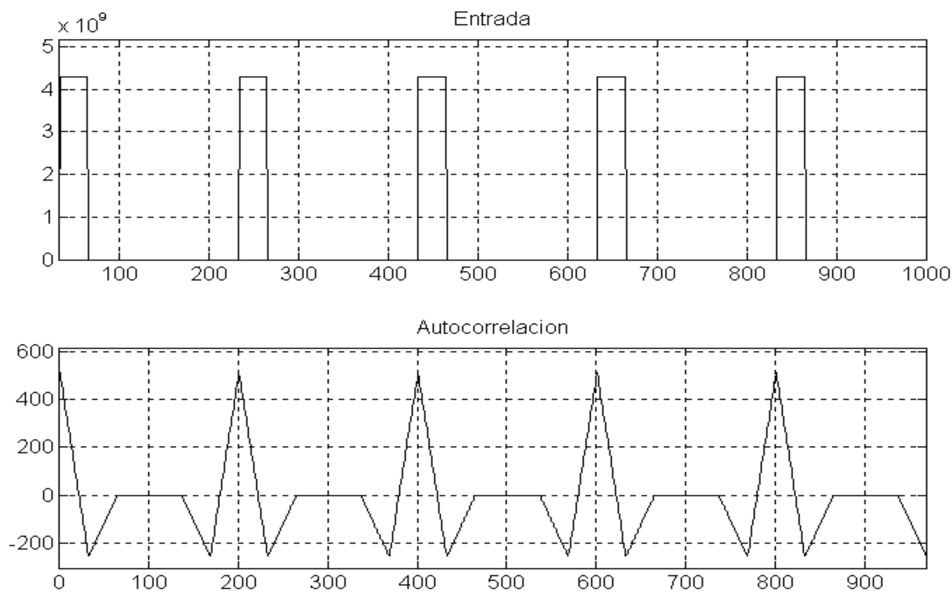


Figura 4.13. Patrón de entrada (pulso cuadrado) y función autocorrelación.

4.3.4. Generación de formas de onda

Mediante la utilización de DSP pueden obtenerse generadores de forma de onda de precisión. En este apartado veremos dos ejemplos, el primero de obtención de onda seno, y el segundo de ruido aleatorio.

Generador de onda seno

Existen diversas formas de generar onda seno digitalmente. Una usual es filtrando el armónico principal de una onda cuadrada. En este caso, utilizaremos la ecuación de un oscilador digital dada por:

$$y(n) = A \cdot y(n-1) + B \cdot y(n-2) + C \cdot x(n-1)$$

donde:

$A = 2 \cos \omega T = 1.618034;$
$B = -1;$
$C = \sin \omega T = 0.587785$

suponiendo frecuencia de la onda seno, f=1 kHz, y frecuencia de muestreo, fm=10 kHz.

Para generar la oscilación inicial, suponemos un impulso unidad en x(0) y obtenemos la onda a partir de n=2. La ecuación se modifica para n≥2:

$$y(n) = A\cdot y(n-1) + B\cdot y(n-2)$$

siendo:

$$y(0) = 0;$$
$$y(1) = C;$$
$$y(2)= A\cdot C.$$

Puesto que A es mayor que la unidad, y vamos a utilizar notación Q15, descompondremos su valor en parte entera y fraccionaria, de manera que A = 1 + A2, donde A2 es la parte fraccionaria de A. En este caso, la ecuación a calcular será:

$$y(n) = y(n-1) + A2\cdot y(n-1) - y(n-2)$$

```
***************************************
*       GEN_SEN.ASM
*
*       fm=10KHz   f=1KHz
*
*       PA1: salida (gen_sen.1kh)
*
*       .bss en RAMB1
*
****************************************

NDATOS      .set    1500            ;datos fichero
Kt          .set    1250            ;Kt=50 MHz/(4*10 KHz)=1250
MASCARA     .set    8               ;activa sólo la interrupción del temporizador
TIM         .set    2               ;registro timer (página 0)
PRD         .set    3               ;registro periodo (página 0)
IMR         .set    4               ;registro máscara interrupción (página 0)


*       VARIABLES
            .bss    A2,1
            .bss    C,1
            .bss    YNM1,1
            .bss    YNM2,1


*       TABLA DE VECTORES DE INTERRUPCION

            .asect  "vectores",0h           ;RESET
            B       INICIO                  ;salta a inicialización del procesador
```

```
            .space  160h                ; TINT
            B       SENO                ; salta a la rutina de servicio


            .text

*       COEFICIENTES EN MEMORIA DE PROGRAMA
COEF        .word   20252               ;A2=0.618034
            .word   19261               ;C=0.587785


*       SUBRUTINA DE SERVICIO DE INTERRUPCION DEL TEMPORIZADOR

SENO        DINT
            ZALH    YNM1                ;y(n) para n>2 | y(n)=y(n-1)+A2*y(n-1)-y(n-2)
            SUBH    YNM2
            MPYK 0
            LTD     YNM1                ;T=Y(N-1); ACC=y(n-1)-y(n-2)
            MPY     A2                  ;P= A2*y(n-1)
            APAC                        ;ACC=y(n-1)+A2*y(n-1)-y(n-2)
            ADLK    8000h               ;redondea ACC
            SACH    YNM1
            OUT     YNM1,PA1
            EINT
            RET


* INICIALIZACION DEL PROCESADOR

INICIO:     DINT            ;bloquea interrupciones hasta terminar inicialización
            LDPK    0       ;direccionamiento directo
            LACK    MASCARA
            SACL    IMR     ;activa interrupción temporizador
            LALK    Kt
            SACL    PRD     ;carga constante del temporizador
            SACL    TIM     ;en los registros asociados
            SSXM            ;activa modo extensión de signo
            SPM     1       ;elimina signo redundante multiplicador
            SOVM            ;activa aritmética saturada
            LDPK    6       ;.bss en B1
            LARP    AR1     ;AR1 para inicializar coef.
            LRLK    AR1,300h    ;copia coeficientes
            RPTK    1
            BLKP    COEF,*+
            LRLK     AR1,NDATOS-4

            ZALH    C
            SACL    YNM2    ;y(0)=0
            OUT     YNM2,PA1
            SACH    YNM2    ;y(1)=C almacenada en y(n-2)
```

```
              OUT   YNM2,PA1
              LT    A2
              MPY   C            ;P=A2*C;  ACC=C
              APAC               ;ACC=C+A2*C
              ADLK  8000h        ;redondea ACC
              SACH  YNM1         ;y(2)=A*C almacenada en y(n-1)
              OUT   YNM1,PA1
              EINT

BUCLE:        IDLE
              BANZ  BUCLE,*-
```

Generador de ruido aleatorio

El generador de ruido aleatoria permite producir una señal de entrada con amplitud constante en todo el ancho de banda (fm/2). Por tanto, puede utilizarse para comprobar la respuesta frecuencial de un sistema, entre otras aplicaciones.

El generador se basa en el método lineal congruencial, que tiene la siguiente expresión:

$$y(n) = (y(n-1)\cdot MULT) + INC \pmod M$$

donde:

- y(n), y(n-1): número aleatorio actual y previo.
- y(0): valor semilla (constante arbitraria)
- MULT: multiplicador (constante)
- INC: incremento (constante)
- M: módulo (en este caso 16, ya que el procesador trabaja con datos de este tamaño)

```
*
*   GENERADOR DE NUMEROS ALEATORIOS PARA EL TMS320C25
*
*   Premisas:    SPM=0 (sin desplazamiento)
*
*   Entrada:     Ninguna
*   Salida:      ACCL = num. aleatorio de 16 bits: PA1
*   Frecuencia de muestreo: 10 kHz.
*   .bss en RAMB1
*
**************************************************************************

Kt          .set    1250         ;Kt=50 MHz/(4*10 KHz)=1250
MASCARA     .set    8            ;activa sólo la interrupción del temporizador
TIM         .set    2            ;registro temporizador (página 0)
PRD         .set    3            ;registro periodo (página 0)
IMR         .set    4            ;registro máscara interrupción (página 0)
NDATOS      .set    2000         ; datos fichero
SEMILLA     .set    21845        ; 65535/3
MULT_C      .set    31821        ; últimos 3 dígitos son par-2-1
INC_C       .set    13849

*       VARIABLES
            .bss    YN,1
            .bss    MULT,1
            .bss    INC,1

*       TABLA DE VECTORES DE INTERRUPCION

            .asect  "vectores",0h       ;RESET
            B       INICIO              ;salta a inicialización del procesador
            .space  160h                ; TINT
            B       Rand16              ; salta a la rutina de servicio

            .text

*       SUBRUTINA DE SERVICIO DE INTERRUPCION DEL TEMPORIZADOR

Rand16      DINT
            LT      YN
            MPY     MULT
            PAC
            ADD     INC
            SACL    YN
            OUT     YN,PA1
            EINT
            RET

* INICIALIZACION DEL PROCESADOR
```

```
INICIO:     DINT            ;bloquea interrupciones hasta terminar inicialización
            LDPK  0         ;direccionamiento directo
            LACK  MASCARA
            SACL  IMR       ;activa interrupción temporizador
            LALK  Kt
            SACL  PRD       ;carga constante del temporizador
            SACL  TIM       ;en los registros asociados
            SSXM            ;activa modo extensión de signo
            ROVM            ;permite overflow para implementar el módulo
            LDPK  6         ;.bss en RAMB1

            LALK  SEMILLA
            SACL  YN        ;inicializa y(0)
            LALK  MULT_C
            SACL  MULT      ;inicializa MULT
            LALK  INC_C
            SACL  INC       ;inicializa INC

            LARP  AR1
            LRLK  AR1,NDATOS-4

            EINT

BUCLE:      IDLE
            BANZ  BUCLE,*-
```

4.3.5. Modulación

Dentro del campo de comunicaciones, existe un punto relacionado con la transmisión segura de datos. Hay varias técnicas desarrolladas en esta línea, tales como la encriptación previa de la información a transmitir. Otra aproximación consiste en modificar la frecuencia de la portadora con el tiempo, por lo que la señal se está desintonizando continuamente desde el punto de vista del receptor intruso, mientras que el receptor autorizado conoce el patrón de cambio y puede seguir la transmisión. Esta técnica se conoce como salto de frecuencia (*frequency hopping*), y la portadora se modifica aleatoriamente. Para capturar la información transmitida, el receptor debe conocer el número posible de portadoras y sus valores en frecuencia, la secuencia usada para seleccionarlas y los intervalos de tiempo entre un cambio y otro. De este modo, el receptor es capaz de demodular la señal, proporcionando un cierto nivel de seguridad.

Vamos a abordar un ejemplo de esta técnica utilizando modulación AM. En primer lugar, desarrollaremos el programa de modulación AM, que responde al siguiente esquema:

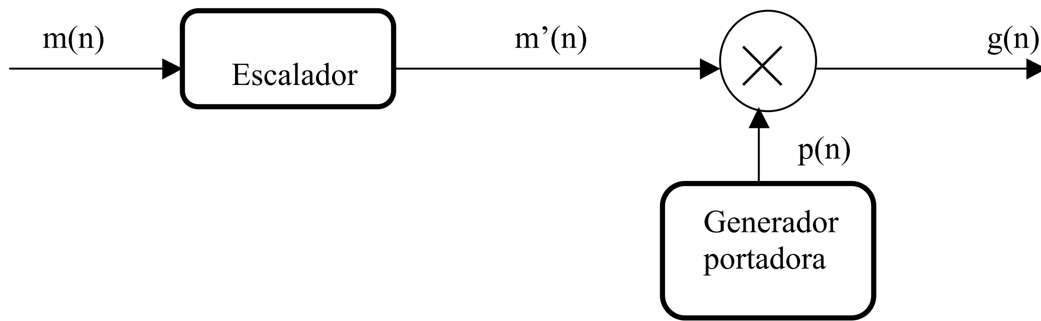


Figura 4.14. Esquema de modulación AM: señal moduladora (m), moduladora escalada (m'), portadora (p) y salida modulada (g).

La señal del ADC se escala para asegurar que siempre es positiva y evitar distorsiones. La salida del escalador se multiplica por la portadora, generada mediante un oscilador senoidal como el desarrollado en el apartado de generación de ondas.

El siguiente programa produce una modulación AM. El temporizador se carga con el periodo de muestreo de la portadora (Tmp), y se activa la interrupción. Cada vez que se produce, la rutina de servicio genera una muestra de la portadora. Se toma una muestra de la moduladora cada f_{mp}/f_{mo} y se modula la portadora. Finalmente, la muestra de la portadora modulada se saca por el DAC.

Vamos a utilizar en este ejemplo una frecuencia de muestreo de la portadora, fmp = 30 kHz, frecuencia de la portadora, fp = 12 kHz, frecuencia de muestreo de la moduladora, fmo = 10 kHz y frecuencia de la moduladora, fo = 2 kHz. La portadora se genera a partir del programa de generación de onda senoidal visto en la sección 7.2.3. Los parámetros calculados para este caso serán:

$A= 2 \cos \omega T = -1.618034$
$C= \sin \omega T = 0.587785$
$B= -1$

Puesto que A es mayor que la unidad, para poder codificar los parámetros en formato Q15 los utilizaremos divididos por 2:

A/2= -0.889017 = -26510 (Q15)
C/2 = 9630 (Q15);
B/2= -16384

```
**************************************************************************
*
*  MODULADOR AM PARA EL TMS320C25
*
*  DATOS:  fmp=30KHz   fp=12KHz   fmo=10KHz  fo=2KHz
*  Entradas:      Moduladora: (PA1).
*  Salida:        Señal modulada: (PA2)
*  bss en RAMB1
*
**************************************************************************

*         CONSTANTES

NDATOS      .set     1000          ;nº de muestras a generar
FMP         .set     30            ;Fmp=30KHz
FMO         .set     10            ;Fmo=10KHz
FMP_FMO     .set     FMP/FMO
Kt          .set     417           ;Kt=50 MHz/(4*30 KHz)=417
MASCARA     .set     8             ;habilita interrupción del timer
TIM         .set     2             ;registro timer (página 0)
PRD         .set     3             ;registro periodo (página 0)
IMR         .set     4             ;registro máscara interrup. (página 0)
OFFSET      .set     4000h         ;offset para hacer moduladora positiva
RAMB1       .set     300h          ;dirección comienzo bloque B1 en mem. datos


*         VARIABLES
            .bss     A2,1          ;coeficientes portadora
            .bss     B2,1
            .bss     C2,1
            .bss     RM,1          ;razón de muestreo Fmp/Fmo
            .bss     XN,1          ;muestra moduladora
            .bss     YM,1          ;salida modulada
            .bss     YNM1,1
            .bss     YNM2,1


*         TABLA DE VECTORES DE INTERRUPCION

            .asect   "reset",0h
            B        MOD_AM              ;RESET

            .space   160h                ;salta hasta vector de TINT (en bits)
            B        GEN_AM              ;TINT  salta a la rutina de servicio


            .text
```

```
COEF        .word   -26510              ;A2=-0.889017
            .word   -16384              ;B2=-0.5
            .word   9630                ;C2=0.587785


*       SUBRUTINA DE SERVICIO DE INTERRUPCION DEL TIMER

GEN_AM      DINT
*
*       Genera portadora
*
            ZAC
            LT      YNM2
            MPY     B2
            LTD     YNM1        ;T=Y(N-1).  Actualiza histórico
            MPY     A2          ;P= A2*y(n-1)
            APAC                ;ACC=A2*y(n-1)+B2*y(n-2)
            SACH    YNM1,2      ;compensa coeficientes y signo redundante

*       Determina si hay que muestrear moduladora (cada Fmp/Fmo)

            LACK    FMP_FMO
            SUB     RM
            BGZ     DAC
            ZAC
            SACL    RM          ;RM=0
            IN      XN,PA1      ;lee portadora desde ADC
            LAC     XN
            SFR
            ADLK    OFFSET
            SACL    XN          ;moduladora positiva siempre

*       Modula y saca señal AM por DAC

DAC         LAC     RM
            ADDK    1
            SACL    RM
            LT      XN
            MPY     YNM1
            PAC
            SACH    YM
            OUT     YM,PA2      ;salida DAC
            EINT
            RET



*       PROGRAMA PRINCIPAL

MOD_AM      DINT                ;bloquea interrup. hasta terminar inicialización
            SSXM
```

```
            ROVM
            LDPK  0              ;inicialización del timer e interrup.
            LACK  MASCARA
            SACL  IMR            ;activa sólo interr. timer
            LALK  Kt
            SACL  PRD            ;carga Tmp
            SACL  TIM

            LDPK  6              ;.bss en B1
            LARP  AR1            ;AR1 para inicializar coef.
            LRLK  AR1,RAMB1      ;copia coeficientes
            RPTK  2
            BLKP  COEF,*+
            LRLK   AR1,NDATOS-4

            ZALH  C2             ;inicializa histórico
            SACH  YNM2           ;y(1)=C almacenada en y(n-2). Mantener dividida
                                 ;por 2 para evitar overflow.
            LT    C2
            MPY   A2             ;P=A2*C2;  ACC=A1*C2
            PAC                  ;ACC=A2*C2
            SACH  YNM1,2         ;y(2)=A*C almacenada en y(n-1)

            IN    XN,PA1         ;lee portadora desde ADC
            LAC   XN
            SFR
            ADLK  OFFSET
            SACL  XN             ;moduladora positiva siempre

            LACK  FMP_FMO
            SACL  RM             ;inicializa razón muestreo

            EINT                 ;activa interrupciones

BUCLE       IDLE
            BANZ  BUCLE,*-
```