Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO FIN DE GRADO

ESTUDIO DEL DSPTMS320C50 Y TRATAMIENTO DE SEÑALES BASADAS EN ÉL.

Alumno/Alumna: Santa Cruz, Granado, Iñigo
Director/Directora (1): Legarreta Etxagibel, Jon Josu

Curso: 2020-2021

Fecha: 22, Junio, 2021

INDICE. Pág.

1. RESUMEN.

1. Objetivos.

- Conocer el funcionamiento de un DSP.
- Estudiar las aplicaciones de los DSP's.
- Desarrollar alguna aplicación.

2. Descripción general.

El DSP es un microprocesador orientado al procesado digital de señales. La señal a procesar debe ser pasada a digital para poder ser procesada por el DSP.

El proceso con el cuál aplicamos a la señal se lleva a cabo por el computador digital y por esto es llamado procesado digital de señales (DSP). En los sistemas DSP modernos se utiliza un solo microcomputador para el procesado digital de señales. Estos chips especiales son conocidos como procesadores de señales digitales o DSPs.

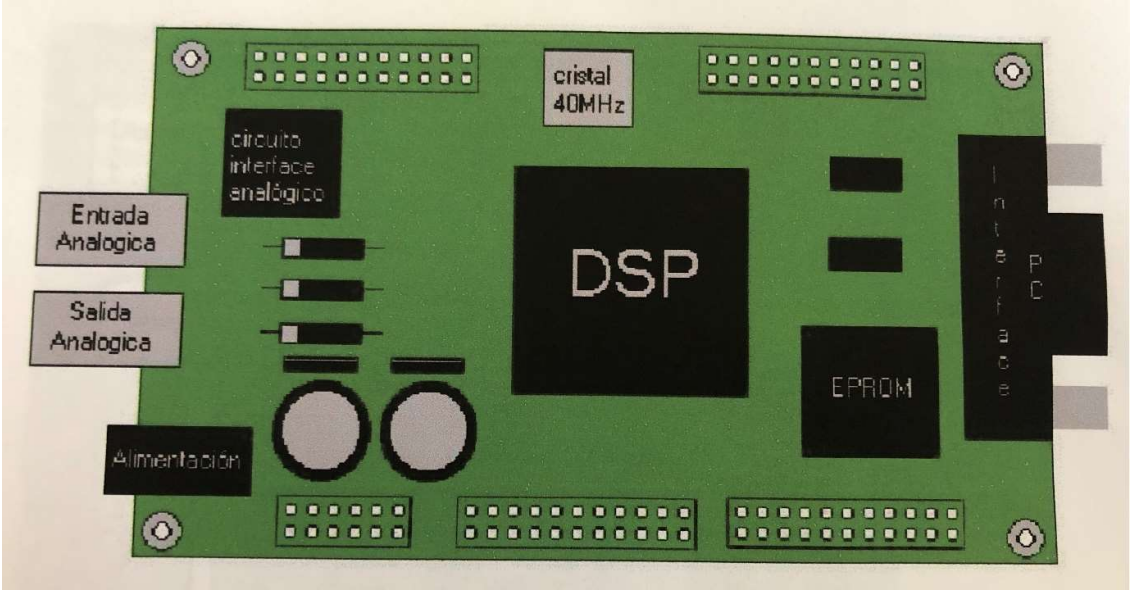
Una vez que la señal ha sido procesada por el DSP, todavía se encuentra en la forma de una secuencia de números y necesitan ser convertidas a señales analógicas antes de ser pasadas a un actuador por ejemplo un altavoz. Este proceso es llamado conversión D/A.

Los DSP tienen las siguientes ventajas frente a otro tipo de microprocesadores:

- Realización, simulación y emulación en tiempo real.
- Flexibilidad.
- Fiabilidad.
- Reducción de costes del sistema.
- Aumento de las prestaciones del sistema.

La placa a utilizar está compuesta principalmente por un chip de 132 patillas de la familia TMS320 de Texas Instruments, más concretamente, el TMS320C50PQ, un puerto serie RS-232 para la comunicación con el PC, una entrada para alimentación de 9V, dos conectadores estándar RCA, uno para entradas y otro para salidas analógicas, que proporcionan una conexión directa a un micrófono, altavoz u otro aparato analógico.

También contiene una memoria EPROM (erasable programable read only memory) que permite la comunicación con el PC, 10K de palabras de memoria RAM, un integrado TLC32040C que realiza conversiones analógico-digital y digital-analógica con 14 bits de resolución y un cristal de 40Mhz el cual define la base de tiempos del sistema. Todo esto se percibe en el dibujo que viene a continuación:



Para realizar cualquier aplicación con el DSP, primero debemos introducir un programa con las instrucciones necesarias que hagan posible el correcto funcionamiento de la aplicación.

Estas instrucciones se escribirán en el editor de MS-DOS, creando un archivo con extensión asm. Mediante el comando dsk5a seguido del nombre del programa se llama al ensamblador DSK.

Por ejemplo, para el programa seno, habría que escribir:

dsk5a seno.asm

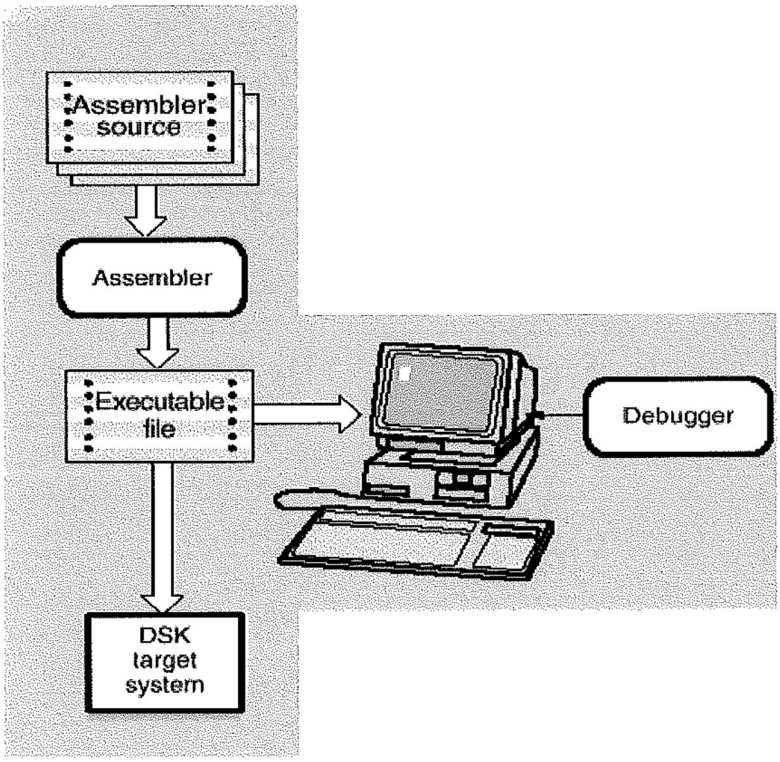
Al introducir este comando, el debugger crea un archivo ejecutable con extensión dsk. Al mismo tiempo, el ensamblador crea un archivo en forma de lista, que puede ser útil porque contiene una lista de los símbolos y códigos sin resolver.

A continuación, se ejecuta el debugger mediante el comando dsk5d, y ya se puede cargar el programa realizado.

Teclear:

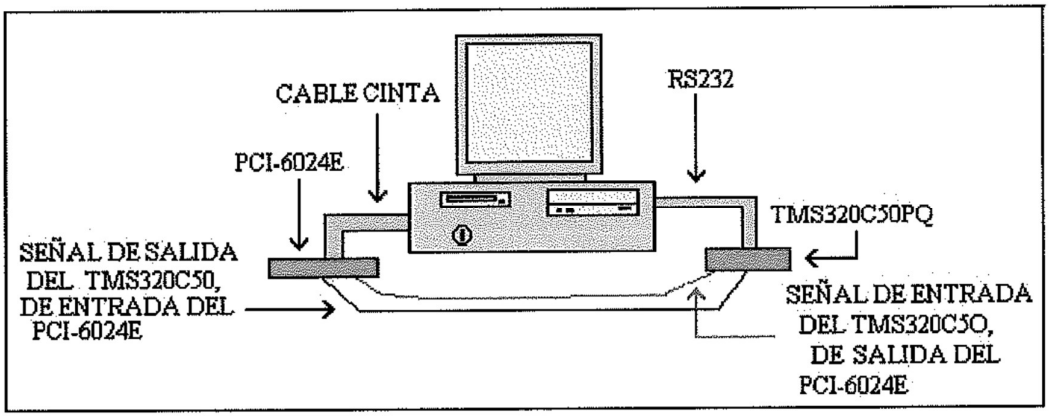
dsk5d

Para que el programa se ejecute en la placa debemos teclear desde Dos, C: \dsk\dsk51 + nombre del programa. La placa únicamente ejecutara el programa, si esta está conectada al ordenador mediante el puerto serie y el programa se ejecuta en él.



Para poder visualizar la salida del DSP, vamos a utilizar el Labview lo que nos va a posibilitar que podamos introducir al DSP cualquier señal que queramos.

Para comunicar el DSP con la CPU en la cual se esté ejecutando el Labview se va a utilizar la tarjeta de adquisición de datos PCI-6024E de National Instruments como se puede ver en el esquema que viene a continuación:



3 Aplicaciones típicas.

- Para aplicaciones generales.
  Filtrado digital y adaptativo, Convolucion, Correlacion, Transformada de Hilbert, rapida de fourier, de coseno discreto y de Hartley, Generacion de formas de onda y Windowing.

- En instrumentacion.
  Análisis espectral, Generación de funciones, Realización de patrones, Análisis de transitorios, Filtrado digital.

- Control.
  Control de motores, maquinas, impresoras láser, robots, servos, disquetes.

- Automoción.
  Frenos antibloqueo y antideslizamiento, Analisis vibratorios, Sistemas Gps, Radios digitales, Suspensión activa, Control del airbag, Sistemas de diagnóstico, Detectores de radar

- Telecomunicaciones.
  Teléfonos inalambricos, Decodificadores

4 Herramientas a utilizar en la elaboración.

• Kit DSP TMS320C50 de Texas Instruments que consta de:

- Una placa con el DSP y demás componentes.
- Una fuente de alimentación de 9V.
- Un cable RS-232.
- Cables auxiliares para introducir o medir señales.

• PC con los siguientes programas:

- Ensamblador DSP Starter Kit 5a (DSK5a)
- Debugger o eliminador de errores DSP Starter Kit 5d (DSK5d)
- Microsoft Word
- Microsoft PowerPoint.
- Labview.

• Tarjeta de adquisición de datos PCI - 6024E.

1. SUMMARY.

1. Goals.

- Know the operation of a DSP.
- Study the applications of DSP's..
- Develop an application

2. General description.

The DSP is a microprocessor oriented to digital signal processing. The signal to be processed must be converted to digital in order to be processed by the DSP.

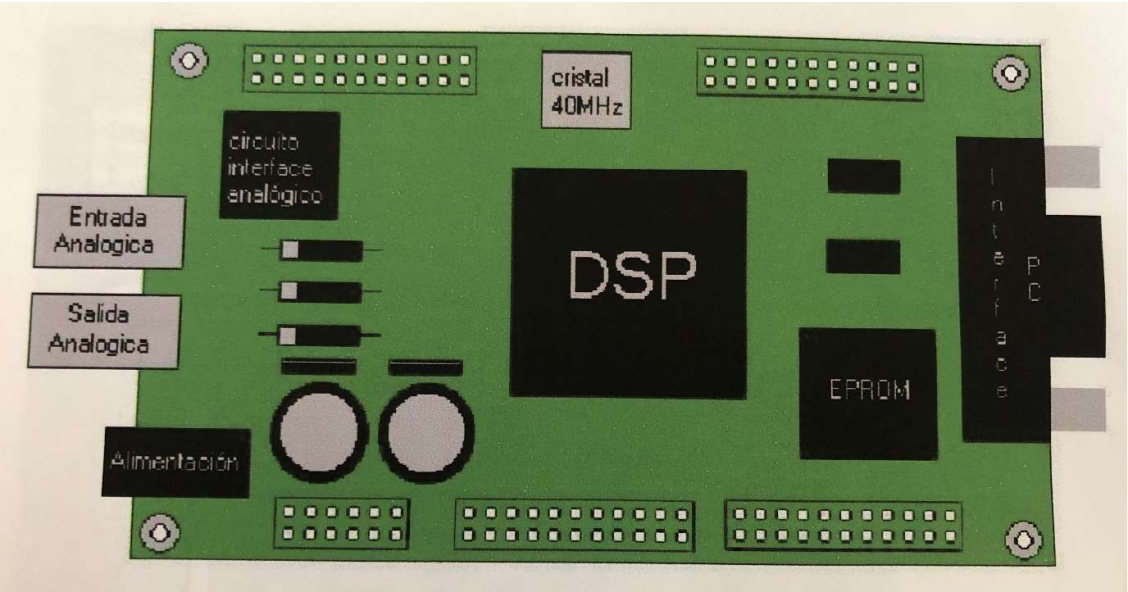
The process we apply the signal is carried out by the digital computer and for this it is called digital signal processing (DSP). In modern DSP systems a single microcomputer is used for the digital signal processing. These special chips are known as digital signal processors or DSPs.

Once the signal has been processed by the DSP, it is still in the form of a sequence of numbers and needs to be converted to analog signals before being passed to an actuator for example a loudspeaker. This process is called D / A conversion.

The DSPs have the following advantages over other types of microprocessors:

- Realization, simulation and emulation in real time.
- Flexibility.
- Reliability.
- Reduction of system costs.
- Increase in system benefits.

The board to be used consists mainly of a 132-pin chip from the Texas Instruments TMS320 family, more specifically, the TMS320C50PQ, an RS-232 serial port for communication with the PC, one input for 9V power supply, two standard RCA connectors, one for inputs and one for analog outputs, providing a direct connection to a microphone, speaker or other analog device. It also contains an EPROM (erasable programmable read only memory) memory that allows communication with the PC, the OK of RAM memory words, an integrated TLC32040C that performs analog-digital and digital-analog conversions with 14 bits of resolution and a crystal of 40Mhz which defines the time base of the system. All this can be seen in the following drawing:



To carry out any application with the DSP, we must first introduce a program with the necessary instructions that make the correct operation of the application possible.

These instructions will be written in the MS-DOS editor, creating a file with an asm extension. Using the dsk5a command followed by the name of the program, the DSK assembler is called.

For example, for the sine program, you would write:

dsk5a seno.asm

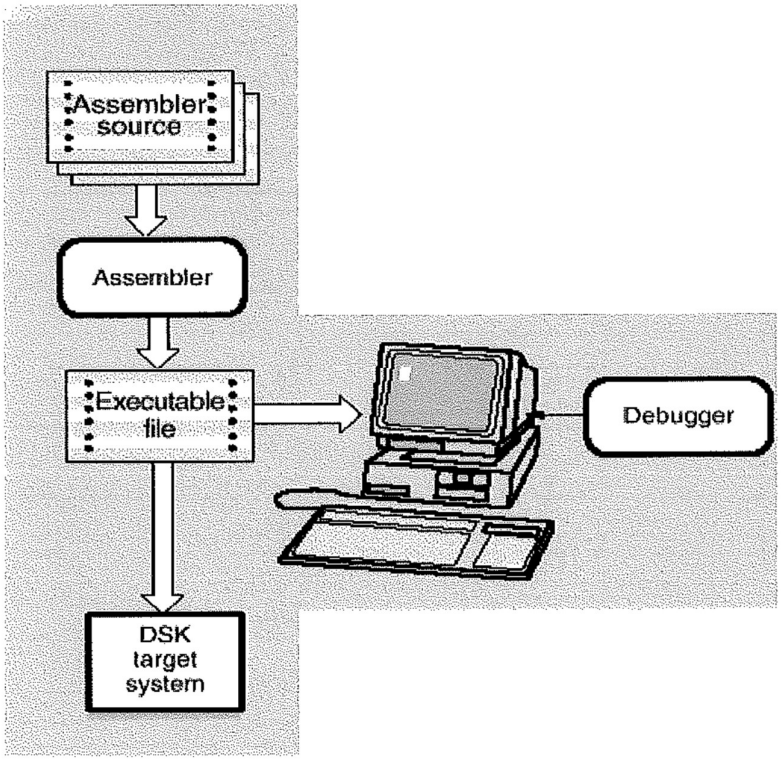
By entering this command, the debugger creates an executable file with a dsk extension. At the same time, the assembler creates a file in the form of a list, which can be useful because it contains a list of the unresolved symbols and codes.

Next, the debugger is run using the dsk5d command, and the completed program can now be loaded.

Key:

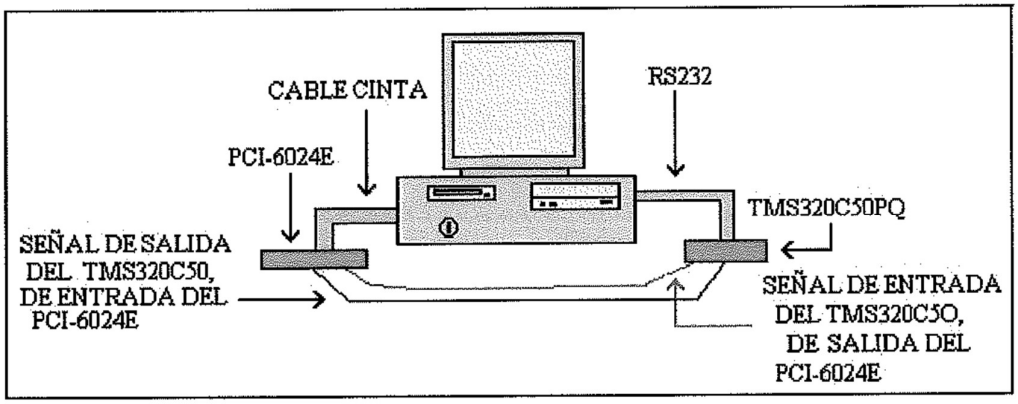
dsk5d

For the program to run on the board we must type from Dos, C: \ dsk \ dsk51 + name of the program. The board will only run the program if it is connected to the computer through the serial port and the program is running on it.



In order to visualize the output of the DSP, we are going to use Labview, which will enable us to introduce any signal we want to the DSP.

To communicate the DSP with the CPU on which Labview is running, the National Instruments PCI-6024E data acquisition card will be used as can be seen in the diagram below:



3 Typical applications.

- General applications.
  Digital and Adaptive Filtering, Convolution, Correlation, Hilbert Transform, Fast Fourier, Discrete Cosine and Hartley, Waveform Generation and Windowing.

- In instrumentation.
  Spectral analysis, Function generation, Pattern making, Transient analysis, Digital filtering.

- Control.
  Control of motors, machines, laser printers, robots, servos, floppy disks.

- Automotive.
  Anti-lock and anti-slip brakes, Vibration analysis, Gps systems, Digital radios, Active suspension, Airbag control, Diagnostic systems, Radar detectors ...

- Telecommunications.
  Cordless phones, Set-top boxes

3.4 Tools to be used in the elaboration.

• Texas Instruments DSP Kit TMS320C50 consisting of:

- A board with the DSP and other components.
- A 9V power supply.
- An RS-232 cable.
- Auxiliary cables to introduce or measure signals.

• PC with the following programs:

- Assembler DSP Starter Kit 5a (DSK5a)
- Debugger or debugger DSP Starter Kit 5d (DSK5d)
- Microsoft Word
- Microsoft PowerPoint.
- Labview.

• PCI - 6024E Data Acquisition Card.

1. LABURPENA.

1. Helburuak.

- DSP baten funtzionamendua ezagutzea.
- DSPen aplikazioak aztertzea.
- Garatu aplikazio bat.

2. Deskribapen orokorra.

DSP seinale digitala prozesatzera bideratutako mikroprozesadorea da. Prozesatu beharreko seinalea digitalera bihurtu behar da DSP-k prozesatu ahal izateko.

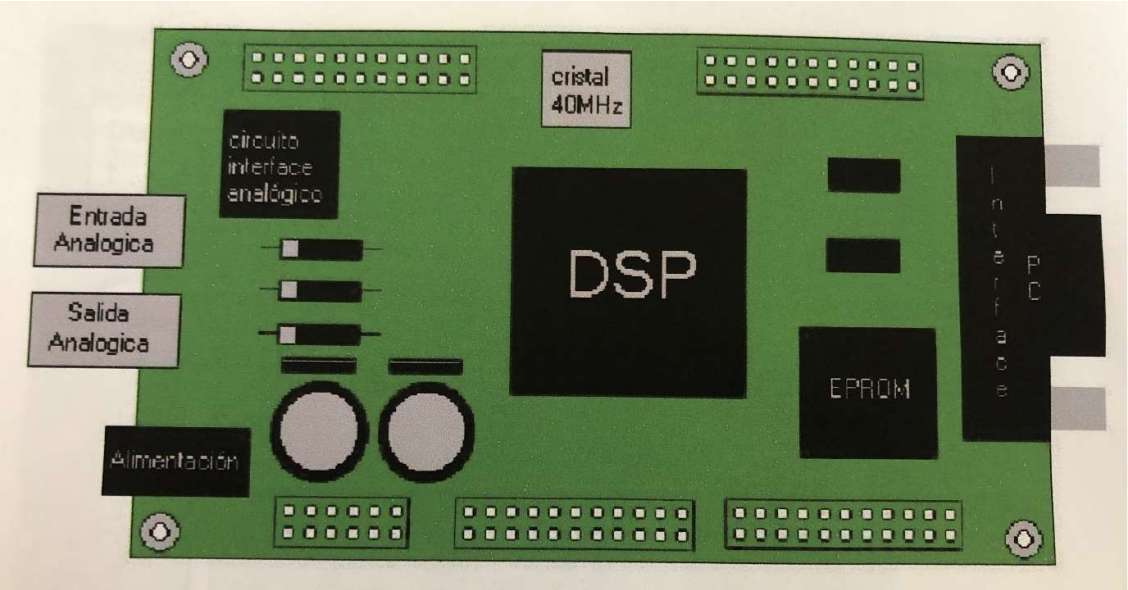
Seinalea aplikatzen dugun prozesua ordenagailu digitalak egiten du eta horretarako seinaleen tratamendu digitala (DSP) deitzen zaio. DSP sistema modernoetan mikroordenagailu bakarra erabiltzen da seinale digitala prozesatzeko. Txip berezi hauek seinale digitalen prozesadore edo DSP izenez ezagutzen dira.

DSP-k seinalea prozesatu ondoren, zenbaki segida baten moduan dago oraindik eta seinale analogiko bihurtu behar da eragingailu batera pasa aurretik, adibidez bozgorailu batera. Prozesu horri D / A bihurketa deritzo.

DSPek abantaila hauek dituzte beste mikroprozesadore mota batzuen aurrean:

- Realizazioa, simulazioa eta emulazioa denbora errealean.
- Malgutasuna.
- Fidagarritasuna.
- Sistemaren kostuak murriztea.
- Sistemaren onurak handitzea.

Erabiliko den taula Texas Instruments-eko TMS320 familiako 132 pineko txip batez osatuta dago, zehazki, TMS320C50PQ, RS-232 serieko ataka Ordenagailua, sarrera bat 9V-ko elikadurarako, bi RCA konektore estandar, bata sarreretarako eta bestea irteera analogikoetarako, mikrofonoarekin, bozgorailuarekin edo beste gailu analogiko batekin konexio zuzena eskainiz. Ordenagailuarekin komunikazioa ahalbidetzen duen EPROM (ezaba daitekeen irakurgai soilik programagarria den memoria) memoria dauka, RAM memoria hitzen OK, TLC32040C integratua, 14 bit bereizmeneko eta 40Mhz-ko kristala duen bihurketa analogiko-digitala eta digital-analogikoa egiten duena. sistemaren denbora oinarria definitzen duena. Hori guztia hurrengo marrazkian ikus daiteke:



DSPrekin edozein aplikazio burutzeko, aplikazioaren funtzionamendu zuzena ahalbidetzen duten beharrezko argibideak dituen programa aurkeztu behar dugu lehenik.

Argibide hauek MS-DOS editorean idatziko dira, asm luzapena duen fitxategia sortuz. Programaren izena eta ondoren dsk5a komandoa erabiliz, DSK muntatzaileari deitzen zaio.

Adibidez, sinus programarako, hau idatziko zenuke:

dsk5a seno.asm

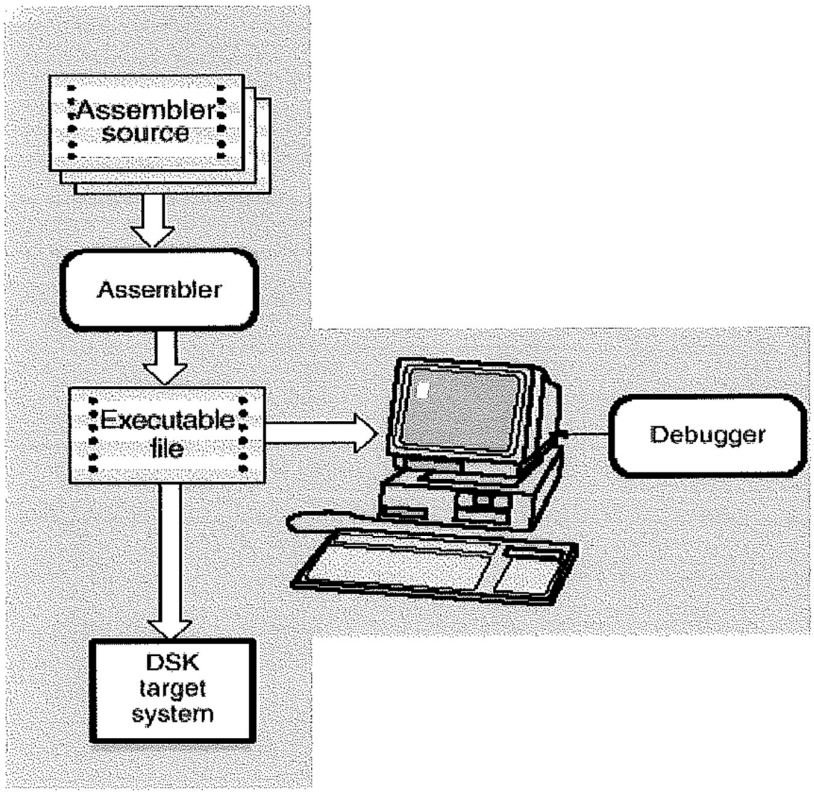
Komando hau sartzean, arazleak dsk luzapeneko fitxategi exekutagarria sortzen du. Aldi berean, muntatzaileak fitxategi bat sortzen du zerrenda moduan, eta hori erabilgarria izan daiteke konpondu gabeko sinbolo eta kodeen zerrenda duelako.

Ondoren, araztailea dsk5d komandoa erabiliz exekutatzen da eta amaitutako programa karga daiteke.

Gakoa:

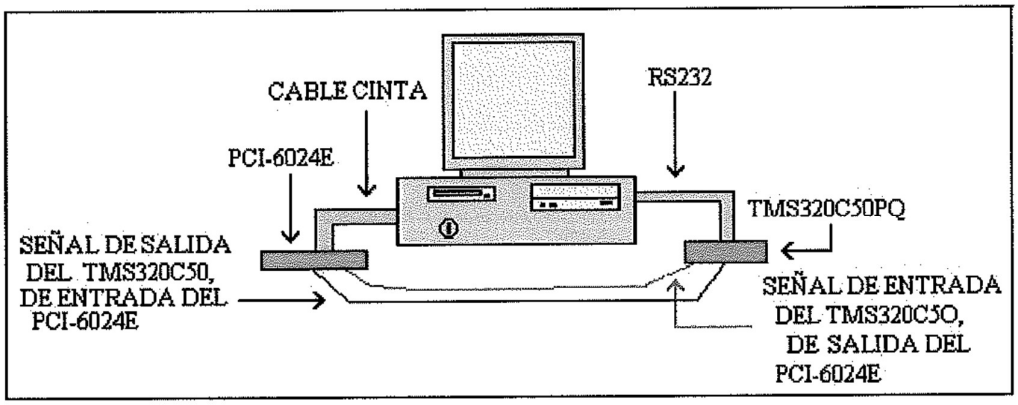
dsk5d

Programa taulan exekutatzeko Dos, C: \ dsk \ dsk51 + programaren izena idatzi behar dugu. Plakak programa exekutatuko du serieko ataka bidez ordenagailura konektatuta badago eta programa bertan exekutatzen ari bada.



DSPren irteera bistaratzeko, Labview erabiliko dugu, eta horrek DSPri nahi dugun seinalea sartzeko aukera emango digu.

DSP Labview abian den CPUarekin komunikatzeko, National Instruments PCI-6024E datuak eskuratzeko txartela erabiliko da beheko diagraman ikus daitekeen moduan:



3 Aplikazio tipikoak.

- Aplikazio orokorretarako.
  Iragazketa Digitala eta Egokigarria, Konvoluzioa, Korrelazioa, Hilbert Transformatua, Fast Fourier, Kosinua eta Hartley Diskretuak, Uhin Formen Sorkuntza eta Windowing.

- Instrumentazioan.
  Analisi espektrala, Funtzioen sorrera, Ereduak egitea, Analisi iragankorra, Iragazki digitala.

- Kontrola.
  Motorrak, makinak, laser inprimagailuak, robotak, serboak, disketeak kontrolatzea.

- Automobilgintza.
  Blokeoaren aurkako eta irristagaitzaren aurkako balaztak, bibrazioen analisia, Gps sistemak, irrati digitalak, esekidura aktiboa, airbagen kontrola, sistema diagnostikoak, radar detektagailuak ..

- Telekomunikazioak.
  Haririk gabeko telefonoak, Decodificadoreak

4 Elaborazioan erabiliko diren tresnak.

• Texas Instruments DSP Kit TMS320C50 osatua:

- DSP eta beste osagai batzuk dituen taula.
- 9V-ko elikadura.
- RS-232 kable bat.
- Seinaleak sartzeko edo neurtzeko kable laguntzaileak.

• Ordenagailua programa hauekin:

- DSP Starter Kit 5a (DSK5a) Ensanbladorea
- Debugger edo arazgailua DSP Starter Kit 5d (DSK5d)
- Microsoft Word
- Microsoft PowerPoint.
- Labview.

• PCI - 6024E Datuak eskuratzeko txartela.

2. MEMORIA.

A- Documentación sobre procesado digital de señales.

1. Introducción a los filtros digitales.

Los filtros digitales tienen 2 usos generales:

(1) Separación de señales que están combinadas.

(2) Restauración de señales que han sido distorsionadas de alguna.

Los filtros analógicos pueden ser usados para estas rutinas; sin embargo, los filtros digitales pueden alcanzar resultados muy superiores.

En este apartado se describen los parámetros que se deben tener en cuenta para estos filtros.

1.1 Filtros Básicos.

Los filtros digitales son una parte importante del DSP. En verdad, su extraordinaria ejecutabilidad es una de las principales razones por las que el DSP ha llegado a ser tan popular. Como se ha mencionado en la introducción, los filtros tienen 2 usos: separación de la señal y restauración de la señal.

1 .La separación de la señal es necesitada cuando una señal está contaminada con una interferencia, ruido, u otras señales. Por ejemplo, un instrumento para medir la actividad eléctrica de un feto en el interior de una madre.

La señal puede estar igualmente corrompida por la respiración y el latido de la madre. Un filtro ha de ser usado para separar estas señales y para que puedan ser analizadas individualmente.

2. La restauración de la señal se usa cuando una señal ha sido distorsionada en alguna medida. Por ejemplo, una grabación de audio realizada con un equipo de mala calidad puede ser filtrada para que mejore el sonido. Otro ejemplo es la depuración de una imagen adquirida con una lente inapropiada, o una cámara que vibre.

Estos problemas se pueden solucionar con filtros analógicos o digitales.

Los filtros analógicos son más baratos, rápidos, y tienen un gran rango dinámico tanto en amplitud como en frecuencia.

Los filtros digitales, en comparación, son muy superiores en el nivel de ejecuciones que pueden realizar.

Por ejemplo, un filtro pasa bajo con una ganancia de 1 +/- 0.0002 de continua a 1000 Hz, y una ganancia de menos de 0.0002 para frecuencias inferiores a 1001 Hz. La transición entera ocurre dentro de sólo 1 Hz. Esto no se debe esperar de un circuito amplificador operacional. Los filtros digitales pueden ejecutar operaciones miles de veces de mejor que los filtros analógicos. En los filtros analógicos, el problema reside tanto en las limitaciones de manipulación de la electrónica, como en la exactitud y la estabilidad de las resistencias y los condensadores. En contraste, los filtros digitales son tan buenos que el desempeño del filtro está frecuentemente ignorado. El énfasis se hace en las limitaciones de las señales, y los asuntos teóricos suponiendo su procesamiento.

Es común en los DSP decir que la entrada de un filtro y las señales de salida están en el dominio de tiempo. Esto es porque las señales son habitualmente creadas tomando muestras en intervalos regulares de tiempo. Pero ésta no es la única forma que el muestreo producir. La segunda forma más común de muestreo está en intervalos iguales en espacio. Un ejemplo, sean lecturas tomadas simultáneamente en un gran número de sensores de tensión que separados un centímetro están situados a lo largo de un ala de un avión.

Muchos otros dominios son posibles; Sin embargo, el tiempo y el espacio son los más comunes. Cuando se ve el término dominio del tiempo en DSP, hay que recordar que puede referirse a muestras tomadas de continuo, o puede ser una referencia general a cualquier dominio en que las muestras son tomadas.

Como se muestra en figura 1, cada filtro lineal tiene una **respuesta de impulso**, una **respuesta de paso** y una **respuesta de frecuencia**. Cada uno de estas respuestas contiene información completa acerca del filtro, pero de una forma diferente. Si uno de lo tres está especificado, entonces los otros dos son fijos y pueden ser calculado directamente. Estas tres representaciones son importantes, ya que describen cómo reaccionará el filtro bajo diferentes circunstancias.

La forma más directa de implementar un filtro digital es hacer la convolución con la respuesta de impulso del filtro digital. Todos los filtros lineales posibles pueden estar hechos de esta manera previa en DSP. Cuando la respuesta de impulso está usada de este modo, los diseñadores del filtro le dan un nombre especial: El **núcleo del filtro**.

Existe también otra forma de hacer filtros digitales, llamada **recursión**.

Cuando un filtro es implementado por convolución, cada muestra en la salida se calcula promediando las muestras en la entrada, y sumándolas. Los filtros recursivos son una extensión de esto, usando los valores previamente calculados, además de los puntos de la entrada.

En lugar de usar un filtro núcleo, los filtros recursivos están definidos por un juego de coeficientes de recursión. Este método se discutirá en detalle más adelante. Por ahora, el punto más importante es que todos los filtros lineales tienen una respuesta de impulso, incluso si no se usa para implementar el filtro. Para encontrar la respuesta de impulso de un filtro recursivo, solamente hay que alimentar un impulso, y ver cuál es la salida. Las respuestas de impulso de filtros recursivos están compuestas de sinusoidales que decaen exponencialmente en amplitud. En principio, esto hace sus respuestas de impulso infinitamente largas. Sin embargo, la amplitud repentinamente desciende por debajo del ruido completo del sistema, y las muestras remanentes pueden ser ignoradas.

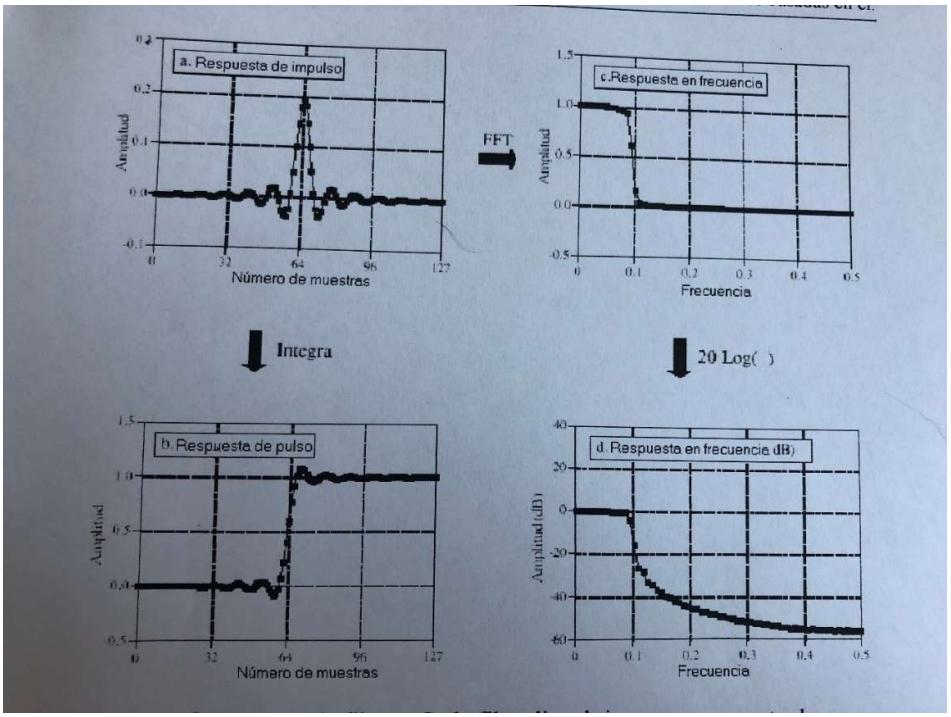


Figura 1 Parámetros de filtros. Cada filtro lineal tiene una respuesta de impulso, de paso y de frecuencia. La respuesta de paso, (b), se puede encontrar mediante la integración discreta de la respuesta de impulso, (a), La respuesta en frecuencia se puede encontrar de la respuesta en impulso mediante la Transformada Rápida de Fourier (FFT), y se pueden visualizar tanto en escala lineal, (c), o en decibelios, (d).

Debido a estas características, los filtros recursivos son también llamados filtros de **Respuesta Infinita al Impulso** o filtros **IIR**. En comparación, los filtros que son realizados mediante convolución son llamados de **Respuesta Finita** al **Impulso** o filtros **FIR**.

La respuesta impulso es la salida de un sistema cuando la entrada es un impulso. De esta misma manera, la respuesta escalón es la salida cuando la entrada es un escalón (también llamadas un borde, y una respuesta borde). Como el escalón es la integral del impulso, la respuesta escalón es la integral de impulso de respuesta. Esto provee dos caminos para encontrar la respuesta escalón:

(1) mandar una onda escalón al filtro y ver cuál es la salida, ó

(2) Integrar la respuesta del impulso. (Para ser matemáticamente correcto: la integración es usada con señales continuas, mientras la integración discreta, esto es, en una suma corriente, se usan señales discretas.

La respuesta en frecuencia se puede encontrar tomando la DFT (usando el algoritmo FFT) de la respuesta de impulso. La respuesta en frecuencia puede ser dibujada en una accisa vetical lineal, como en (c), ó en una escala logarítmica (en decibelios), tal y como se muestra en (d). La escala lineal es la mejor forma de mostrar el rizado de la banda de paso mientras la escala en decibelios se necesita para mostrar la atenuación de la banda de rechazo.

Ecuación 1. Definición de decibelios. Los decibelios es la forma de expresar una propor9ión entre dos señales. Ya sea de potencia P_1 y P_2

$$dB = 10 \log_{10} P_2/P_1$$

$$dB = 20 \log_{10} A_2/A_1$$

Las ecuaciones anteriores son en logaritmo en base 10; sin embargo, muchos lenguajes de programación solo aceptan funciones logaritmos en base *e* (llamado el logaritmo natural, $\log_e x$ ó ln x). El logaritmo natural puede ser usado cambiando ecuaciones:

$dB = 4.342945 \log_e (P2/P1)$ y $dB = 8.685890 \log_e (A2/A1)$.

Como los decibelios son una forma de expresar la relación entre dos señales, son ideales para describir la ganancia del sistema, esto es, la relación entre la señal de entrada y salida. Sin embargo, los ingenieros también usan los decibelios para especificar la amplitud (ó energía) de una señal simple, reverenciándolo a algún estándar. Por ejemplo, el término: **dBV** significa que la señal haya sido referenciada a una señal 1 Voltio.

Del mismo modo, **dBm** indica una señal de referencia produciendo 1 mw en carga de 600 ohmios (en aproximadamente 0.78 voltios).

Dos cosas que hay que tener en cuenta acerca de los decibelios:

La primera, -3dB significa que la amplitud se reduce a O.707 (y la potencia se reduce a 0.5).

La, segunda, es tener en cuenta las siguientes conversiones entre decibelios y la amplitud:

60dB = 1000
40dB = 100
20dB = 10
OdB = 1
-20dB = 0.1
-40dB = 0.01
-60dB = 0.001

1.2 Como la Información se representa en Señales.

La parte más importante de cualquier tarea DSP es comprender como la información es contenida en las señales con las que se trabaja. Existen muchas formas para que la información pueda ser contenida en una señal. Esto es especialmente cierto si la señal es sintética. Por ejemplo, considerando todos los tipos de modulación existentes: AM, FM, ancho de banda simple, modulación en pulsos codificados, modulación en ancho de pulso, etc.

Afortunadamente, solo existen dos caminos que son comunes para que la información sea representada en señales que se produzcan naturalmente. Estos son llamados: **información representada en el dominio de tiempo**, e **información representada en el dominio de la frecuencia**.

La información representada en el **dominio del tiempo** describe cuando ocurre algo y cuál es la amplitud del evento. Por ejemplo, en un experimento en el que se estudie la luz del sol saliendo. La luz del sol saliente es medida y grabada una vez cada segundo. Cada muestra de la señal indica lo que está pasando en cada instante, y su nivel. Si sucede una llamarada solar, la señal provee la información directamente en el tiempo en el que está ocurriendo, la duración, el desarrollo en el tiempo, etc. Cada muestra contiene información que es interpretada sin referencia a otra señal. Incluso si solo se tiene una muestra de esta señal, se puede conocer algo acerca de lo que sé está midiendo. Este es el camino más simple para que la información sea contenida en una señal.

En contraste, la información representada en el dominio de la frecuencia es más indirecta. Existen muchas cosas en nuestro universo que son periódicas. Por ejemplo, al golpe de una copa de vino con un algo metálico, esta oscilará, produciendo un sonido vibrante; un reloj de péndulo oscilando; la rotación de estrellas y planetas en sus ejes y alrededor de ellos, y así sucesivamente. Mediante la medida de la frecuencia, fase, y amplitud de estos fenómenos periódicos, se puede obtener la información contenida en estas señales. En el caso del sonido producido por la vibración en el vaso, la frecuencia fundamental y los armónicos de la vibración periódica están relacionadas con la masa y la elasticidad del material. Una muestra simple, en sí misma, no contiene información sobre el fenómeno periódico, y por tanto tampoco contiene información sobre la copa. La información contenida hay que sacarla entre muchos puntos de la señal.

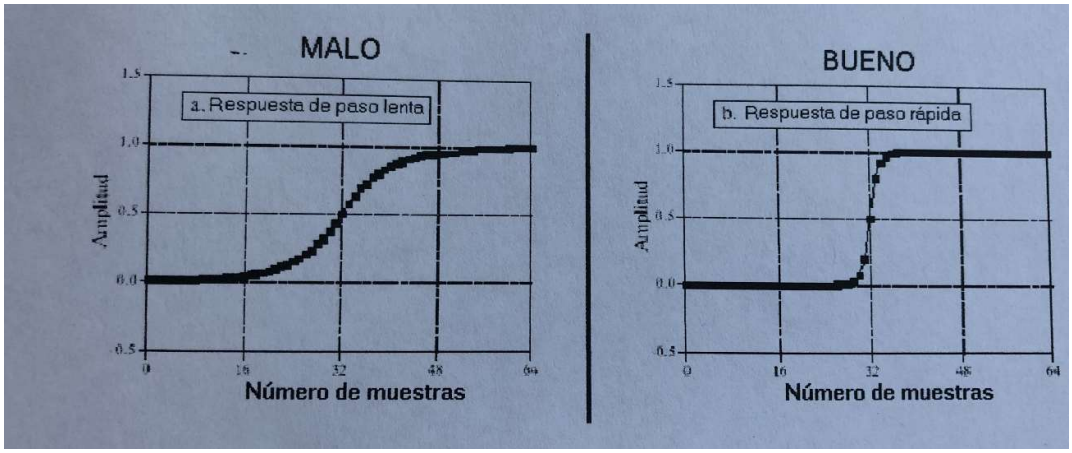
Esto nos enseña la importancia de las respuestas en paso y en frecuencia. La respuesta de paso describe como la información representada en el dominio del tiempo está siendo modificada por el sistema. En contraste, la respuesta en frecuencia muestra como la información representada en el dominio de la frecuencia es cambiada. Esta distinción es muy importante en el diseño de los filtros, ya que no es posible optimizar un filtro para ambas aplicaciones. Una buena presentación en el dominio del tiempo produce una mala presentación en el dominio de la frecuencia, y viceversa. Si se ha diseñado un filtro para eliminar el ruido de una señal de un electrocardiograma (información representada en el dominio del tiempo), la respuesta de paso es el parámetro más importante, y la respuesta en frecuencia tiene poca importancia. Si se tiene la intención de diseñar un filtro digital para un audímetro (con la información en el dominio de la frecuencia), la respuesta en frecuencia es muy importante, mientras que la respuesta de paso no importa. A continuación se muestra que hace a un filtro óptimo para aplicaciones en el dominio del tiempo o en el dominio de la frecuencia.

1.3 Parámetros en el dominio del tiempo.

Puede que no sea obvio que la respuesta en paso concierna a filtros en el dominio del tiempo. La respuesta está en la forma en que la mente humana entiende y procesa la información. Hay que recordar que la respuesta paso y la respuesta en frecuencia contienen idéntica información. La respuesta de paso es más usada en el dominio del tiempo ya que muestra la forma en que los humanos perciben la información contenida en las señales.

Por ejemplo, si se quiere analizar una señal de origen desconocido, la primera cosa que se ha de hacer es dividir la señal en porciones de características similares. Esto es un proceso que la mente humana lo realiza automáticamente. Algunas de las porciones pueden tener poca actividad; otras pueden tener picos de gran amplitud; otras pueden estar calladas. Esta segmentación se lleva a cabo identificando los puntos que separan las porciones. Aquí es donde viene la función paso. La función paso es el camino más puro de representar una división entre dos porciones diferentes. Se puede marcar cuando un evento comienza, o cuando un evento termina. Informa si cualquier cosa que está a la derecha es de algún modo diferente a cualquier cosa que está a la izquierda. Esta es la forma en que la mente humana percibe la información en el dominio del tiempo: un grupo de funciones pasa dividiendo la información en porciones de características similares. La respuesta de paso es importante ya que describe como las líneas divisorias son modificadas por el filtro.

Los parámetros de la respuesta de paso que son importantes en el diseño de filtros se muestran en la figura 2. Para distinguir eventos en una señal, la duración de la respuesta de paso debe ser más corta que el espacio de los eventos. Esto dictamina que la respuesta de paso deba ser tan rápida como sea posible. Esto se muestra en las figuras (a) y (b). El camino más común para especificar el **tiempo de subida** es tener en cuenta el número de muestras entre el 10% y el 90% de los niveles de amplitud. Debido a diferentes razones no siempre es posible un tiempo de subida rápido, estas son: reducción del ruido, limitaciones inherentes a los sistemas de adquisición de datos, anulación del aliasing, etc.



Las figuras (c) y (d) muestran el siguiente parámetro que es importante: él rebose en la respuesta de paso. Él rebose normalmente ha de ser eliminado debido a que cambia la amplitud de las muestras en la señal; esta es una distorsión básica de la información contenida en el dominio del tiempo.

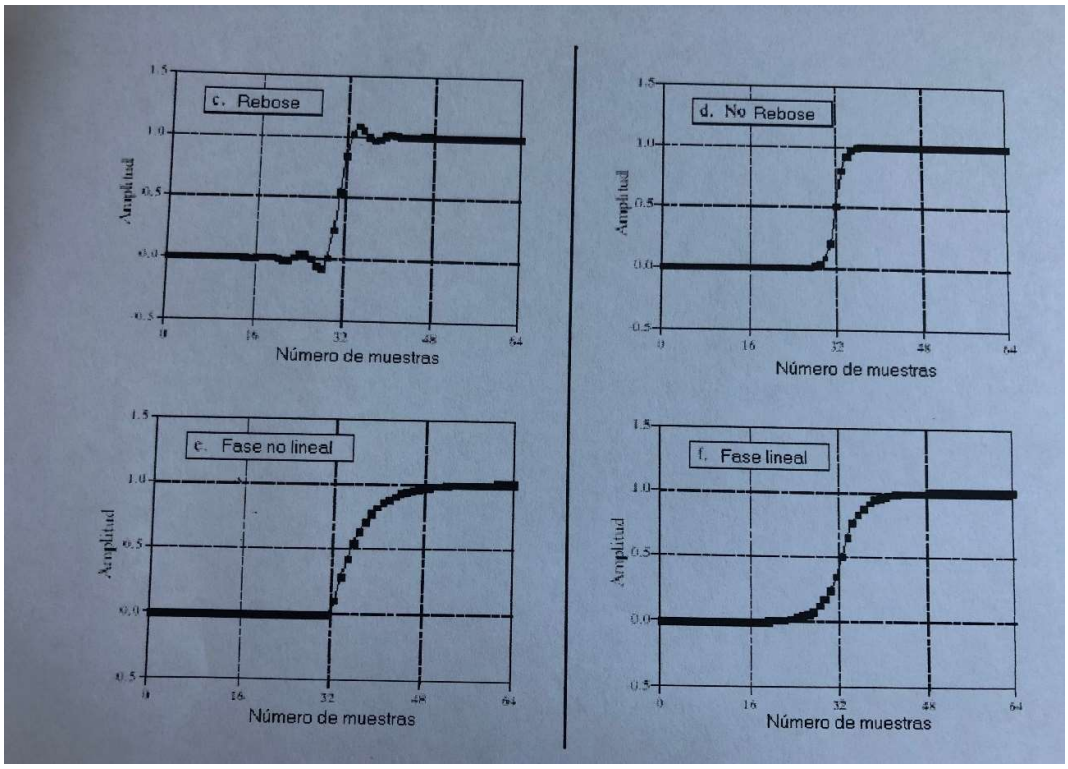


Figura 2. Parámetros para evaluar la ejecución en el dominio del tiempo. La respuesta de paso es usada para medir como de bien se ejecuta un a filtro en el dominio del tiempo. Hay tres parámetros importantes (1) la velocidad de transición (tiempo de subida), mostrado en (a) y (b), (2) rebose, mostrado en (c) y (d), y (3) linealidad de fase (simetría entre la parte altas y bajas del paso), mostrado en (e) y (f).

A menudo se observa que la mitad superior de la respuesta de paso es simétrica con la mitad inferior, tal como se muestra en (e) y (f). Esta simetría se necesita para hacer que los bordes de subida sean iguales que los de bajada. Esta simetría es llamada **fase lineal**, porque la respuesta en frecuencia está en fase con una línea recta. Hay que entender bien estos tres parámetros; son la clave para evaluar los filtros en el dominio en el tiempo.

1.4 Parámetros en el dominio de la frecuencia

La figura 3 muestra las 4 respuestas en frecuencia básicas. Estos filtros se utilizan para permitir que algunas frecuencias pasen inalteradas, mientras que otras frecuencias se bloquean completamente.

Los **Paso-banda** hacen referencia a aquellas frecuencias que permiten pasar, mientras que los **Rechazo-banda** a aquellas frecuencias que bloquean. La **banda de transición** está en el medio. Un rápido descenso significa que la banda de transición es muy estrecha. La división entre la banda de paso y la banda de transición se llama **frecuencia de corte.** En los diseños de los filtros analógicos, la frecuencia de corte generalmente es definida para estar situada donde la amplitud se reduce al 0.707 (-3dB).

Los filtros digitales están menos estandarizados, y es común ver que la frecuencia de corte está definida en el 99%, 90%, 70.7%, y 50% de los niveles de amplitud.

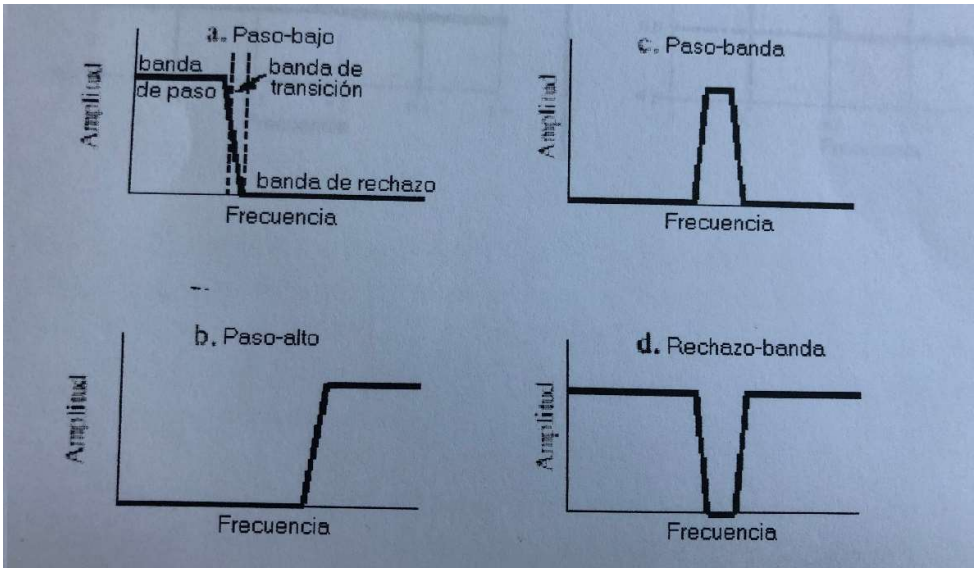


Figura 3. Las cuatro respuestas en frecuencia comunes. Los filtros de dominio de la frecuencia son generalmente utilizados para que pasen ciertas frecuencias (paso-bajo, paso-alto ó paso-banda) ó que bloqueen ciertas otras (rechazo-banda).

La figura 4 muestra tres parámetros que miden como trabaja un filtro en el dominio de la frecuencia. Para separar frecuencias muy juntas entre sí, el filtro debe tener una rápida caída, como el mostrado en (a) y (b). Para que las frecuencias del paso-banda pasen inalteradas a través del filtro, no debe haber rizado en la banda de paso, tal y como se muestra en (c) y (d). Por último, para un rechazo adecuado de las frecuencias del Rechazo-banda, es necesario tener una buena atenuación de la banda de rechazo, vista en (e) y (t).

No aparece la fase en estos parámetros porque no es importante en la mayoría de las aplicaciones de dominio de la frecuencia. Por ejemplo, la fase de una señal de radio es casi completamente ruido, y contiene muy poca información útil. Además, si la fase es importante, es muy fácil realizar filtros digitales con una respuesta de fase perfecta, por Ej. , Todas las frecuencias pasan a través del filtro con un desplazamiento de fase cero. En comparación, los filtros analógicos son muy inferiores a este respecto.

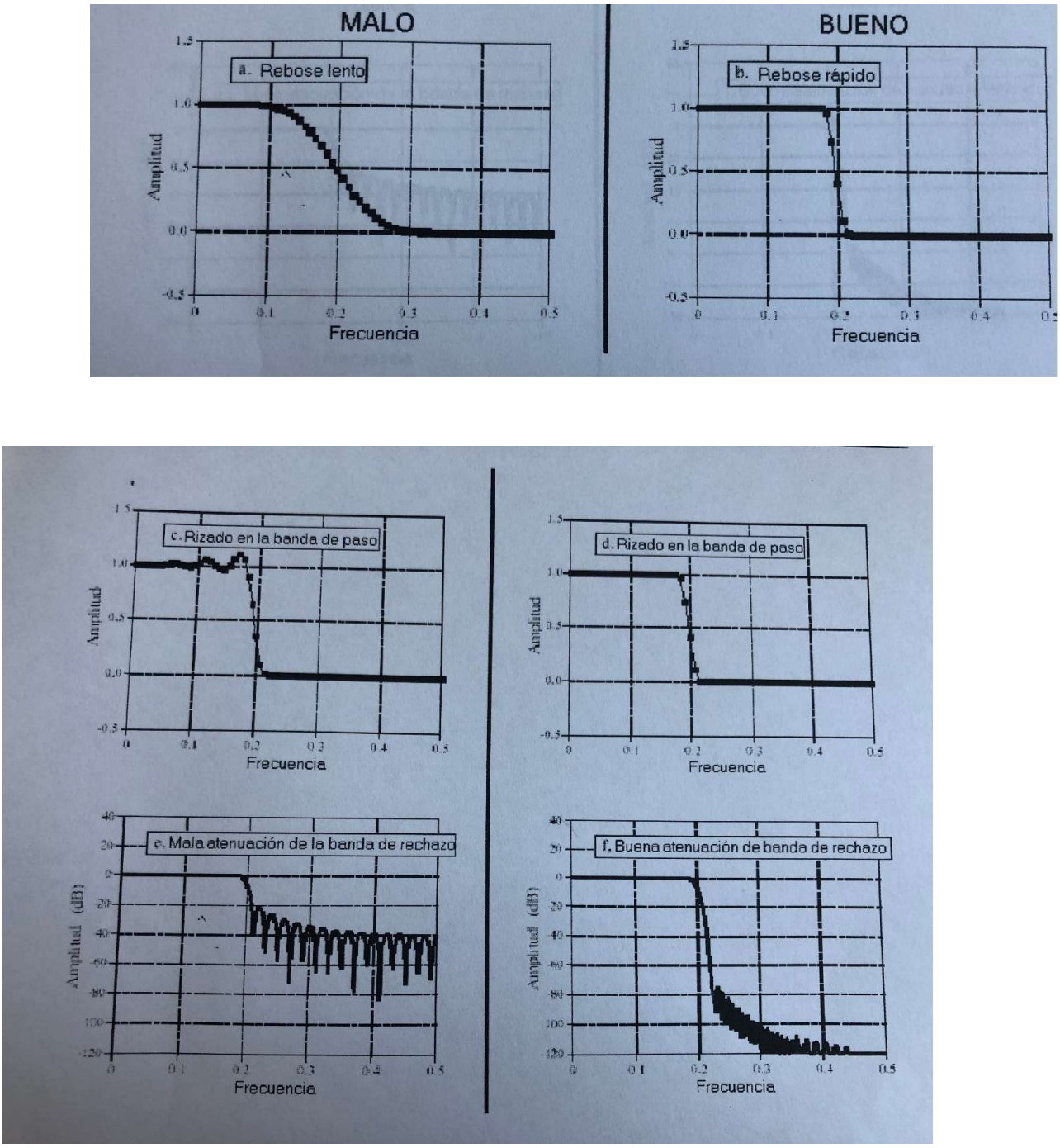


Figura 4. Parámetros para evaluación de la ejecución en el dominio del tiempo. La respuesta en frecuencia es mostrada para filtros paso-bajo. Hay tres parámetros importantes: (1) filo del rebose, mostrado en (a) y (b), (2) rizado del paso-banda, mostrado en (c) y (d), y (3) atenuación de la banda de rechazo, mostrada en (e) y (f).

Anteriormente se ha visto como el DFT convierte la respuesta impulso de un sistema en su respuesta en frecuencia. La forma más rápida de calcular la DFT es mediante del algoritmo FFT. Empezado con un filtro de N muestras de longitud, el FFT calcula el espectro de frecuencia consistente de una parte real de N puntos y una parte de imaginaria de N puntos. Solo las muestras de O a N/2 de las partes reales e imaginarias de la FFT contienen información útil; los otros puntos están duplicados (frecuencias negativas) y pueden ser ignorados. Como las partes reales e imaginarias son difíciles de entender para los humanos, normalmente se convierten a notación polar. Esto proporciona las señales de magnitud y fase, yendo desde la muestra O a la muestra N/2 (por Ej. N/2 + 1 muestras en cada señal). Hay que tener en cuenta que ninguna frecuencia mayor que la ½ de la ratio de la muestra puede aparecer en los datos muestreados.

El número de las muestras usadas para representar la respuesta del impulso puede ser arbitrariamente largo. Por ejemplo, suponiendo que se quiera encontrar la respuesta en frecuencia de 80 puntos. Como el FFT solo trabaja con señales que son potencias de 2, se necesita añadir 48 ceros a la señal para ofrecer una longitud de 128 muestras. Este relleno con ceros no cambia la respuesta del impulso. Para entender porque es esto, hay que pensar que pasa a estos ceros añadidos cuando la señal de entrada es convolucionada con la respuesta impulso del sistema. Los ceros añadidos simplemente desaparecen en la convolución, y no afectan al resultado.

La idea importante es que las de respuestas impulso largas generan un corto espacio de puntos de datos en la respuesta en frecuencia. Según esto, existen más muestras repartidas entre DC y ½ del coeficiente de muestreo.

Llevando esto al extremo, si la respuesta impulso es rellenada con un número infinito de ceros, los puntos de datos en la respuesta en frecuencia están infinitamente más juntos, por ej. , Una línea continua. En otras palabras, la respuesta en frecuencia de un filtro es realmente una señal continua entre DC y ½ del ratio de muestreo. La salida del DFT es una muestra de esta línea continua.

Hay que tener en cuenta que los buenos o malos parámetros discutidos en este apartado son solo generalizaciones. Muchas señales no se pueden meter en ninguna categoría.

Por ejemplo, considerando que una señal de electrocardiograma esta contaminada con una interferencia de 50 Hz. La información es codificada en el dominio del tiempo, pero la interferencia esta más relacionada con el dominio de la frecuencia.

1.5 Filtros Paso-alto, Paso-banda y Rechazo-banda.

Estos filtros son diseñados a partir de un filtro paso-bajo, y después se convierten en la respuesta deseada. Es por ello que en la mayoría de los textos en donde se habla del diseño de filtros solo se dan ejemplos de filtros paso-bajo. Existen dos métodos para la conversión paso-bajo a paso-alto: **inversión espectral** y **reverso espectral**. Ambos son usados por igual.

Un ejemplo de la inversión espectral se muestra en la figura 5. La figura (a) muestra un filtro núcleo paso bajo llamado ventana síncrona. Este filtro tiene una longitud de 51 puntos, aun cuando muchas de las muestras tienen un valor muy pequeño para que aparezca cero en este gráfico.

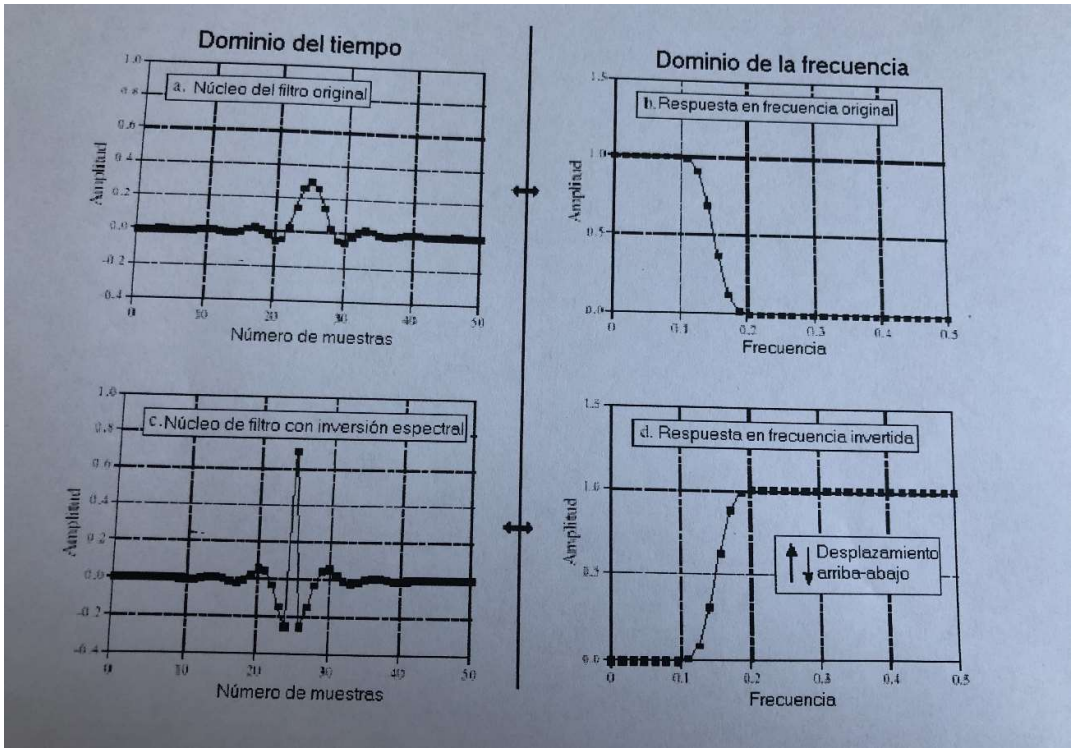


Figura 5.Ejemplo de inversión espectral. El núcleo del filtro paso-bajo en (a) tiene la respuesta en frecuencia mostrada en (b). Un núcleo de filtro paso-alto, (c), es formado mediante el cambio del signo en cada muestra en (a), y añadiendo uno a la muestra en el centro de simetría. Esta acción en el dominio del tiempo invierte el espectro en frecuencia (esto es, desplazamiento de arriba a abajo), tal y como se muestra en la respuesta en frecuencia paso-alto en (d). La correspondiente respuesta en frecuencia se muestra en (b), añadiendo 13 ceros al filtro del núcleo y tornando 64 puntos de FFT. Para pasar de un filtro paso-bajo a un filtro paso-alto se debe cambiar el signo de cada muestra del filtro y añadir uno a la muestra en el centro de simetría. Esto se observa en el filtro paso-alto de (c), con la respuesta en frecuencia mostrada en (d). La inversión espectral invierte la respuesta en frecuencia de arriba a abajo, cambiando los pasa-bandas a rechazo-bandas, y viceversa. Es decir, cambiando un filtro de paso-bajo a paso-alto, paso-alto a paso-bajo, paso- banda a rechazo-banda, o rechazo-banda a paso-banda.

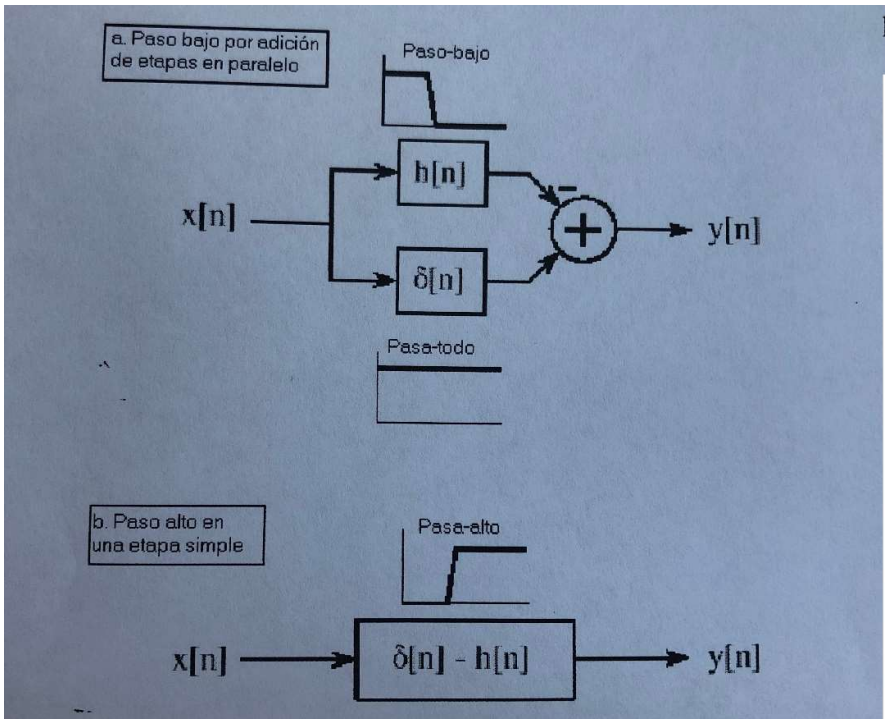


Figura 6. Diagrama de bloques de la inversión espectral. En (a), la señal de entrada, x[n], es aplicado a dos sistemas en paralelo, teniendo una respuesta de impulso de h[n] y D[n]. Tal y corno se muestra en (b), el sistema combinado tiene una respuesta en impulso de D [n] - h[n]. Esto significa que la respuesta en frecuencia del sistema combinado es la inversión de la respuesta en frecuencia de h[n].

Esta Figura muestra porque estas dos modificaciones de paso al dominio del tiempo producen un espectro de la frecuencia invertido.

En (a), la señal de entrada, x[n], es aplicada a dos sistemas en paralelo. Uno de estos sistemas es un filtro paso-bajo, con una respuesta impulso dada por h[n]. El otro sistema no hace nada a la señal, y por ello tiene una respuesta de impulso que es una función delta, o[n].

La salida total, es igual a la salida del sistema pasa-todo *menos* la salida del sistema paso-bajo. Corno los componentes de baja frecuencia son restados de la señal original, solo los componentes de alta frecuencia aparecen en la salida. Así, se realiza un filtro paso-alto.

Esto puede ser ejecutado como una operación de dos pasos en un programa, ejecutando la señal a través de un filtro paso-bajo, y después restar la señal filtrada a la original. Sin embargo, la operación entera se puede ejecutar en la representación de la señal combinando los dos filtros.

Sistemas paralelos con salidas sumadas se pueden combinar en etapas simples añadiendo sus respuestas impulsos. Tal y como se muestra en (b), el núcleo del filtro para el filtro paso-bajo viene dado por: 0[n] - h[n]. Esto es, cambiar el signo de todas las muestras, y después sumar uno a la muestra del centro de simetría.

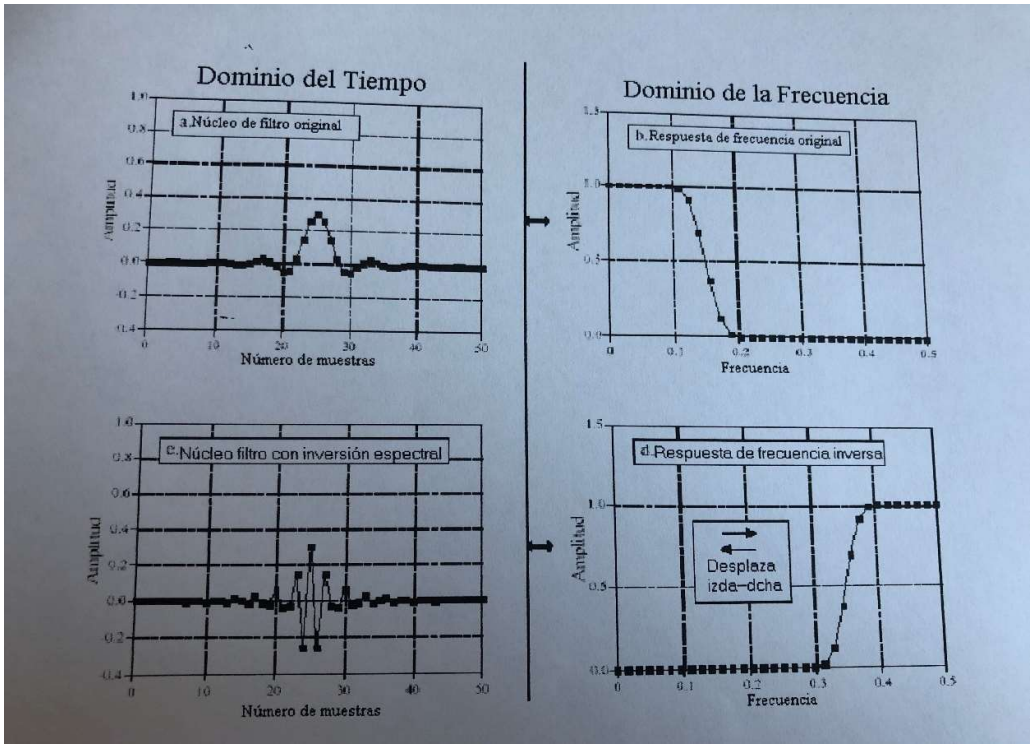


Figura 7. Ejemplo de la respuesta espectral. El núcleo de filtro paso-bajo en (a), tiene la respuesta en frecuencia de (b). El núcleo del filtro paso alto (c), está formado cambiando el signo a todas las muestras de (a). Esta acción en el dominio del tiempo produce un desplazamiento izda-drcha en el dominio de la frecuencia, produciendo Ja respuesta en frecuencia paso alto de (d).

Para poder utilizar esto, los componentes de baja frecuencia contenidos en el filtro paso-bajo deben tener la misma fase que los componentes de baja frecuencia existentes en el sistema paso-todo. Sin embargo no se puede realizar una resta completa. Esto introduce dos restricciones en el método:

(1) el núcleo del filtro original debe tener una simetría izquierda-derecha (p.e., cero o fase lineal), y
(2) impulso debe ser sumado al centro de simetría.

El segundo método para la conversión paso-bajo a paso-alto, reverso espectral, como se muestra en la figura 7. Igual que antes, el núcleo del filtro paso-bajo de (a) le corresponde le corresponde la respuesta en frecuencia de (b). El núcleo del filtro paso-bajo, (c), se forma cambiando el signo de todas las otras muestras en (a). Tal y como se muestra en d, esto invierte el dominio de la frecuencia de izquierda a derecha: 0 cambia a 0.5 y 0.5 cambia a 0.5

La frecuencia de corte del filtro paso-bajo del ejemplo es 0.15, resultando en la frecuencia de corte del filtro paso-alto ser 0.35.

Cambiar el signo de cada muestra es equivalente a multiplicar el núcleo del filtro por una sinusoidal con una frecuencia de 0.5. Esto tiene el efecto de desplazar el dominio de la frecuencia en 0.5. Si se mira en (b) e imagina las frecuencias negativas entre -0.5 y O estas son espejo de las frecuencias entre O y 0.5. Las frecuencias que aparecen en (d) son las frecuencias negativas de (b) desplazadas en 0.5.

Por último, las figuras 8 y 9 muestran como los núcleos de filtros paso-altos pueden ser combinados para formar paso-bandas y rechazo-banda. Además añadiendo los núcleos de los filtros produce un filtro rechazo-banda, mientras realiza la convolución los núcleos de los filtros producen un filtro paso-banda. Estos están basados en camino cascada y sistemas paralelos combinados. Se pueden utilizar múltiples combinaciones de estas técnicas. Por ejemplo, un filtro pasa-banda se pueden diseñar añadiendo los núcleos de dos filtros para realizar un filtro paso-bajo, y después usar la inversión espectral o la reversa espectral tal y como se describe previamente. Todas estas técnicas trabajan muy bien y apenas producen problemas.

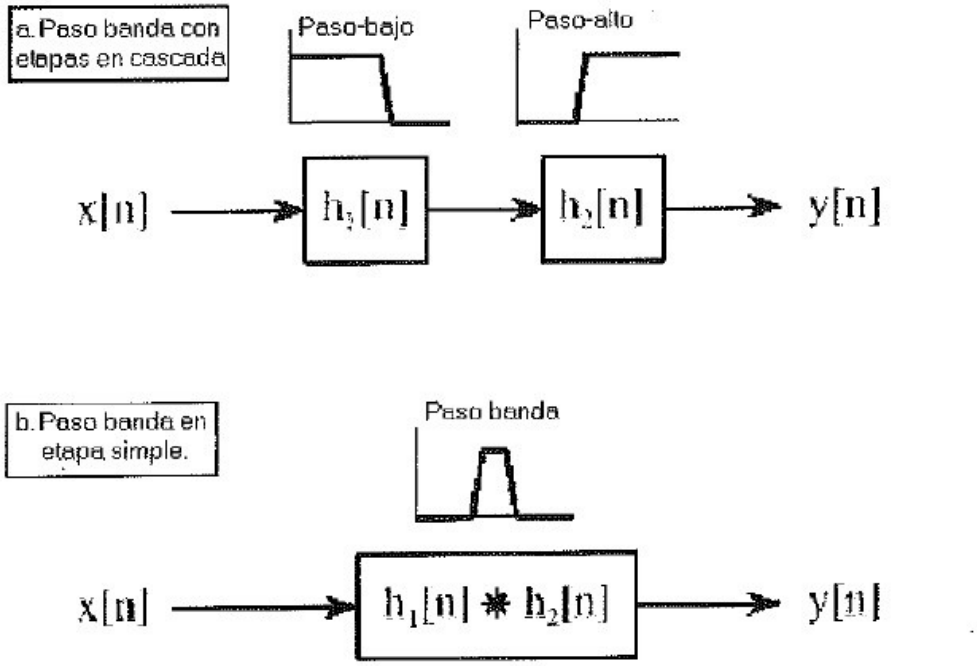


Figura 8. Diseño de un filtro paso banda. Tal y como se muestra en (a), un paso-banda se puede formar mediante un paso bajo y un paso alto en cascada. Esto se puede reducir a una etapa, (b). El núcleo de filtro de una etapa simple es equivalente a la convolución de núcleos de filtro paso bajo y paso alto.

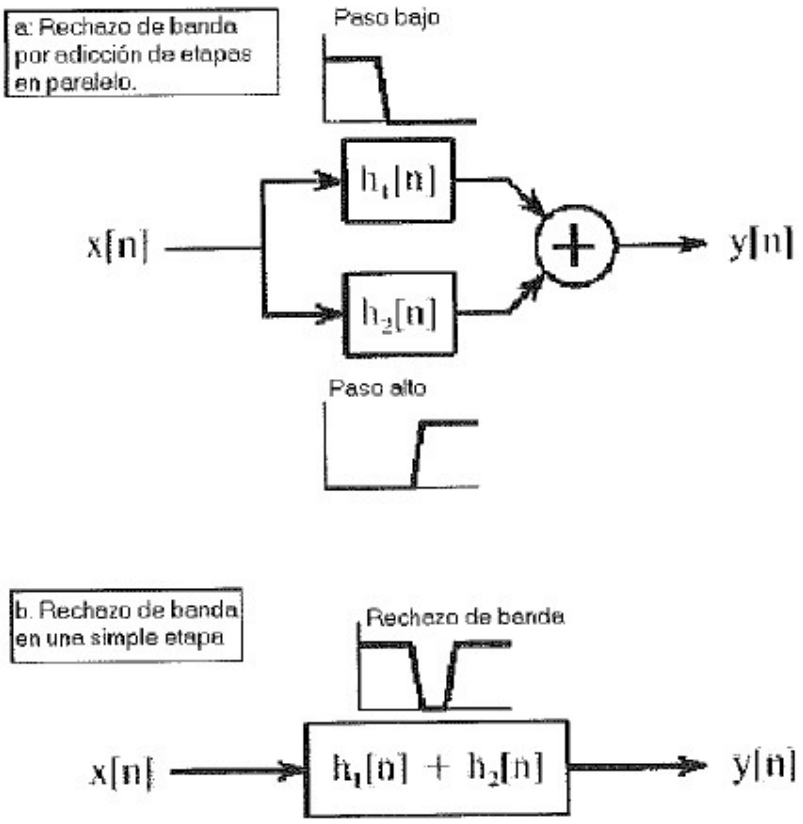


Figura 9, Diseño de un filtro de rechazo banda. Tal y como se muestra en la figura (a), un filtro de rechazo de banda es formado mediante la combinación en paralelo de un filtro paso bajo y un filtro paso alto con sus salidas sumadas. La figura (b) muestra esto reducido a una sola etapa, con un núcleo de filtro realizado mediante la adición de los núcleos de los filtros paso bajo y paso alto.

1.6 Clasificación de filtros.

La Tabla 1 indica cómo están clasificados los filtros digitales según su uso y su implementación. El uso de un filtro digital se puede clasificar en tres categorías: dominio del tiempo, dominio de la frecuencia y rutina. Como previamente se describe, los filtros de dominio en el tiempo son usados cuando la información es codificada en la forma de la onda de la señal.

El filtrado 'del dominio del tiempo se utiliza para acciones como: alisamiento, la eliminación de continua, modelado de la forma de onda, etc. En contraste, los filtros en dominio de la frecuencia son utilizados cuando la información es contenida en la amplitud, frecuencia, y fase de los componentes sinusoidales. El objetivo de estos filtros es separar una banda de frecuencias de otra. Los filtros son usados cuando una acción especial es requerida por el filtro, algo más elaborado que las 4 respuestas básicas (paso-alto, paso-bajo, paso-banda y rechazo-banda). Cuando los filtros son usados para la convolución inversa, es una forma de neutralizar una convolución no necesitada.

		FILTROS IMPLEMENTADOS POR:	
		Convolución Respuesta de Impulso Finita (FIR)	Recursión Respuesta de impulso finita (IIR)
FILTROS USADOS PARA:	Dominio del tiempo (Alisamiento, supresión de continua)	Desplazamiento común	Polo simple
	Dominio de la frecuencia (frecuencias separadas)	Ventanamiento Sincrono	Chebyshev
	Rutina (Deconvolución)	FIR Rutinario	Diseño Iterativo

Tabla 1. Clasificación de filtros. Los filtros se pueden dividir en su uso y en como son implementados.

Los filtros digitales pueden ser implementados de dos formas, mediante la convolución (también llamada respuesta de impulso finito o FIR) y por recursión (también llamada respuesta de impulso infinito o IIR). Los filtros realizados por convolución pueden tener mucho mejor ejecutabilidad que los filtros usados mediante recursión, pero se ejecutan mucho más lentamente.

En sucesivos apartados se describen los filtros digitales según las clasificaciones de la tabla 1. Primero, se verán los filtros realizados por convolución. El desplazamiento promedio es usado en el dominio del tiempo, el ventanamiento asíncrono es utilizado en el dominio de la frecuencia, y el **FIR** rutinario es utilizado cuando se necesita algo especial. Para acabar con los filtros FIR, existe una técnica llamada convolución FFT. Esto es un algoritmo para incrementar la velocidad de convolución, permitiendo a los filtros FIR una ejecución más rápida.

Después, se ven los filtros recursivos. El filtro recursivo de **polo simple** es usado en el dominio del tiempo, mientras el filtro **Chebyshev** es usado en el dominio del tiempo. Los filtros recursivos teniendo una respuesta rutinaria son diseñados por las **técnicas iterativas**.

Tal y como se muestra en la tabla 1, la **convolución** y la **recursión** son técnicas contrarias; se debe usar uno u otro para una aplicación particular. Más adelante se muestra una comparación de ambos, en dominio del tiempo y dominio de la frecuencia.

2. Filtros de desplazamiento común.

Este es el tipo de filtro más común en DSP, principalmente porque es el filtro digital más fácil para entender y usar. A pesar de su simplicidad, este tipo de filtro es óptimo para una tarea común: reducir el ruido aleatorio mientras retiene una respuesta de filo de paso. Esto lo hace el mejor filtro para señales codificadas en el dominio del tiempo. Sin embargo, es el peor filtro en el dominio de la frecuencia de señales codificadas, con poca facilidad de separar una banda de frecuencia de otra. Familiares de los filtros de desplazamiento común incluyendo el Gaussiano, Blackman, y múltiples-pasos de desplazamiento común. Estos tienen ligeramente mejor ejecución en el dominio de la frecuencia, a expensas del incremento del tiempo de procesamiento de datos.

2.1 Implementación por convolución.

Como su nombre implica, este filtro opera desplazando un número de puntos de la señal de entrada para producir cada punto en la señal de salida. En forma de ecuación, se escribe del siguiente modo:

$$y[i] = \frac{1}{w} \sum_{j=0}^{w-1} x[i+j]$$

Ecuación !.Donde *x* [ ) es la señal de entrada, *y* [ ] es la señal de salida, y M es el número de puntos del desplazamiento. Esta ecuación solo usa puntos en un lado de la muestra de salida que ha sido calculada. Por ejemplo, en un filtro de desplazamiento común de 5 puntos, el punto 80 de la señal de salida del viene dado por:

$$y[80] = \frac{x[80] + x[81] + x[82] + x[83] + x[84]}{5}$$

Como una alternativa, el grupo de puntos de la señal de entrada se pueden elegir simétricamente alrededor del punto de salida:

$$y[80] = \frac{x[78] + x[79] + x[80] + x[81] + x[82]}{5}$$

Esto corresponde a cambiar en la ecuación 1: j = O de M - 1 de: j = – (M - 1) / 2 a (M & 1) / 2. Por ejemplo, en un filtro de desplazamiento común de 11 puntos, el índice, j, puede ir desde O a 11 (desplazamiento de un lado) ó de -5 a 5 (desplazamiento simétrico). El desplazamiento simétrico requiere que M sea un número singular. Programando es ligeramente más sencillo con los puntos a un solo lado; sin embargo, esto produce un desplazamiento relativo entre las señales de entrada y salida.

Se puede reconocer que el filtro de desplazamiento común es una convolución usando un núcleo de filtro muy simple. Por ejemplo, un filtro de 5 puntos tiene el núcleo de filtro: 0, 0,1/5,1/5, 1/5, 1/5, 1/5, 1/5, 0, 0...
Esto es, que este filtro es un convolución de la señal de entrada con un pulso rectangular teniendo un área de uno.

2.2 Reducción del ruido frente a respuesta de paso.

Debido a su simplicidad, el filtro de desplazamiento común es a menudo lo primero que se intenta cuando nos enfrentamos a un problema. Este tipo de filtro no sólo es muy bueno para muchas aplicaciones, es óptimo para un problema común, reduciendo el ruido blanco mientras se conserva la respuesta más bien definida de paso.

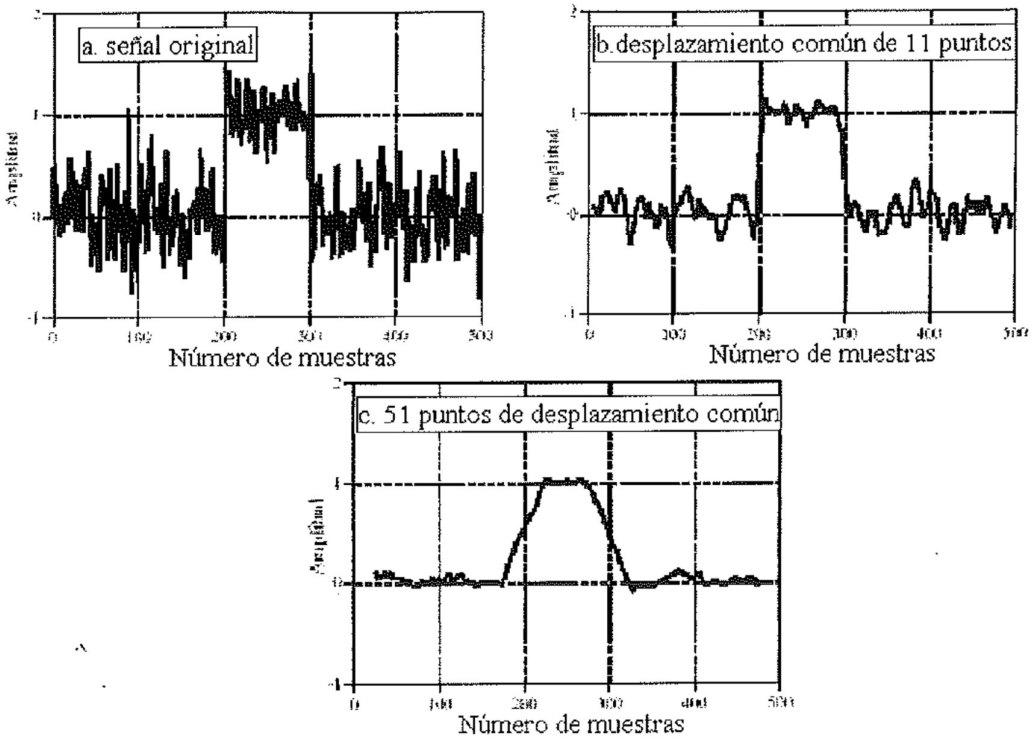


Figura 1. Ejemplo de filtro de desplazamiento común. En (a), un pulso rectangular se aplica en el ruido aleatorio, en (b) y (c), esta señal se filtra con filtros de desplazamiento común de 11 y 51 puntos, respectivamente. Como el número de puntos en el filtro se incrementa, el ruido llega a ser menor; sin embargo, los filos llegan a ser menos afilado. Estos filtros son la solución óptima para este problema, ofreciendo el menor ruido posible para un filo dado. La cantidad de reducción de ruido es igual a la raíz cuadrada del número de puntos en el desplazamiento. Por ejemplo, un filtro de desplazamiento común de 100 puntos reduce el ruido en un factor de 10.

Para entender porque el desplazamiento común es la mejor solución, hay que imaginar que se quiere diseñar un filtro con un filo fijo del borde. Por ejemplo, con once puntos en la subida de la respuesta de paso. Esto requiere que el núcleo del filtro tenga once puntos. Lo que hay que saber es cuales son los valores del núcleo del filtro que hay que tomar para ·minimizar el ruido en la señal de salida. Como el ruido que se quiere minimizar es aleatorio, ninguno de los puntos de entrada es especial; Cada uno es tan ruidoso como su contiguo. Por ello, es inservible dar tratamiento preferencial para cualquier punto de entrada asignándole un coeficiente mayor en el núcleo del filtro. El ruido mínimo es obtenido cuando todas las muestras de entrada son tratadas por igual, esto es, el filtro de desplazamiento común. (Más tarde se muestra que otros filtros son esencialmente tan buenos. Lo que pasa es que, ningún filtro es mejor que él).

2.3 Respuesta en frecuencia.

En la siguiente figura se muestra la respuesta en frecuencia del filtro de desplazamiento común. Está matemáticamente descrito por la transformada de Fourier del pulso rectangular.

$$H[f] = \frac{\sin(\pi f M)}{M \sin(\pi f)}$$

Ecuación 2. La respuesta en frecuencia de un filtro de desplazamiento común de M puntos. La frecuencia, f, recorre desde 0 a 0.5. Desde $f = 0$, usando: $H[f] = 1$.

El rebose es muy lento y la atenuación de la banda de paso es muy mala. Claramente, el filtro de desplazamiento común no puede separar una banda de frecuencias de otra. Hay que tener en cuenta que un buen funcionamiento en el dominio del tiempo implica uno malo en el dominio de la frecuencia, y viceversa. En resumen, el desplazamiento común es un excepcionalmente bueno **filtro de alisamiento** (acción en el dominio del tiempo), pero un excepcionalmente malo **filtro paso-bajo** (acción en el dominio de la frecuencia).

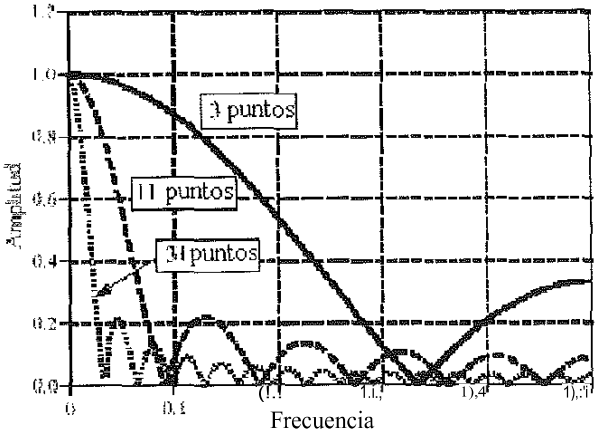


Figura 2. Respuesta en frecuencia del filtro de desplazamiento común. El desplazamiento común es un filtro paso-bajo muy pobre, además de su bajo rebose y pobre atenuación del paso de banda. Estas gráficas son generadas mediante la ecuación 2.

2.4 Familia de los filtros de desplazamiento común.

En comportamiento ideal, los diseñadores de filtros sólo tendrían que elegir entre el dominio del tiempo ó el dominio de la frecuencia para codificar la información, pero nunca una mezcla de los dos en la misma señal. Pero existen algunas aplicaciones en donde ambos dominios son simultáneamente importantes. Por ejemplo, las señales de televisión. La información de video está codificada en el dominio de tiempo, esto es, la forma de la onda corresponde con la de los patrones de brillo en la imagen. Sin embargo, mientras dura la transmisión de la señal del video es tratada según su composición en frecuencia, así como a su ancho de banda total, cómo están las ondas portadoras de sonido y color añadidos, la eliminación y la restauración del componente de CD, etc. Otro ejemplo, la interferencia electromagnética es más conveniente sobreentendida en el dominio de frecuencia, incluso sí la información de la señal está codificada en el dominio de tiempo.

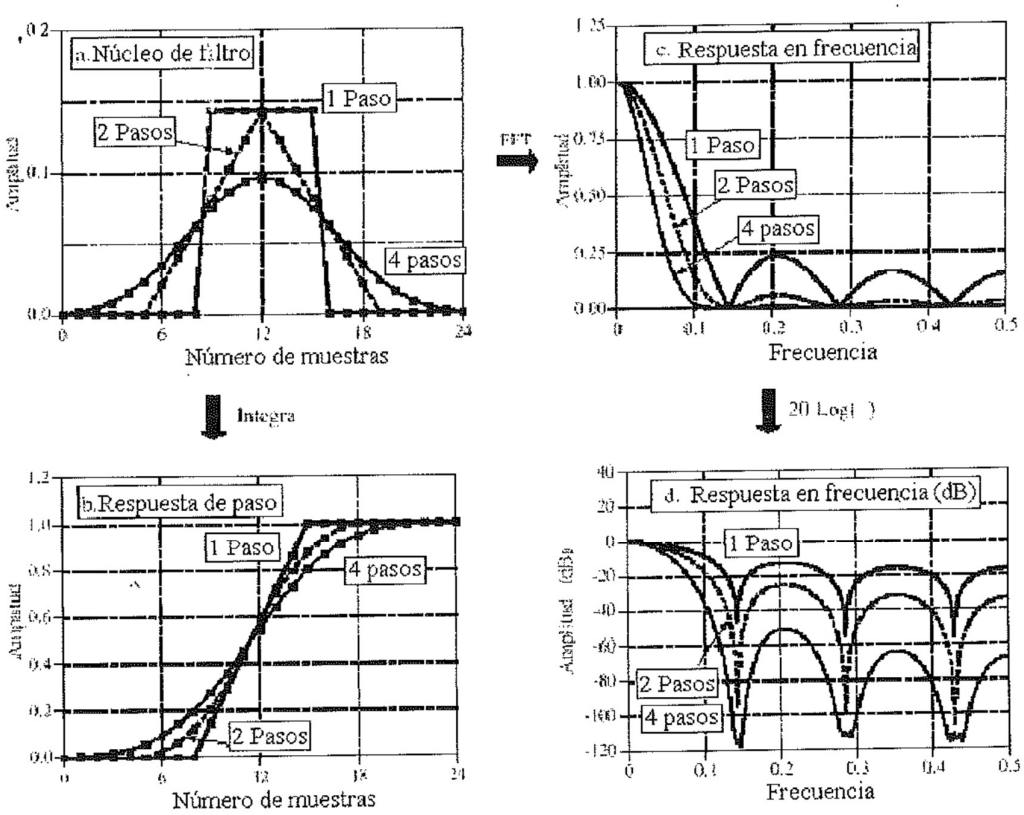


Figura 3. Características de los filtros de desplazamiento común de múltiples pasos. La figura (a) muestra los núcleos de filtros resultantes de pasar un filtro de desplazamiento común de 7 pasos sobre los datos una, dos, y cuatro veces. La figura (b) muestra la respuesta de paso correspondiente, mientras (c) y (d) muestra la respuesta en frecuencia correspondiente.

En esta figura se ven las características de filtros de desplazamiento común de múltiples pasos. En (a) se muestra el núcleo del filtro resultante de pasar a un filtro de desplazamiento común los datos una, dos y cuatro veces. En (b) la correspondiente respuesta de paso, (c) y (d) muestran las correspondientes respuestas en frecuencia.

Por ejemplo, el monitor de temperatura en un experimento científico podría estar contaminado por ondas de 50 Hz correspondientes a la conducción eléctrica, de 30 kHz de una fuente de alimentación conmutada, ó 1320 kHz de una estación de radio local AM. La familia del filtro de desplazamiento tiene mejores ejecuciones de dominio de frecuencia, y pueden ser útiles en estas aplicaciones mixtas de dominio.

Los filtros de desplazamiento común de múltiples pasos involucran a pasar la señal a través de un filtro de desplazamiento común dos o más veces. Las anteriores figuras muestran el núcleo del filtro global resultante de un, dos y cuatro pasos. Dos pasos son equivalentes a usar un filtro **triangular** (un filtro rectangular convolucionado con sí mismo). Después de cuatro o más pasos, el filtro equivalente es como el Gaussiano (recordando el teorema del límite central). Como se muestra en (b), el múltiple paso produce una "s" con forma de respuesta de paso, como se comparara con la línea recta del paso simple.

Las respuestas de frecuencia en (c) y (d) son dadas por la ecuación 2 **multiplicado por sí mismo** para cada paso. Esto es, cada convolución en el dominio del tiempo produce una multiplicación en el espectro de la frecuencia.

La siguiente figura muestra la respuesta en frecuencia de dos familiares del filtro de desplazamiento común. Cuando un **Gaussiano** puro es utilizado como un núcleo de filtro, la respuesta en frecuencia es también Gaussiano. El Gaussiano es importante porque es la respuesta de impulso de muchos sistemas naturales y sintéticos. Por ejemplo, un breve pulso de luz entrando en una larga línea de transmisión de fibra óptica regresará como un pulso Gaussiano, debido a los caminos diferentes tomados por los protones dentro de la fibra. El núcleo del filtro Gaussiano es también usada extensamente en el **procesado de imagen** porque tiene propiedades únicas que permiten rápidas convoluciones bidimensionales. La segunda respuesta en frecuencia corresponde a usar una **ventana Blackman** como un núcleo de filtro. La forma exacta de la ventana Blackman se ve mucho como un Gaussiano.

Estos familiares de los filtros de desplazamiento de común mejoran el filtro de desplazamiento común de tres formas: Primero, y más importante, estos filtros tienen mejor **atenuación de la banda de rechazo**. En segundo lugar, los núcleos del filtro se reducen a una amplitud más pequeña cerca de los extremos. Hay que recordar que cada punto en la señal de salida es una suma ponderada de un grupo de muestras de la entrada.

Si el núcleo del filtro se reduce, entonces las muestras en la señal de entrada que están más alejadas reciben menos peso que los que están a corta distancia. Tercero, las respuestas de paso son alisadas curvas, en vez de la línea recta escarpada del anterior caso. Estos dos últimos son usualmente de beneficios limitados, aunque se pueden encontrar aplicaciones donde son genuinas ventajas.

El filtro de desplazamiento común y sus familiares tienen todo el mismo ruido aleatorio mientras mantiene una respuesta bien definida de paso. La ambigüedad radica en cómo el **tiempo de subida** de la respuesta es medida. Si el tiempo de subida está medido del O % al 100 % del paso, entonces el filtro de desplazamiento común es el mejor que usted puede hacer, como previamente se muestra. En comparación, midiendo el tiempo de subida del 10 % al 90 % hacen la ventana Blackman mejor que el filtro de desplazamiento común. La idea es, considerar estos filtros iguales en este parámetro.

La diferencia más grande en estos filtros es la **velocidad de ejecución.** Usando un algoritmo recursivo (descrito posteriormente), el filtro de desplazamiento común se ejecutará en un instante. De hecho, es el filtro **digital más rápido** disponible. Múltiples pasos del desplazamiento común serán correspondientemente más lento, pero muy rápido. En comparación, los filtros Gaussiano y Blackman son mucho más lentos, porque deben usar la convolución. Pensando un factor de 10 veces el número de puntos del núcleo del filtro (basado en la multiplicación siendo aproximadamente 1O veces más lento que la suma). Por ejemplo, se espera que un Gaussiano de 100 puntos va a ser 1000 veces más lento que una recursión que usa desplazamiento común.

2.5 Implementación recursiva.

Una gran ventaja del filtro de desplazamiento común en movimiento es que puede ser implementado con un algoritmo muy rápidamente. Para entender este algoritmo, hay que imaginar que pasa una señal de entrada x[ ], a través de un filtro de desplazamiento común de siete puntos para formar una señal de salida, y[ ].

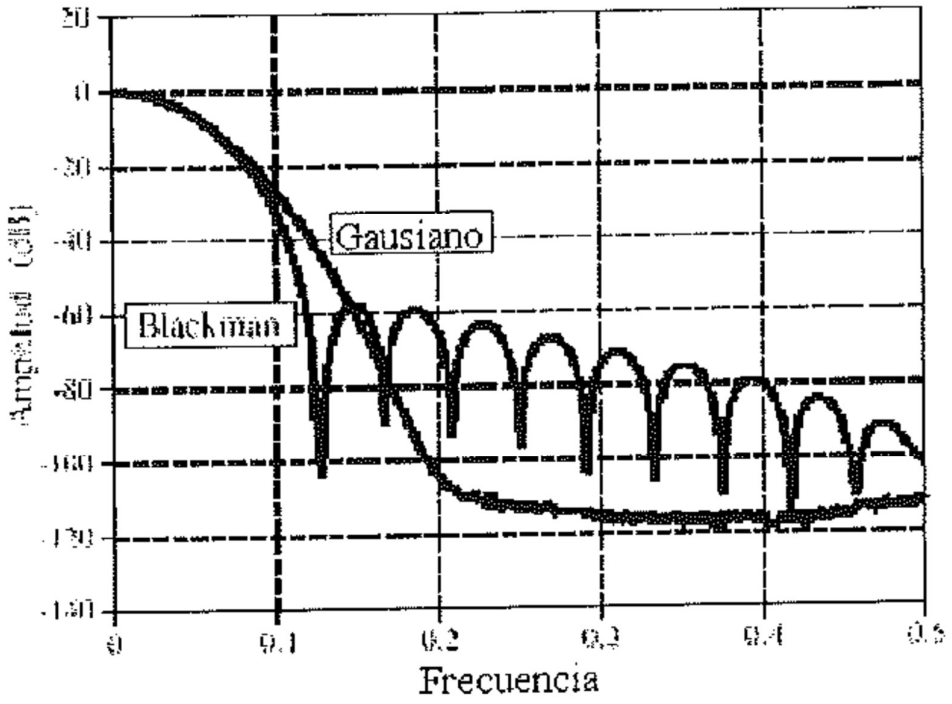


Figura 4. La respuesta en frecuencia de la ventana Blackman y los núcleos de filtros Gaussianos. Ambos filtros proveen una mejor atenuación de la banda de rechazo quel filtro de desplazamiento común. Esto no es una ventaja en la supresión del ruido aleatorio de las señales codificadas en el dominio del tiempo, pero puede ser útil en problemas de dominio común. La desventaja de estos filtros es que deben usar la convolución, que es un algoritmo muy lento.

Vemos como se forman dos puntos de salida adyacente y [50] e y [51]:

$$y\,[50] = X\,[47] + x[48] + X\,[49] + x[50] + X\,[51] + x[52] + X\,[53]$$

$$y\,[51] = x\text{--}[48] + x[49] + x\,[50] + x[51] + x\,[52] + x[53] + x\,[54]$$

Se observa que los cálculos son muy parecidos, por lo que se puede escribir:

$$y(51] = y(50] + X\,(54) - X\,(47)$$

Después de que el primer punto es calculado y [ ], todos los otros puntos se pueden encontrar realizando una única suma y resta por punto.

Esto puede ser expresado mediante la ecuación:

$$y(i] = y(\acute{I} - I] + X\ (i+p] - X\ (i - q]$$

donde: $p = (M - 1)/2$ y $q = p + 1$

Ecuación 3. Implementación recursiva de los filtros de desplazamiento común. En esta ecuación, x[ ] es la señal de entrada, y[ ] es la señal de salida, M es el número de puntos en el desplazamiento común. Antes de que esta ecuación pueda ser usada, el primer punto de la señal puede ser calculada usando la suma estándar.

Se ve que en esta ecuación usa dos fuentes de datos para calcular cada punto en la salida: los puntos de la entrada y los puntos previamente calculados de la salida. Ésta se llama una ecuación **recursiva,** significando que el resultado de un cálculo es usado en cálculos futuros. El término "recursivo" también tiene otros significados, especialmente en la ciencia de la computación. Más adelante se discute una variedad de filtros recursivos en más detalle. Hay que ser consciente que los filtros recursivos de desplazamiento común es muy diferente de otros filtros recursivos típicos. En particular, la mayoría de filtros recursivos tienen una respuesta de impulso (IIR) infinitamente larga, compuestas de sinusoidales y exponenciales. La respuesta de impulso de desplazamiento común promedio en movimiento es un pulso rectangular

(Respuesta finita de impulso, ó FIR).

Este algoritmo es más rápido que otros filtros digitales por varias razones. Primero, que hay sólo dos computaciones por punto, a pesar del largo del núcleo del filtro. En segundo lugar, la adición y la sustracción son las únicas operaciones de matemáticas necesitadas, mientras la mayoría de filtros digitales requieren multiplicación consumidora de tiempo. Tercera que el esquema de indexación es muy simple. Cada índice en la ecuación 3 es encontrado añadiendo o sustrayendo un número entero de constantes que pueden calcularse antes de que el filtrado comience (como, p y q). Cuarto, el algoritmo entero puede ser efectuado con representación de números enteros. Dependiendo del hardware usado, los enteros pueden tener mayor orden y magnitud más rápida que en punto flotante.

Sorprendentemente, la representación de entero trabaja mejor que el punto flotante en este algoritmo, además de ser más rápido. El error de redondeo de la aritmética del punto flotante puede producir resultados inesperados si no se tiene cuidado. Por ejemplo, una señal de 10.000 muestras siendo filtrado con este método, la última muestra de la señal filtrada contiene el error acumulado de 10.000 sumas y 10,000 restas. Esto aparece en la señal de salida como una deriva del offset. Los enteros no tienen este problema ya que no existe un error de redondeo en la aritmética.

3. Filtros de ventanamiento síncrono.

Los filtros de ventanamiento síncrono se usan para separar una banda de frecuencias de otra. Son muy estables, producen pocas sorpresas, y pueden ser llevados para niveles de ejecución increíbles. Estas características del dominio de la frecuencia excepcional son obtenidas a expensas de un desempeño pobre en el dominio de tiempo, incluyendo el excesivo rizado y rebose de la respuesta de paso. Cuando ejecutados por convolución estándar, los filtros de ventanamiento síncrono son fáciles para programar, pero lentos de ejecutar. Más adelante se muestra como la FFT puede ser usada para mejorar la velocidad de computación de estos filtros.

3.2 Estrategia del Ventanamiento síncrono

La figura 1 ilustra la idea del filtro de ventanamiento síncrono. En (a), se muestra la respuesta de frecuencia del filtro de paso bajo **ideal**. Todas las frecuencias por debajo de la frecuencia de corte, $\mathbf{f}_c$ pasan con amplitud unidad, mientras que todas las frecuencias superiores son bloqueadas. La banda de paso es perfectamente plana, la atenuación en la banda de rechazo es infinita, y la transición entre los dos es infinitesimalmente pequeña.

Tomar la Transformada inversa de Fourier de esta respuesta en frecuencia ideal produce el núcleo del filtro (respuesta impulso) mostrada en (b).Esta curva está en la forma general: sen (x) /x llamada **función seno**, viene dada por:

$$h[i] = \frac{\sin(2\pi f_c i)}{i\pi}$$

Convolucionar una señal de entrada con este núcleo de filtro provee un perfecto filtro paso bajo. El problema es, la función síncrona continúa para ambos infinitos negativo y positivo sin descender por debajo del cero de amplitud. Aun cuando este largo infinito no es un problema para los **matemáticos**, es un cuello de botella para las computadoras.

Para solucionar este problema, se hacen dos modificaciones a la función síncrona en (b), resultando la forma de onda mostrada en (c).

Primero, que están truncado para **M + 1** puntos, simétricamente seleccionado alrededor del lóbulo principal, donde **M** es un número dado. Toda muestra fuera de estos **M + 1** puntos son puestos a cero, o simplemente ignorados.

En segundo lugar, la secuencia de entrada es desplazada a la derecha a fin de que vaya de O para M. Esto permite al núcleo del filtro ser representado usando sólo índices positivos. Mientras muchos lenguajes de programación permiten índices negativos, los cuales son molestos de usar. El efecto exclusivo de este desplazamiento M/2 en el núcleo del filtro es para desplazar la señal de salida por la misma magnitud.

Como el núcleo modificado del filtro es sólo una aproximación para el núcleo del filtro ideal, no tendrá una respuesta de frecuencia ideal. Para encontrar la respuesta de frecuencia que es obtenida, la transformada de Fourier puede ser tomada de la señal en (c), resultando la gráfica en (d). El excesivo rizado en la banda de paso y la pobre atenuación en la banda de rechazo. Estos problemas resultan de la discontinuidad abrupta en los extremos de la función síncrona truncada. Aumentar la longitud del núcleo del filtro no reduce estos problemas; La discontinuidad es significante sin tener en cuenta como de largo esta **M** hecha.

Afortunadamente, hay un método simple de mejorar esta situación. La figura (e) muestra que una curva suavemente adelgazada llamada ventana

Blackman. Multiplicando la síncrona truncada, (c), por la ventana Blackman, (e), resulta el núcleo del filtro de **ventanamiento síncrono** mostrado en (f). La idea es reducir la brusquedad de los extremos truncados y por consiguiente mejorar la respuesta de frecuencia. La figura (g) demuestra esto. El paso banda es ahora plano, y la atenuación de la banda de rechazo es tan buena que no puede verse en esta gráfica.

Varias ventanas diferentes están disponibles, la mayoría de ellas dadas el nombre por sus desarrolladores originales en los 1950s. Sólo dos merecen la pena, la **ventana Hamming** y la **ventana Blackman**, dadas por:

$$w[i] = 0.54 - 0.46 \cos(2 \pi i / M)$$

Ecuación 1. La ventana Hamming. Esta ventana recorre desde i = O a M, para un total de M + 1 puntos.

$$w[i] = 0.42 - 0.5 \cos(2 \pi i / 1M) + 0.08 \cos(4 \pi i / M)$$

Ecuación 2.La ventana Blackman.

La figura 16-2a muestra la forma de estas dos ventanas para **M = 50** (51 puntos totales en las gráficas). ¿Cuál de estas dos ventanas se debe usar? Es un intercambio entre parámetros. Tal y como se muestra en Fig. I6-2b, la ventana Hamming tiene acerca de un rebose del 20 % más rápido que el Blackman.

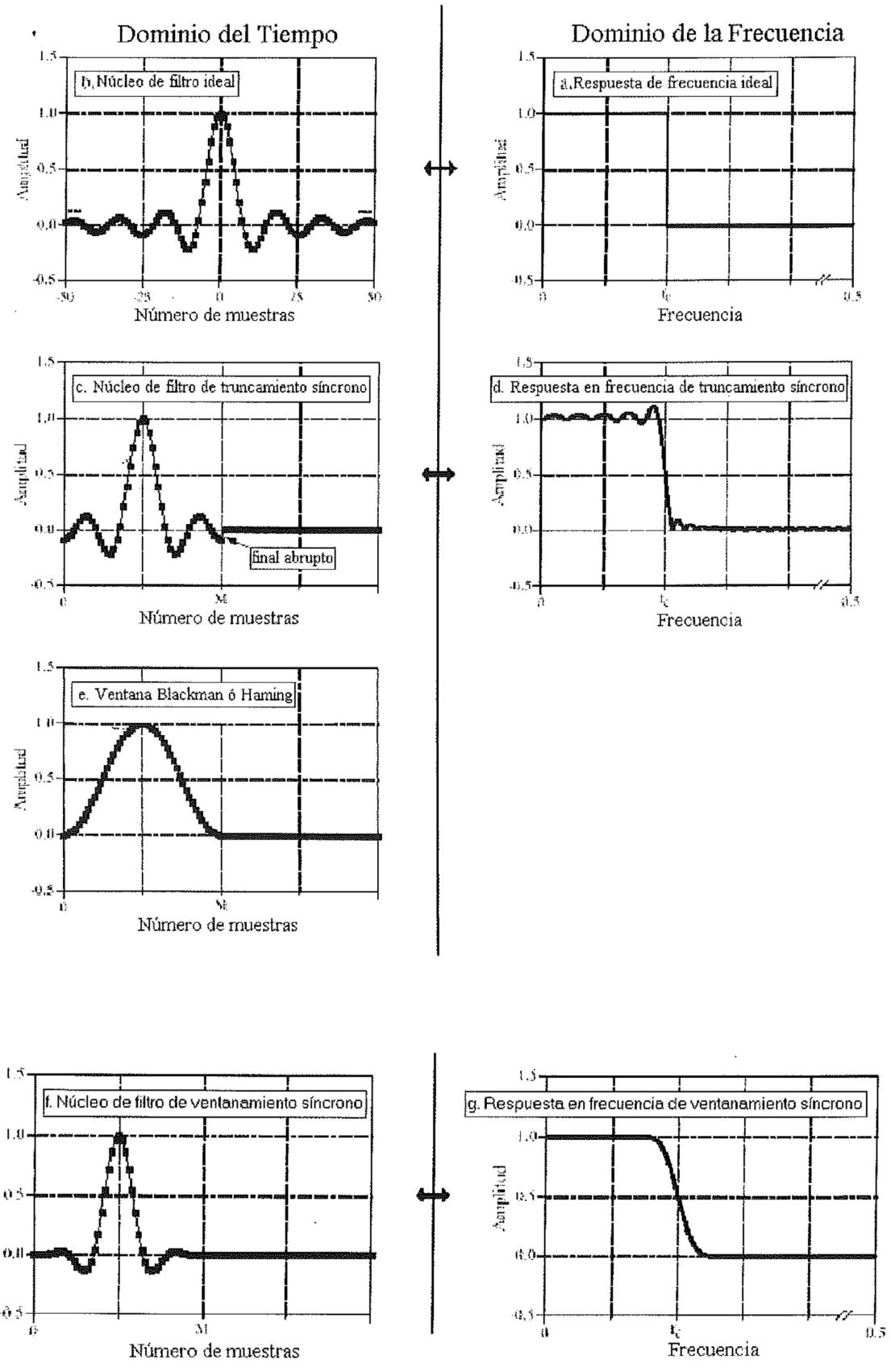

Dominio del Tiempo
Dominio de la Frecuencia
b, Núcleo de filtro ideal
a, Respuesta de frecuencia ideal
c. Núcleo de filtro de truncamiento síncrono
d. Respuesta en frecuencia de truncamiento síncrono
final abrupto
e. Ventana Blackman ó Haming
f. Núcleo de filtro de ventanamiento síncrono
g. Respuesta en frecuencia de ventanamiento síncrono
Amplitud
Número de muestras
Frecuencia

Figura 1. Derivada del núcleo de filtro de ventanamiento síncrono. La respuesta en frecuencia de un filtro paso bajo ideal es mostrada en (a), con el correspondiente núcleo de filtro en (b), una función síncrona. Como el síncrono es infinitamente largo, debe ser truncado para ser usado en un ordenador, tal y como se muestra en (c). Sin embargo, esta truncación produce en cambios indeseables en la respuesta en frecuencia, (d). La solución es multiplicar el truncamiento-síncrono con una ventana lisa, (e), resultante en el núcleo de filtro de ventanamiento síncrono, (f). La respuesta en frecuencia del ventanamiento síncrono, (g), es alisada y tiene buen comportamiento. Estas figuras no están a escala.

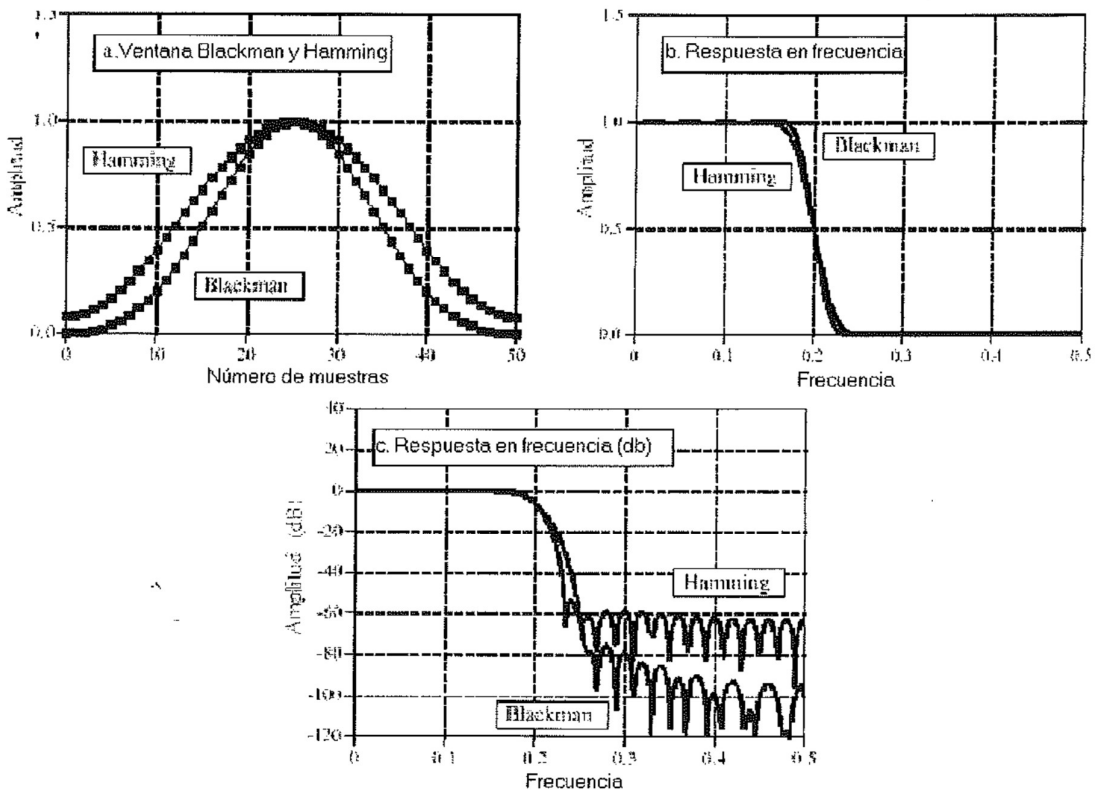


Figura 2. Características de las ventanas Blackman y Hamming. Las formas de estas dos ventanas se muestran en (a), y están dadas mediante las ecuaciones 1 y 2. Tal y como se muestra en (b), la ventana Hamming produce aproximadamente un 20% de rebose más rápido que la ventana Blackman. Sin embargo, la ventana Blackman tiene mejor atenuación en el banda de rechazo (Blackman: 0.02%, Hamming: 0.2%), y un rizado de la banda de paso más baja (Blackman: 0.02% Hamming: 0.2%).

Sin embargo (c) muestra que el Blackman tiene una mejor **atenuación de la banda de rechazo**. Para ser exacto, la atenuación de la banda de rechazo para el Blackman es - 74dB (-+ 0.02 %), mientras para el Hamming es sólo - 53dB (-+ 0.2 %). Aunque no puede verse en estas gráficas, el Blackman tiene un rizado de la banda de paso de sólo aproximadamente el 0.02 %, mientras que para el Hamming es típicamente el 0.2.

En general, el Blackman debería ser la primera elección; Un rebose lento es más fácil para manejar que una atenuación pobre de la banda de rechazo.

Existen otras ventanas que se deberían conocer, aunque se quedan cortas comparadas con el Blackman y el Hamming. La **ventana Bartlett** es un triángulo, usando líneas rectas. La **ventana Hamming**, también llamada la **ventana de coseno levantada**, viene dada por: **w[i] = 0.5 - 0.5cos(2rci /M**). Estas dos ventanas tienen aproximadamente la misma velocidad de rebose que el Hamming, pero peor atenuación de la banda de rechazo: (Bartlett: -25dB ó 5.6%, Hamming -44dB ó 0.63%). La **ventana rectangular** no es exactamente una ventana, solo un truncamiento de la cola (tal y corno la figura 16-lc).

Mientras él rebose es +-2.5 veces más rápido que el Blackrnan, la atenuación de la banda de paso es solo – 21 dB (8.9 %).

3.2 Diseñando el Filtro.

Para diseñar un ventanamiento síncrono, se debe seleccionar dos parámetros: La frecuencia de corte, fe, y el tamaño del núcleo del filtro, M.

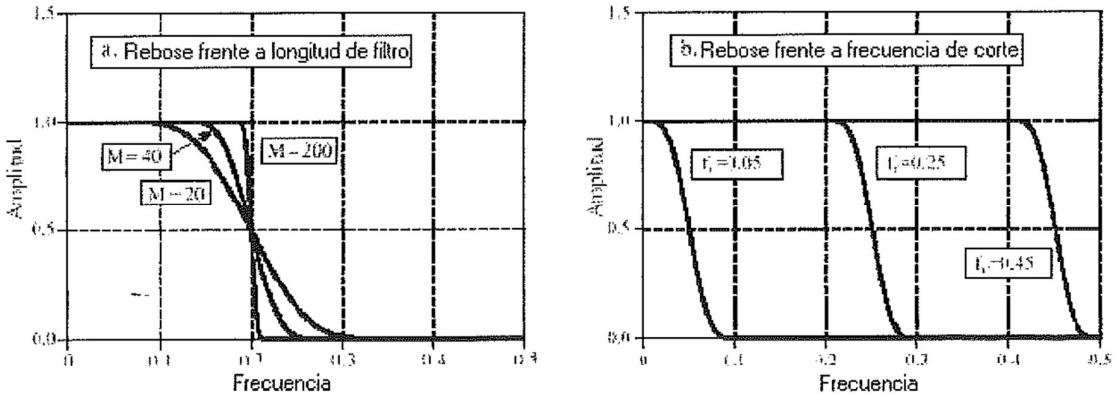


Figura 3. Longitud de filtro frente a rebose del filtro de ventanamiento síncrono. Tal y corno se muestra en (a), para M = 20, 40 y 200, los anchos de banda de transición son BW = 0.2, 0.1, y 0.02 del ratio de muestreo, respectivamente. Tal y como se muestra en (b), la forma de la respuesta en frecuencia no cambia con diferentes frecuencias de corte. En (b), M =60.

La frecuencia de corte es expresada como una fracción del ratio de muestreo, y por ello debe estar entre 0 y 0.5. El valor de **M** determina él rebose según la aproximación:

$$M = \frac{4}{BW}$$

Ecuación 3. Longitud del filtro frente a rebose. La longitud del núcleo del filtro, M, determina el ancho de banda de transición del filtro, BW. Esto es solo una aproximación, ya que el rebose depende de la ventana particular que se use.

Relación entre rebose y longitud del filtro. La longitud del núcleo del filtro, M, determina el ancho de banda de transición del filtro, BW. Esto es únicamente una aproximación ya que él rebose depende solo de la ventana usada.

Donde **BW** es la anchura de la banda de transición, medida donde la curva justamente .,apenas deja a uno, para donde ella casi cae a cero (esto es, 99 % a 1 % de la gráfica). El ancho de banda de transición está también expresado como una fracción de la frecuencia de muestreo, y debe entre O y 0.5. La figura 3a demuestra un ejemplo de cómo es usada esta aproximación. Las tres curvas mostradas son generadas de núcleos de filtro con: **M** = 20, 40, y 200. De la ecuación 3, los anchos de banda de transición son: **BW** = 0.2, 0.1, y 0.02 respectivamente. La figura (b) muestra que la forma de la respuesta en frecuencia no depende en la frecuencia de corte seleccionada.

Como el tiempo requerido para la convolución es proporcional a la longitud de las señales, la ecuación 3 expresa un intercambio entre el **tiempo de computación** (depende del valor de *BW)* y la (depende del valor de M) y del **filtro**. Por ejemplo, el 20 % más lento del rebose de la ventana Blackman (en comparación con el Hamming) pueden ser compensado usando una núcleo de filtro un 20 % más largo. En otras palabras, se podría decir que la ventana Blackman es un 20 % más lenta para ejecutar un rebose equivalente de la ventana Hamming. Esto es importante porque la velocidad de ejecución de los filtros de ventanamiento síncrono es muy lenta.

Como también se ve en la figura 3b, la frecuencia de corte del filtro de ventanamiento síncrono es medida en el punto de amplitud 1/2. El uso de 0.5 en lugar del estándar 0.707 (- 3dB) usado en la electrónica analógica y otros filtros digitales, es porque la respuesta de frecuencia del ventanamiento síncrono es simétrica entre la banda de paso y la banda de rechazo. Por ejemplo, la ventana Hamming produce una rizado de la banda de paso del 0.2% y una y una atenuación idéntica de la banda de rechazo del 0.2. Otros filtros no demuestran esta simetría, y por consiguiente no tienen ventaja en usar el punto de amplitud de ½ para marcar la frecuencia de corte. Tal y como se demuestra posteriormente, esta simetría hace el ventanamiento síncrono ideal para la inversión espectral.

Después de que **fc** y **M** hayan sido seleccionadas, el núcleo del filtro es calculado en la siguiente relación:

$$h[i] = K \frac{\sin(2\pi f_c (i - M/2))}{i - M/2} \left[0.42 - 0.5\cos\left(\frac{2\pi i}{M}\right) + 0.08\cos\left(\frac{4\pi i}{M}\right) \right]$$

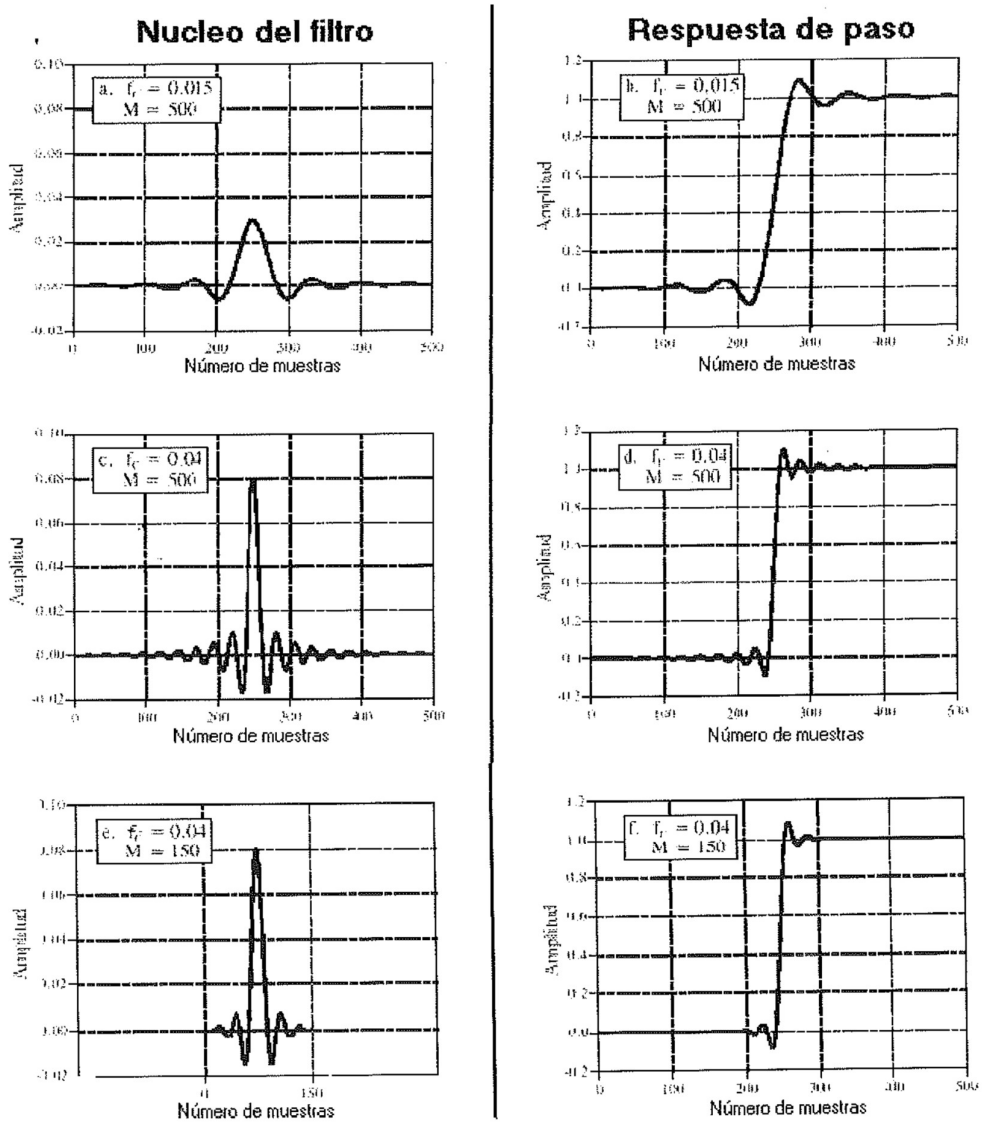
Ecuación 4. Núcleo de filtro de ventanamiento síncrono. La frecuencia de corte, f c, se4 muestra como una fracción del muestreo entre 0 y 0.5. La longitud del núcleo del filtro es determinada por M, el cual debe ser un número entero. El número de muestra i, es un entero que va de 0 a M, resultando M + 1 puntos totales en el núcleo del filtro. La constante K es seleccionada para proveer una ganancia 1 a frecuencia 0. Para evitar un error de división por 0, para i = M/2, usando h[i] = 2 D *f e* K.

Se deben identificar tres componentes: La **función síncrona**, el desplazamiento **M/2**, y la **ventana Blackman**. Para que el filtro tenga ganancia unidad en DC, la constante K debe estar seleccionada de forma que la suma de todas las muestras sea igual a uno.

En la práctica, se ignora K durante el cálculo del núcleo del filtro, y entonces se **normalizan** todas las muestras que se necesitan. Hay que darse cuenta de cómo el cálculo es manipulado en el centro de sincronía, **i = M/20**, lo cual implica una división por cero. Esta ecuación puede ser lenta, pero es fácil de usar.

Sea un ejemplo de la ecuación 4, sea **M** seleccionada como 100, (**M** debe ser un número par). El primer punto del núcleo del filtro está en la posición de orden O, mientras que el último punto está en posición de orden 100. Esto quiere decir que la señal entera tiene 101 puntos de longitud. El centro de simetría está en punto 50, esto es M/2. Los 50 puntos a la izquierda de punto 50 son simétricos con los 50 puntos a la derecha.

El punto O tiene el mismo valor que el punto 100, y el punto 49 que el punto 51.



La figura 4 muestra ejemplos de núcleos de filtros de ventanamiento síncrono, y sus respuestas de paso correspondientes. La frecuencia de la oscilación senoidal es aproximadamente igual a la frecuencia de corte, f_c, mientras que M determina la longitud del núcleo. Las muestras en el comienzo y el fin de los núcleos del filtro son tan pequeñas que no pueden verse en las gráficas.

Aunque sean pequeñas estas muestras, colectivamente tienen un gran efecto en el desempeño del filtro. Esto es también por lo que la representación en punto flotante es típicamente utilizada para implementar filtros de ventanamiento síncrono. Normalmente a los números enteros no tienen suficiente rango dinámico para captar la gran variación de valores contenidos en el núcleo del filtro. El filtro de ventanamiento síncrono en el dominio de tiempo trabaja muy mal ya que la respuesta de paso tiene rebose y resonancia, por lo que este no es un filtro para señales con información codificada en el dominio de tiempo.

Si se debe tener un número específico de muestras en el núcleo del filtro, tal y como usa el FFT, simplemente agrega ceros de un extremo al otro. Por ejemplo, con M = 100, se pueden hacer las muestras de 101 a 127 iguales a cero, resultando en un núcleo del filtro de 128 puntos de largo.

3.3 Ejemplos de filtros de ventanamiento síncrono.

Un electroencefalograma es una medida de la actividad eléctrica del cerebro. Puede ser detectado como milivoltios en el nivel de señales aparecidas en un conjunto de electrodos pegados a la superficie de la cabeza. En el cerebro cada célula del sistema nervioso genera pulsos eléctricos pequeños. El electroencefalograma es el resultado combinado de un número enorme de estos pulsos eléctricos siendo generado de una forma coordinada. Aunque la relación entre el pensamiento y esta coordinación eléctrica son poco conocidas, diferentes frecuencias en el electroencefalograma pueden ser identificadas con condiciones mentales específicas. En estado de relajación, el patrón predominante del electroencefalograma será una oscilación lenta entre aproximadamente 7 y 12 Hz Esta onda de forma es llamada el estado **alfa**, y es asociado con la satisfacción y un nivel disminuido de atención. Abriendo los ojos y mirando alrededor genera al electroencefalograma que se convierta en el estado **beta**, ocurriendo entre aproximadamente 17 y 20 Hz. Otras frecuencias y forma de ondas son vistas en niños, diferentes niveles de sueño, y en diferentes desórdenes del cerebro como la epilepsia.

En este ejemplo, se da por supuesto que la señal del electroencefalograma ha sido amplificada mediante electrónica analógica, y entonces digitalizada en una tasa de muestreo de 100 muestras por segundo. Adquiriendo datos para 50 segundos produce una señal de 5,000 puntos. El cometido es separar el estado alfa de las betas. Para hacer esto, se diseña un filtro paso bajo digital con una frecuencia del corte de 14Hz o 0.14 de la tasa de muestreo.

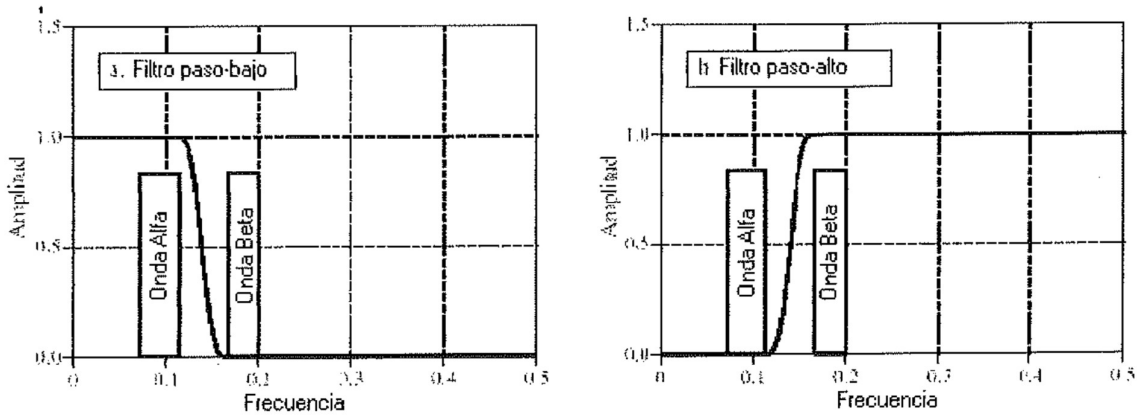


Figura 5. Ejemplo de filtros de ventanamiento síncrono. Los riunos Alfa y Beta de un electroencefalograma están separados por filtros paso-bajo y paso- alto con M = 100. El programa para implementar el filtro paso bajo se muestra en la tabla 1. El programa para el filtro paso bajo es idéntico a excepción de la inversión espectral de un núcleo de filtro paso bajo.

El ancho de la banda de transición estará colocado en 4 Hz ó 0.04 de la tasa de muestreo. En la ecuación 3, el núcleo del filtro necesita tener aproximadamente 101 puntos de longitud, y será arbitrariamente seleccionado para usar una ventana Hamming. La respuesta en frecuencia del filtro, obtenida al tomar la transformada de Fourier del núcleo del filtro, según se demuestra en la figura 5.

En un segundo ejemplo, diseñaremos un filtro de paso-banda para aislar un tono de señalización en una señal de audio, como cuando un botón en un teléfono es apretado. Se da por supuesto que la señal ha sido digitalizada a 10 kHz, y el cometido es aislar una banda de 80 Hz de frecuencias centradas en 2 kHz. En términos de la tasa de muestreo, se quiere bloquear todas las frecuencias por debajo de 0.196 y por encima de 0.204 (correspondiendo a 1960 Hz y 2040 Hz, respectivamente). Para lograr un ancho de banda de transición de 50 Hz (0.005 de la tasa de muestreo), se hace el núcleo del filtro de 801 puntos de largo, y se usa una ventana Blackman. La figura 6 muestra la respuesta en frecuencia. El diseño implica varias etapas. Primero, son diseñados dos filtros paso bajo, uno con un corte de 0.196, y el otro con un corte de 0.204. Este segundo filtro es entonces de espectro invertido, haciéndolo un filtro paso alto. Después, los dos núcleos del filtro son añadidos, resultando un filtro rechazo de banda. Finalmente, otras **inversiones espectrales** hechos en los filtros deseados de banda de paso.

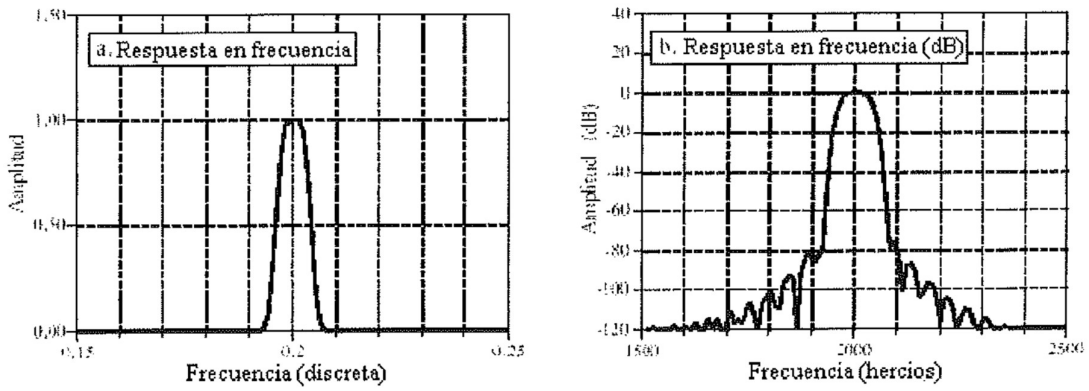
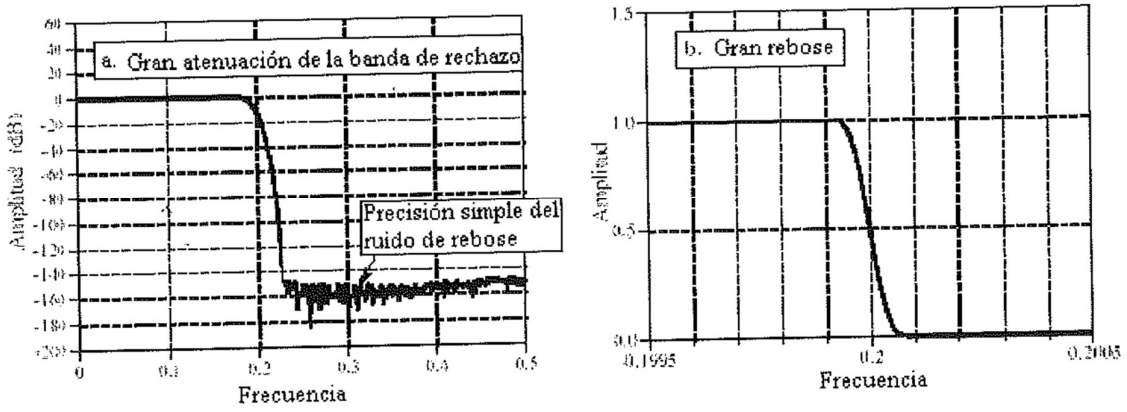


Figura 6. Ejemplo de filtro paso banda de ventanamiento síncrono. En (a) se muestra la respuesta en frecuencia resultante en escala lineal, mientras que en (b) se muestra en decibelios. La accisa de la frecuencia en (a) es expresada como una fracción de la frecuencia de muestreo, mientras que (b) es expresada en términos de señal analógica después de la digitalización.

3.4 Llevándolo al límite

El filtro de ventanamiento síncrono puede ser llevado a niveles de ejecución muy grandes sin sorpresas desagradables. Por ejemplo, para aislar una señal 1 mV dentro de una línea de conducción eléctrica de l 20V. El filtro del paso bajo que se necesita debe tener una atenuación de la banda de rechazo de al menos -120dB (un parte de un millón en decibelios). Como se ha señalado anteriormente, la ventana Blackman sólo provee - 74dB (una parte de cinco mil). Afortunadamente, la atenuación mayor de la banda de rechazo es fácil de obtener. La señal de entrada puede ser filtrada usando l\11 núcleo del filtro de ventanamiento síncrono convencional, proveyendo una señal intermedia. La señal intermedia puede pasar por el filtro una segunda vez, adicionalmente aumentando la atenuación de la banda de rechazo a - 148dB (1 parte de 30 millones). También es posible combinar las dos etapas en un solo filtro. El núcleo del filtro combinado es igual a la convolución de los núcleos del filtro de las dos etapas. Esto también significa que convalidando un núcleo del filtro **consigo mismo** resulta un núcleo del filtro con una atenuación de la banda de rechazo muy mejorada. Los inconvenientes son tener un núcleo de filtro más largo y un rebose más lento. La figura 7a demuestra la respuesta en frecuencia de un filtro paso-bajo de 201 puntos, formado por la convolución de una ventana síncrona Blackman de 101 puntos consigo mismo. (Si se necesita más que -100dB de atenuación de la banda de rechazo, se debe usar una doble precisión.

El ruido de rebose de precisión simple en las señales de la banda de paso erráticamente puede aparecer erráticamente en la banda de rechazo con amplitudes en el rango de – l00dB a - 120dB).



La figura 7b muestra otro ejemplo del gran desempeño del ventanamiento síncrono: Un filtro del paso bajo con 32.001 puntos en el núcleo. La respuesta en frecuencia aparece como se espera, con un rebose de 0.000125 de la tasa de muestreo. Si se intenta construir un filtro analógico que pase señales de continua a 1000 Hz con menos de una variación del 0.02 %, y bloquee todas las frecuencias por encima de 1001 Hz con un residuo de menos del 0.02 %. Los filtros de la figura 7 tienen **precisión simple**. Usar **precisión doble** permite que estos niveles de desempeño se extiendan un millón de veces.

La limitación mayor del filtro de ventanamiento síncrono es el tiempo de ejecución; Puede ser inaceptablemente largo si hay muchos puntos en el núcleo del filtro y se usa la convolución estándar. Un algoritmo de alta velocidad para este filtro (convolución FFT) es presentado más adelante. Los filtros recursivos también proveen una buena separación de frecuencia y es una alternativa razonable para el filtro de ventanamiento síncrono.

El ventanamiento síncrono es el núcleo de filtro óptimo para frecuencias separadas, los núcleos de los filtros necesitan técnicas más sofisticadas y así ser mejores. Hay que tener cuidado, antes de introducirse en este campo matemático, hay que considerar cual es la ganancia que se espera. El ventanamiento síncrono proveerá cualquier nivel de ejecución que se pueda necesitar. Lo que los métodos de diseño de filtros avanzados pueden proveer es un núcleo del filtro ligeramente más corto para un nivel dado de ejecución. Esto, a su vez, puede significar una velocidad de ejecución ligeramente más rápida. Hay que tener en cuenta que se puede obtener un menor retorno que el esfuerzo gastado.

4. Usos de los filtros

La mayoría de filtros tienen una de las cuatro respuestas estándar de frecuencia: El paso-bajo, paso-alto, paso-banda o rechazado-banda. En este apartado se presenta un método general de diseño de filtros digitales con una respuesta en frecuencia arbitraria, hecho a la medida a las necesidades de su aplicación particular. Los DSP sobresalen en esta área, resolviendo problemas que están muy por encima de las capacidades de electrónica analógica. Dos usos importantes de filtros personalizados se discuten en este apartado: La deconvolución, una forma de restituir señales que han experimentado una convolución no deseada, y un filtrado óptimo, el problema de separar señales con espectros de frecuencia superpuestos. Éste DSP es el mejor.

4.1 Respuesta en frecuencia arbitraria

Anteriormente se ha visto que los filtros de ventanamiento síncrono se pueden utilizar para diseñar filtros con cualquier respuesta en frecuencia. La única diferencia es cómo está la respuesta deseada se traslada del dominio de la frecuencia al dominio de tiempo. En el filtro de ventanamiento síncrono, la respuesta en frecuencia y el núcleo del filtro son ambas representados por ecuaciones, y la conversión entre ellos está hecha evaluando las fórmulas matemáticas de la transformada de Fourier. En el método aquí presentado, ambas señales están representadas por una gran colección de números, con un programa computerizado (el FFT) usando para encontrar uno desde él otro.

La figura 1 muestra un ejemplo de cómo trabaja éste. La respuesta en frecuencia que se quiere que el filtro a produzca es el mostrado en (a). Por no decir más, es muy irregular y sería virtualmente imposible obtenerla con electrónica analógica. Esta respuesta en frecuencia ideal está definida por un conjunto de números que han sido seleccionados, no de alguna ecuación matemática. En este ejemplo, hay 513 muestras puestas entre O y 0.5 de la tasa de muestreo. Más puntos podrían usarse para representar mejor la respuesta en frecuencia deseada, mientras que un número más pequeño puede ser necesario para disminuir el tiempo de computación durante el diseño del filtro. Sin

embargo, estas preocupaciones son usualmente pequeñas, y 513 es una buena longitud para la mayoría de aplicaciones.

Además la gran colección del conjunto de magnitud deseada mostrada en (a), debe haber el correspondiente array de fase de la misma longitud.

En este ejemplo, la fase de la respuesta en frecuencia deseada es completamente cero (este array no está mostrado en 1). Así como el array de magnitud, el array de fase puede ser cargado con cualquier gráfica arbitraria que se quiere que produzca el filtro. Sin embargo, hay que recordar que las primeras y últimas muestras (esto es, 0 y 512) y el array de fase deben tener un valor de cero (ó un múltiplo de 2, que es la misma cosa). La respuesta en frecuencia también puede estar especificada en forma rectangular definiendo las entradas de array para las **partes reales** e **imaginarias**, en lugar de usar la magnitud y la fase.

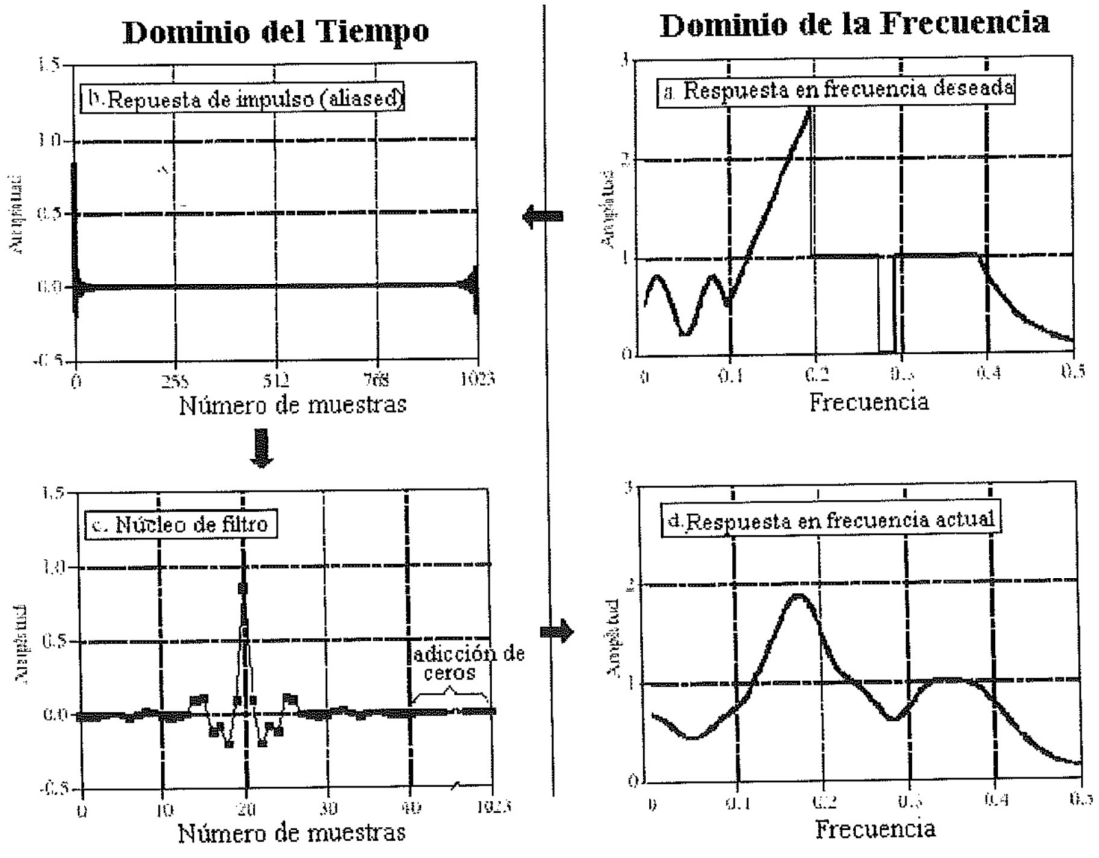


Figura !1. Ejemplo de diseño de filtro FIR.

El siguiente paso es llevar la DFT Inversa para mover el filtro al dominio del tiempo. La forma más rápida de hacer esta conversión del dominio de la frecuencia a la forma rectangular, y entonces usar el FFT Inverso. Esto resulta en una señal de muestreo de 1024 muestras que vaya desde 0 a 1023, tal y como se muestra en (b). Ésta es la respuesta de impulso que corresponde a la respuesta en frecuencia requerida; Sin embargo, no es adecuado para el uso como núcleo del filtro. Tal y como se ha visto en el apartado anterior, necesita ser desplazado, truncado, y aplicado el filtro de ventanamiento. En este ejemplo, se diseña el núcleo del filtro con M = 40, por ejemplo, 41 puntos que van desde la muestra 0 a la 40. Al igual que con el filtro de ventanamiento síncrono, los puntos cerca de los extremos del núcleo del filtro son tan pequeñas que parecen ser cero cuando son ejecutados. No hay que pensar que pueden ser suprimidos.

El último paso es probar el núcleo del filtro. Esto se realiza tomando el DFT (usando el FFT) para encontrar la actual respuesta en frecuencia, tal y como se muestra en (d). Para obtener mejor resolución en el dominio de frecuencia, hay que rellenar el núcleo del filtro con ceros antes del FFT. Por ejemplo, usar 1024 muestras totales (41 en el núcleo del filtro, y 983 ceros), resultan 513 muestras entre 0 y 0.5.

Como se muestra en la figura 2, la longitud del núcleo del filtro determina como de bueno es la respuesta **rea**l de frecuencia en comparación con la respuesta en frecuencia **deseada**.

El excepcional desempeño de los filtros digitales FIR es aparente; Virtualmente cualquier respuesta en frecuencia puede ser obtenida si un núcleo del filtro suficientemente larga es usado.

Éste es el método entero del diseño; Sin embargo, hay a un sutil asunto teórico que necesita ser aclarado. No es posible usar directamente la respuesta de impulso demostrada en 1b como el núcleo del filtro. Aunque (a) es la transformada de Fourier de (b), no convoluciona una señal de entrada con (b) produciendo la respuesta exacta de frecuencia que se necesita.

Al diseñar un filtro personalizado, la respuesta deseada de frecuencia está definida por los valores en un array. Hay que tener en cuenta lo que hace la respuesta en frecuencia entre los puntos especificados. Puede haber dos casos. En el caso más favorable, la respuesta en frecuencia es una curva suave entre las muestras definidas. En él más desfavorable hay grandes fluctuaciones en medio, como en (b). Esto se puede mostrar

rellenado con un número grande de ceros, y entonces tomando la DFT. La respuesta en frecuencia obtenida mediante este método demuestra el comportamiento errático entre las muestras originalmente definidas.

centellador, los átomos cercanos están excitados a un nivel de energía superior. Estos átomos aleatoriamente desexcitados, cada uno produciendo un solo fotón de luz visible. El resultado neto es un pulso de luz cuya amplitud se decae en varios centenares de ns (para el yodato sódico). Como la llegada de cada rayo gamma es un impulso, el pulso de salida del detector (por ej la exponencial de una cara) es la respuesta de impulso del sistema.

La figura 4a muestra los pulsos generados por el detector en respuesta a los rayos gamma llegados al azar. La información que se debe extraer de esta señal de salida es la amplitud de cada pulso, lo cual es proporcional a la energía de los rayos gamma que son generados. Ésta información es importante ya que la energía puede informar acerca de donde el rayo gamma ha estado.

Por ejemplo, puede proveer información médica en un paciente, puede contar la edad una galaxia distante, puede detectar una bomba en una maleta, etc. Todo estaría bien si sólo un único rayo gamma fuera detectado, pero normalmente este no es el caso. Como se muestra en (a), dos o más pulsos se pueden sobreponer, desplazando la amplitud medida. Una respuesta para este problema es la **deconvolución** de la señal de salida del detector, haciendo los pulsos más estrechos. Lo ideal sería que cada pulso re ensamblara el impulso original. Como se puede sospechar, esto no es posible y se debe fijar para un pulso que es finito en longitud, pero significativamente más pequeño que el pulso detectado. Este cometido es ilustrado en la figura 4b.

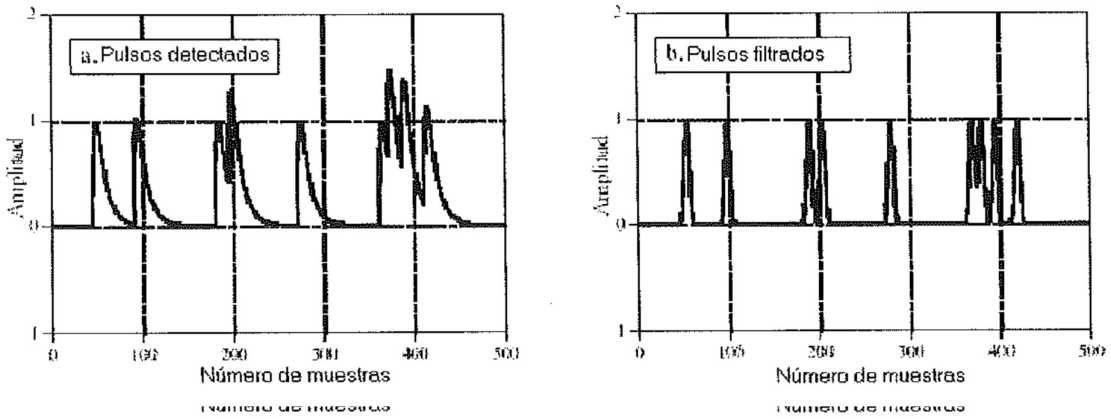


Figura 4. Ejemplo de deconvolución. La figura (a) muestra la señal de salida de un detector de rayos gamma en respuesta de una serie de rayos gamma que llegan aleatoriamente. El filtro de deconvolución es diseñado para convertir (a) en (b), para reducir la anchura de los filtros deseado se hace más estrecho, entonces la ganancia del filtro de deconvolución debe ser incluso mayor a altas frecuencias.

El problema es que los errores pequeños son muy inclementes en esta situación. Por ejemplo, si alguna frecuencia es amplificada por 30, entonces cuando sólo son requeridos 28, la señal de deconvolución no será buena.

Cuándo la deconvolución es empujada a niveles mayores de desempeño,
Las características de la convolución no deseada deben ser entendidas con mayor **exactitud y precisión**. Existen siempre incógnitas en el mundo de las aplicaciones reales, causadas por factores como: El ruido electrónico, la deriva de la temperatura, la variación entre dispositivos, el etc. Estas incógnitas colocan un límite en cómo debe trabajar la deconvolución.

Aun si la convolución no deseada queda perfectamente entendida, hay todavía un factor qué limita el desempeño de deconvolución: El ruido. Por ejemplo, la mayoría de convoluciones no deseadas toman forma en un filtro de paso bajo, reduciendo la amplitud de los componentes de alta frecuencia en la señal. La deconvolución corrige esto amplificando estas frecuencias. Sin embargo, si la amplitud de estos componentes cae por debajo del ruido inherente del sistema, entonces la información contenida en estas frecuencias se pierde y

no puede ser recuperado mediante ningún proceso. Tratar de rescatar estos datos sólo amplificará el ruido. En un caso extremo, la amplitud de algunas frecuencias se puede reducir completamente a cero. Esto no sólo anula la información, tratará de hacer que el filtro de deconvolución tenga ganancia infinita en estas frecuencias. La solución es diseñar un menos filtro menos agresivo de deconvolución y/ o colocar límites de cuál es la ganancia permitida en cualesquiera de las frecuencias.

Si la señal tiene buen comportamiento y bajo ruido, es probable que se haga una mejora significante (un factor de 5-10). Si la señal es cambiante con el tiempo, no está adecuadamente bien entendida, o es ruidoso, la mejora no será tan significante (factor de 1-2). La deconvolución exitosa implica una gran cantidad de pruebas. Si se dedica a algún nivel, se puede ir más allá. Ninguna cantidad de trabajo teórico va a permitir pasar por encima de este proceso iterativo.

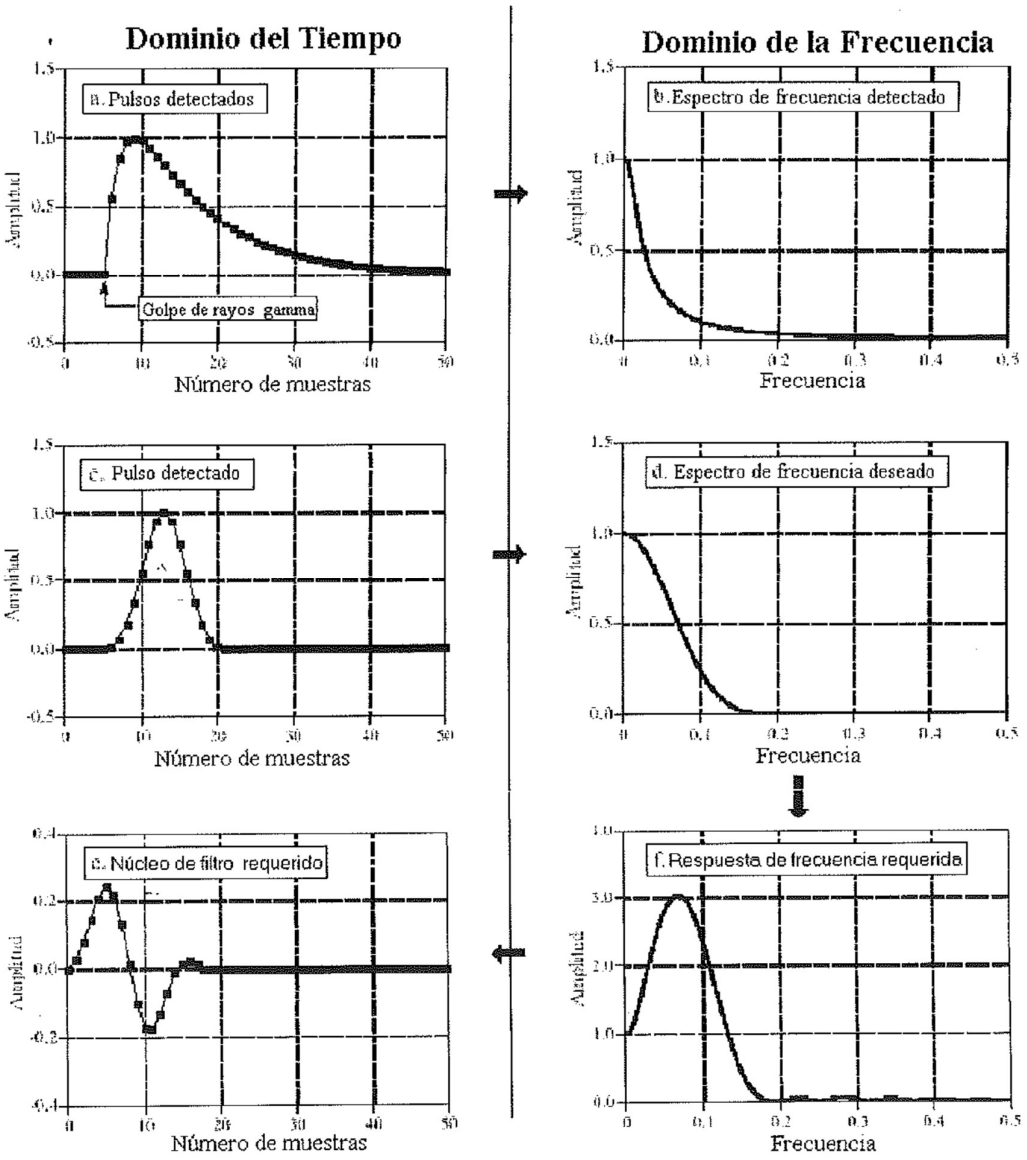


Figura 5. Ejemplo de deconvolución en el dominio del tiempo y de la frecuencia. La respuesta de impulso del ejemplo de detector de rayos gamma es mostrada en (a), mientras que la respuesta de impulso deseada se muestra en (c). El espectro de frecuencia de estas dos señales se muestran en (b) y(d), respectivamente. El filtro que cambia (a) a (c) tiene una respuesta en frecuencia, (f), igual a (b) dividido entre (d). El núcleo del filtro de este filtro, (e), es luego encontrada de la respuesta en frecuencia usando el método de diseño de filtro común (inverso DFT, truncación, ventanamiento). Solo las magnitudes de las señales en el dominio de la frecuencia se muestran en esta figura; sin embargo, las fases son no nulas y deben ser usadas.

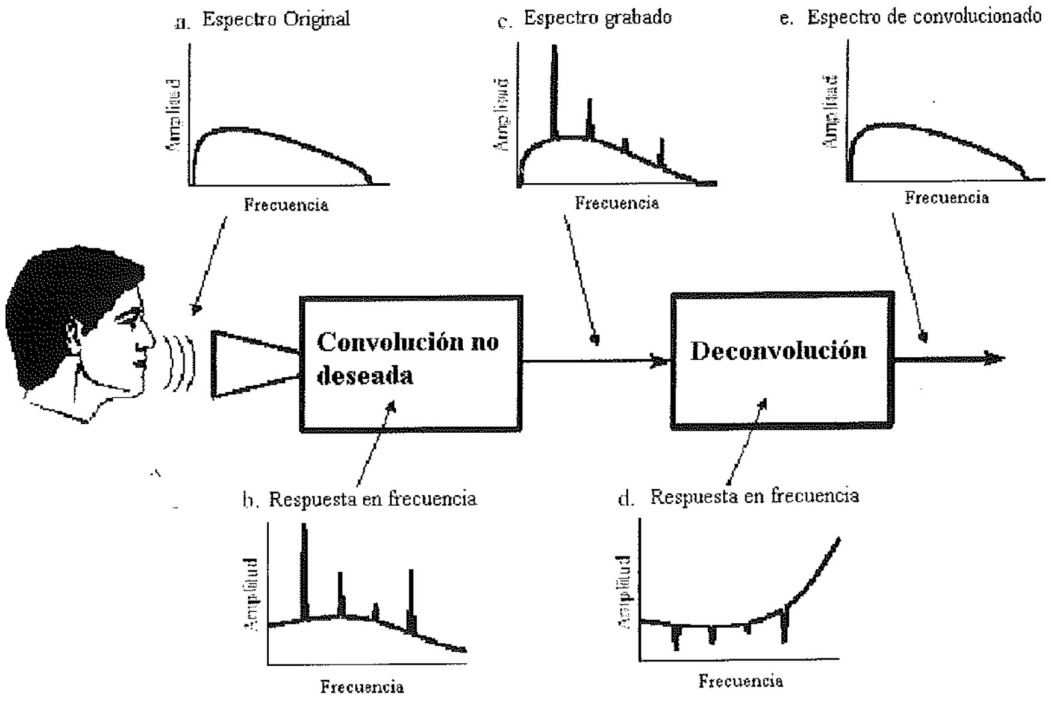


Figura 6. Deconvolución de viejas grabaciones de fonógrafo.

La deconvolución también puede ser aplicada para el **dominio de frecuencia** de las señales codificadas. Un ejemplo es la recuperación de viejas grabaciones de canciones. La figura 6 demuestra un acercamiento general. El espectro de frecuencia de la señal de audio original es ilustrado en (a). La figura (b) demuestra la respuesta en frecuencia del equipo de grabación, una curva relativamente suave excepto por varios picos afilados de resonancia. El espectro de la señal registrada, mostrada en (c), es igual al espectro original, (a), multiplicado por la respuesta de frecuencia irregular, (b). El cometido de la deconvolución es contrarrestar la convolución indeseada. En otras palabras, la respuesta en frecuencia del filtro de deconvolución, (d), debe ser la **inversa** de (b). Esto es, cada pico de (b) es cancelado por una depresión correspondiente en (d). Si este filtro fuera perfectamente diseñado, entonces la señal resultante tendría un espectro, (e), idéntico a la original. Este es un problema **deconvolución ciega**.

Los problemas de deconvolución ciega son normalmente acometidos haciendo una estimación o una suposición acerca de los parámetros desconocidos. Para ocuparse de este ejemplo, el espectro común de la música original - se asume - hace juego con el espectro común de la misma música realizada por un cantante presente de día usando un equipo moderno. El espectro común es encontrado por técnicas de tales como dividir la señal en un número grande de segmentos, tomar el DFT para cada segmento, convertirlo en forma polar, y entonces promediar las magnitudes conjuntamente. En el caso más simple, la respuesta de frecuencia desconocida es tomada como el espectro común de la vieja grabación, dividida por el espectro común de la grabación moderna.

4.3 Filtros óptimos

La figura 7 a ilustra un problema común de filtrado: Tratando de extraer una onda de forma (en este ejemplo, un pulso exponencial) del interior del ruido aleatorio. C01no se muestra en (b), este problema no será más fácil en el dominio de frecuencia. La señal tiene un espectro compuesto principalmente de componentes de baja frecuencia. En contraste, el espectro del ruido es **blanco** (misma amplitud en todas las frecuencias). Como el espectro de la señal y del ruido está superpuesto, no está claro cuál es la mejor forma de ser separados. Hay que mirar estos tres filtros, siendo cada uno el más óptimo de diferente manera. La figura 8 muestra la respuesta del núcleo del filtro y

De frecuencia para cada uno de estos filtros. La figura 9 demuestra el resultado de usar estos filtros en la onda de forma del ejemplo de la figura 7a.

En el **filtro de desplazamiento corriente**, visto anteriormente, hay que recordar que cada punto de salida produce el desplazamiento de un cierto número de puntos de la señal de entrada. Esto genera en el núcleo del filtro un pulso rectangular con una amplitud igual al recíproco del número de puntos en el desplazamiento. El filtro común de desplazamiento es óptimo en el sentido que provee la respuesta más rápida de paso para una reducción dada de ruido.

Como se muestra en la figura 8a, el núcleo del filtro del filtro matched es el mismo que la señal buscada detecta excepto si ha sido desplazada de la izquierda a la derecha.

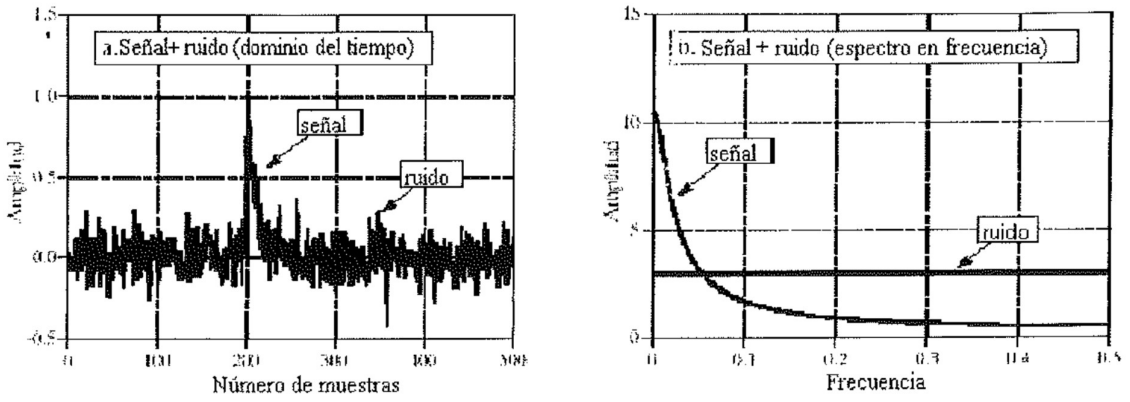


Figura 7. Ejemplo de filtrado óptimo. En (a) se muestra un pulso exponencial en una señal aleatoria. El espectro de frecuencia del pulso y el ruido se muestran en (b). Como la señal y el ruido están superpuestos tanto en el dominio del tiempo y de la frecuencia, el mejor camino para separar ambos no es obvio.

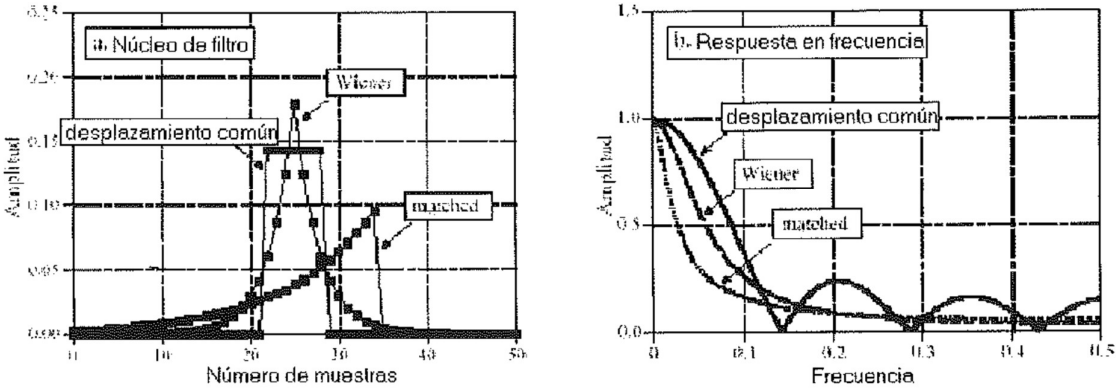


Figura 8. Ejemplo de filtros óptimos. En (a), se muestran tres filtros óptimos, cada uno óptimo en cierto sentido. La correspondiente respuesta en frecuencia se muestra en (b). El filtro de desplazamiento común es diseñado para tener un pulso rectangular de un núcleo de filtro. En comparación, el núcleo del filtro es parecido a la señal que ha sido detectada. El filtro Wiener es diseñado en el dominio de la frecuencia, basa en la relativa cantidad de señal y ruido presente en cada frecuencia.

La idea detrás del filtro matched es la **correlación**, y este desplazamiento es requerido para realizar la **correlación** usando la **convolución**. La amplitud de cada pm1to de la señal de salida es una medida de qué tan adecuadamente el núcleo del filtro hace juego con la sección correspondiente de la señal de entrada.

Hay que recordar que la salida de un filtro matched necesariamente no se parece a la señal que ha sido detectada. Esto realmente no tiene importancia; Si un filtro matched es usado, entonces la forma de la señal de blanco ya debe ser conocida. El filtro matched es óptimo en el sentido que la parte superior del pico está más por encima del ruido que puede ser logrado con cualquier otro filtro lineal (figura 9b).

El **Filtro Wiener** (nombrado después de la estimación óptima de la teoría de Norbert Wiener) separa señales basadas en sus espectros de frecuencia. Tal y como se muestra en figura 7b, en algunas frecuencias hay en su mayor parte señal, mientras en otro ruido. Parece lógico que las frecuencias " que son señales en su mayor parte" deben pasar a través del filtro, mientras que las que son "en su ·mayor parte ruido" deberían ser bloqueadas. El filtro Wiener lleva esta idea un paso más; La ganancia del filtro para cada frecuencia es determinada por la cantidad relativa de señal y el ruido a esa frecuencia:

$$H[f] = \frac{S[f]^2}{S[f]^2 + N[f]^2}$$

Ecuación 1. Filtro Wiener. La respuesta en frecuencia, representada por H[/], esta determinada por el espectro en frecuencia del ruido, N[/], y la señal, S[/]. Únicamente son importantes las magnitudes; todas las fases son cero.

Esta relación se usa para convertir los espectros de la en figura 7b en la respuesta en frecuencia del filtro Wiener en la figura 8b. El filtro Wiener es óptimo en el sentido que maximiza la relación entre la energía de la señal y la del ruido (sobre la longitud de la señal, no en cada punto individual). Un apropiado núcleo del filtro es diseñado para la respuesta en frecuencia Wiener usando el método habitual.

Mientras las ideas sobre estos filtros óptimos son matemáticamente elegantes, a menudo salen mal en el sentido práctico. Esto no debe decir que nunca debería ser usado. A continuación se ven varios inconvenientes de su uso.

Primero, la diferencia entre las señales en la figura 9 es muy poco sorprendente. De hecho, si no se dice que los parámetros eran optimizados, entonces probablemente no se podría contar todo mirando las señales. Éste es normalmente el caso para problemas que implican solapamiento de espectros de frecuencia. La cantidad pequeña de desempeño adicional obtenida de un filtro óptimo no puede valer la incrementada complejidad del programa, el esfuerzo de diseño adicional, o el tiempo de ejecución más largo.

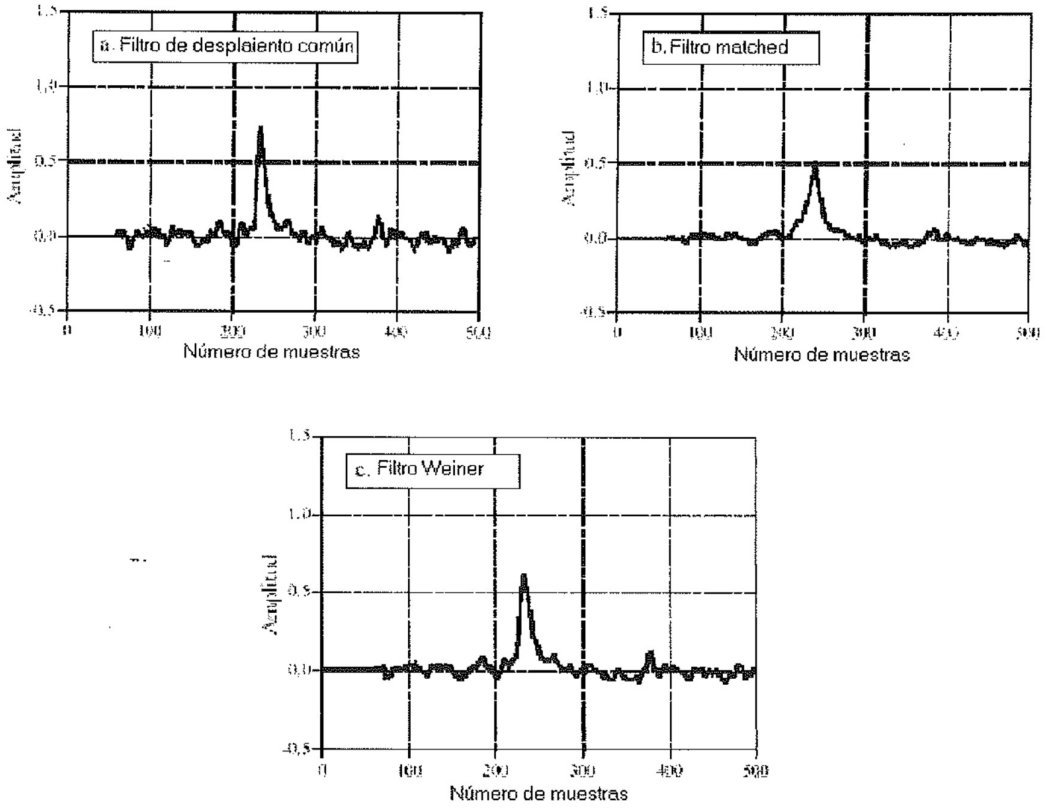


Figura 9. Ejemplo de uso de tres filtros óptimos. Estas señales resultan del filtrado de las formas de onda de la figura 7 con los filtros de la figura 8. Cada uno de estos filtros es óptimo en cierta manera. En (a) el filtro de desplazamiento común produce la respuesta para un nivel dado de reducción del ruido aleatorio. En (b), el filtro matched produce un pico

que esta más lejos del ruido residual que se provee mediante cualquier otro filtro. En (c), el filtro Wiener optimiza la relación señal ruido.

En el segundo: El Wiener y los filtros matched están completamente determinados por las características del problema. Otros filtros, como el ventanamiento síncrono y desplazamiento corriente, pueden estar hechos a la medida de nuestros gustos. Como se ha expresado anteriormente, cada uno de estos filtros es óptimo de una forma específica. Esto es rara vez suficiente para pretender que el problema entero haya sido optimizado, especialmente si las señales resultantes son interpretadas por un observador humano. Por ejemplo, un ingeniero biomédico podría usar un filtro Wiener para maximizar la relación señal-ruido en un electro-cardiograma. Sin embargo, no es obvio que éste también sea optimice la habilidad física de un médico para detectar actividad irregular del corazón mirando la señal.

Tercero: El Wiener y filtro matched deben ser realizados por **convolución,** haciéndolo extremadamente lento en la ejecutar. Aún con las mejoras de velocidad discutidas en el siguiente apartado (convolución de FFT), el tiempo de computación puede ser excesivamente largo. En contraste, los filtros **recursivos** son mucho más rápidos, y pueden proveer un nivel aceptable de ejecución.

5. Convolución FFT.

En este apartado se presentan dos técnicas importantes de los DSP, el **método de adicción de la superposición**, y la **convolución** del FFT. La superposición es el método que se usa para despedazar señales largas en segmentos más pequeños, para que se pueda procesar más fácilmente. La convolución FFT usa el método de la adicción de la superposición junto con la Transformada rápida de fourier, permitiendo a las señales ser convalidadas multiplicando su espectro en frecuencia. Para núcleos de filtro más largos que aproximadamente 64 puntos, la convolución FFT es más rápida que la convolución estándar, produciéndose un resultado exactamente igual.

5.1 El método de la adicción de la superposición.

Existen muchas aplicaciones de los DSP donde una señal larga debe ser filtrada en segmentos. Por ejemplo, la alta fidelidad del **audio** digital requiere una tasa de datos de casi 5 Mbytes/ min. , Mientras el **video** digital requiere aproximadamente de 500 Mbytes/ min. Con el ratio de los datos es muy alto, es común que los ordenadores no tengan suficiente memoria para mantener simultáneamente la señal entera para ser procesada. Existen también sistemas que procesan segmento por segmento debido a que funcionan en **tiempo real**. Por ejemplo, las señales telefónicas no pueden ser retrasadas más de varios centenares de milisegundos, limitando la cantidad de datos que están disponibles para el procesamiento a cada instante. En otras aplicaciones, el procesa miento puede requerir que la señal sea segmentada. Un ejemplo es la convolución FFT, el tema principal de este capítulo.

El método de la adicción de la superposición está basado en la técnica fundamental en DSP: (1) descompone la señal en componentes sencillos, (2) procesa cada uno de los componentes en alguna forma útil, y (3) recombina los componentes procesados en la señal final. La figura 1demuestra un ejemplo de cómo está echo éste para él método de la adicción de la superposición. La figura (a) es la señal que debe ser filtrada, mientras (b) muestra el núcleo del filtro para ser

usado, un filtro paso-bajo de ventanamiento síncrono. Pasando a la base de la figura, (i) se muestra la señal filtrada, una versión suavizada de (a).

La clave de este método es cómo las **longitudes** de estas señales son afectadas por la convolución. Cuando una señal de N muestras es convolucionada con un núcleo de filtro de M muestras, la señal de salida es de $N + M - 1$ muestras de longitud.

Por ejemplo, la señal de entrada, (a), es de 300 muestras (que van de 0 a 299), el núcleo del filtro, (b), tiene 101 muestras (que van de 0 a 100), y la señal de salida, (i), tiene 400 muestras (de 0 a 399).

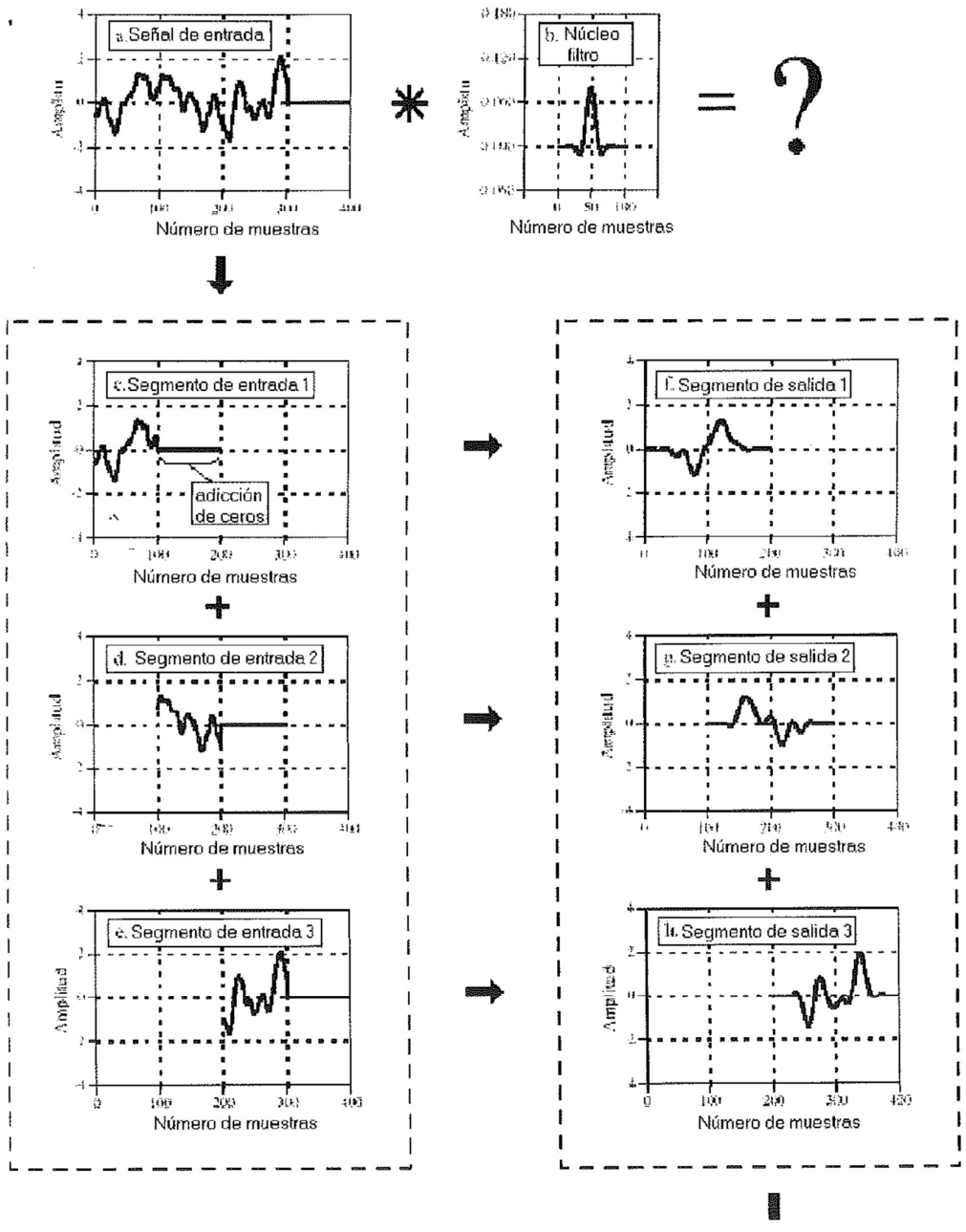
En otras palabras, cuando una señal de **N** muestras es filtrada, será expandida M - 1 puntos a la derecha. (Esto da por supuesto que el núcleo del filtro va de los índices 0 a M. Si los índices negativos son usados en el núcleo del filtro, entonces la expansión también será a la izquierda). En (a), los ceros se han agregado a la señal entre las muestras 300 y 399 para ilustrar donde se producirá la expansión. No hay que confundirse con los pequeños valores en los extremos de la señal de salida, (i). Éste es simplemente un resultado del núcleo del filtro de ventanamiento síncrono teniendo en pequeños valores cerca de sus extremos. Todas las 400 muestras en (i) son no nulas, aunque alguna de ellas sea muy pequeña para ser visto en la gráfica.

Las figuras (c), (d) y (e) muestran la descomposición usada en él método de adicción de la superposición. La señal está dividida en segmentos, con cada segmento se hacen 100 muestras de la señal original. Además, 100 ceros se añaden a la derecha de cada segmento. En la siguiente etapa, cada segmento es filtrado individualmente convolucionándolo con el núcleo del filtro. Esto produce los segmentos de salida mostrados de en (t), (g), y (h). Como cada segmento de entrada tiene 100 muestras de longitud, y el núcleo del filtro tiene 101 muestras de largo, cada segmento de salida tendrá 200 muestras de longitud. El concepto a entender es que los 100 ceros añadidos para cada segmento de entrada permiten expansión durante la convolución.

Hay que tener en cuenta que la expansión resulta en los segmentos de salida superpuesta sobre los otros. Estos segmentos superpuestos en la salida son sumados para producir la señal de salida, (i). Por ejemplo, las muestras desde 200 a 299 en (i) son encontradas añadiendo las correspondientes muestras en (g) y (h). El método de superposición produce exactamente la misma señal de salida que la convolución directa. La desventaja es una mucha mayor complejidad del programa para seguir la pista a las muestras superpuestas.

5.2 **Convolución FFT.**

La convolución FFT usa el principio de la **multiplicación** en el dominio en frecuencia corresponde a la **convolución** en el dominio de tiempo. La señal de entrada es transformada al dominio en frecuencia usando la DFT, multiplicado por la respuesta en frecuencia del filtro, y entonces volviendo a transformar al dominio de tiempo usando la DFT Inversa.



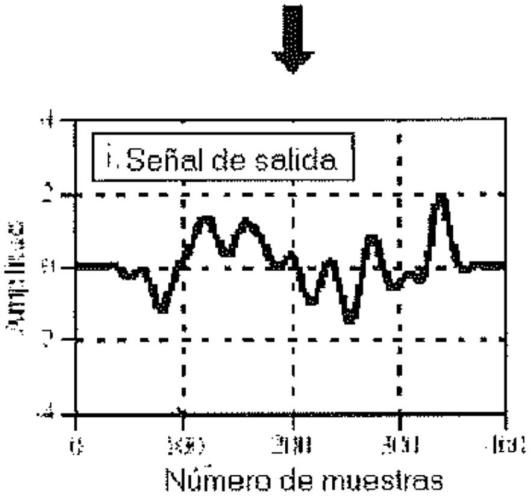


Figura 1. Método de adicción de la superposición.

Esta técnica básica fue conocida desde Fourier; Sin embargo, nadie la tuvo en cuenta. Esto es porque el tiempo requerido para calcular la DFT es más largo que el tiempo para calcular directamente la convolución. Esto cambia en 1965 con el desarrollo de la Transformada rápida de Fourier (FFT). Usando el algoritmo FFT para calcular la DFT, la convolución por medio del dominio de la frecuencia puede ser más rápida que convolucionando directamente las señales en el dominio de tiempo. El resultado final es el mismo; Sólo el número de cálculos se ha variado por un más algoritmo eficiente. Por esta razón, la convolución FFT es también llamada **convolución de alta velocidad**.

La convolución FFT usa el método de adicción de la superposición demostrada en la figura 1; únicamente varía la forma en que los segmentos de entrada son convertidos en los de salida. La figura 2 demuestra un ejemplo de cómo se convierte un segmento de entrada en un segmento de salida mediante convolución FFT. Para comenzar, la respuesta en frecuencia del filtro es encontrada tomando la DFT del núcleo del filtro, usando la FFT. Por ejemplo, (a) muestra un ejemplo del núcleo del filtro, un filtro paso de banda de ventanamiento síncrono. El FFT convierte esto en las partes reales e imaginarias de la respuesta de frecuencia, mostrado en (b) y (c). Estas señales de dominio en frecuencia no pueden aparecer en un filtro de paso-banda porque están en forma rectangular. Hay que tener en cuenta que la forma polar es normalmente la más conveniente para que los humanos entiendan el dominio de frecuencia, mientras que la forma rectangular es normalmente la más conveniente para cálculos matemáticos. Estas partes reales e

imaginarias se almacenan en el ordenador para usar cuando cada segmento está siendo calculado.

La figura (d) muestra el segmento de entrada para ser procesada. El FFT es usado para encontrar su espectro de frecuencia, mostrado en (e) y (f). El espectro en frecuencia del segmento de salida, (h) y (i) es entonces encontrado multiplicando la respuesta en frecuencia del filtro, (b) y (c), por el espectro del segmento de entrada, (e) y (f). Estos espectros consisten de partes reales e imaginarias, son multiplicados según la ecuación:

$$ReY[f] = ReX[f]\,ReH[f] - ImX[f]\,ImH[f]$$

$$ImY[f] = ImX[f]\,ReH[f] + ReX[f]\,ImH[f]$$

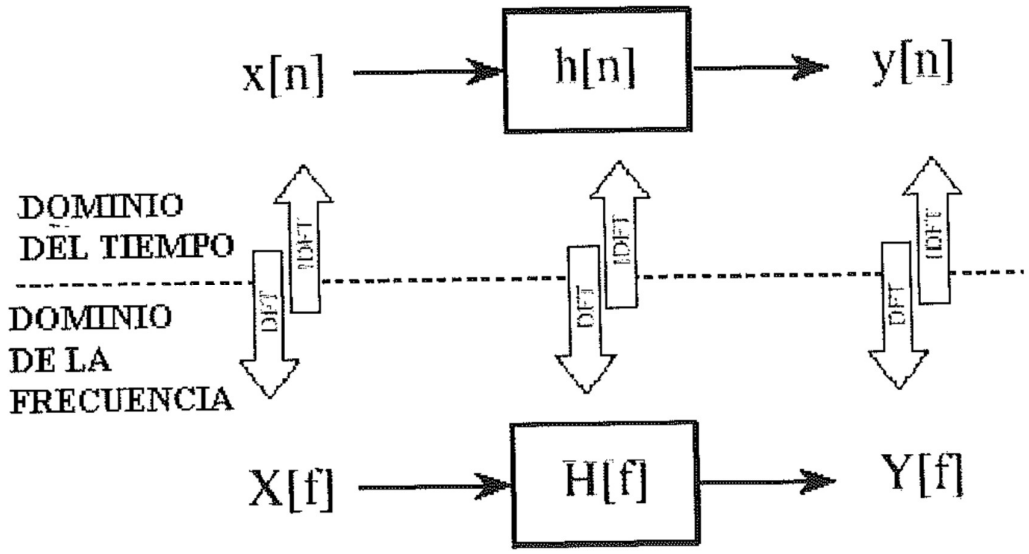
Multiplicación de señales en el dominio de la frecuencia en forma rectangular: $Y[f] = X[f] \times H[f]$.

La FFT inversa es entonces usada para encontrar el segmento de salida, (g), desde su espectro de frecuencia, (h) y (i). Es importante reconocer que este segmento de salida es exactamente igual que el que será obtenido mediante la convolución directa del segmento de entrada, (d), y el núcleo del filtro, (a).

Los FFTs deben ser lo suficientemente largos para que la **convolución circular** no tenga lugar. Esto quiere decir que la FFT debería tener la misma longitud que el segmento de salida, (g). En el ejemplo de la figura 2, el núcleo del filtro contiene 129 puntos y cada segmento contiene 128 puntos haciendo tener al segmento de salida 256 puntos de largo. Esto llama a los 256 puntos de FFTs para que sean usadas.

Esto significa que el núcleo del filtro, (a), debe ser rellenado con 127 ceros para que tenga un largo total de 256 puntos. Asimismo, cada uno de los segmentos de entrada, (d), debe ser rellenada con 128 ceros. Otro ejemplo, hay que imaginar que se necesita convolucionar una señal muy larga con un núcleo del filtro teniendo 600 muestras. Una alternativa sería usar segmentos de 425 puntos y 1024 puntos FFTs. Otra alternativa sería usar segmentos de 1449 puntos, y 2048 puntos FFTs.

En la siguiente figura se muestra la forma de pasar del dominio del tiempo al de la frecuencia y viceversa.



Mejoras de velocidad

Para conocer cuando la convolución FFT es más rápida que la convolución estándar, hay que mirar el largo del núcleo del filtro, como se muestra en La figura 3. El tiempo para la convolución estándar es directamente proporcional al número de puntos del núcleo del filtro. En comparación, el tiempo requerido para la convolución FFT aumenta muy lentamente, sólo como el **logaritmo** del número de puntos en el núcleo del filtro. El paso ocurre cuando el 1úcleo del filtro tiene aproximadamente de 40 a 80 muestras
(Dependiendo del hardware particular usado).

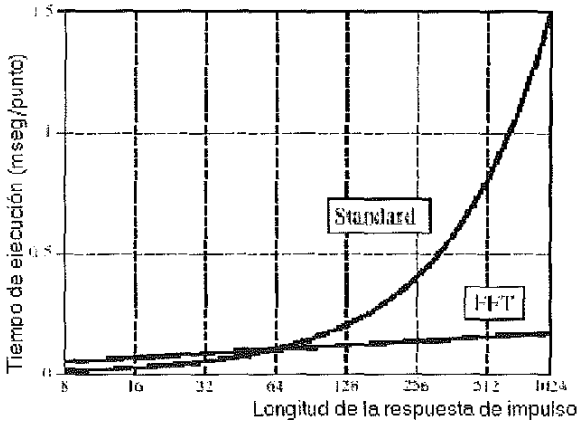


Figura 3. Tiempos de ejecución de la convolución FFT. La convolución FFT es más rápida que el método estándar cuando el núcleo del filtro es mayor que unos 60 puntos. Estos tiempos de ejecución son para un Pentium de 100 MHz, usando precisión simple de punto flotante.

Tiempos de ejecución de la convolución FFT. Esta convolución es más rápida que el método estándar cuando el núcleo del filtro es más largo que unos 60 puntos.

Un concepto importante: núcleos de filtros más cortos de aproximadamente 60 puntos pueden ser implementados más rápidos con convolución estándar, y el tiempo de ejecución es proporcional a la longitud del núcleo. Los núcleos de filtros más largos pueden ser implementados más rápidos con la convolución FFT. Con convolución FIIT, el filtro que el núcleo puede ser hecho tan largo como se quiera, con una muy pequeña penalización en el tiempo de ejecución. Por ejemplo, un núcleo de filtro de 16.000 puntos sólo requiere aproximadamente el **doble** de longitud que ejecutar uno con tan sólo 64 puntos.

La **velocidad** de convolución también dicta la **precisión** del cálculo. Esto es debido a que el error de rebose en la señal de salida depende del número total de cálculos, lo cual es directamente proporcional al tiempo de computación. Si la señal de salida es calculada más **rápido**, entonces también estará calculado más **precisamente**. Para el ejemplo, de convolucionar una señal con un núcleo de filtro de 1000 punto, con un punto flotante de precisión simple. Usando la convolución estándar, el ruido de rebose típico se espera que sea aproximadamente 1 parte de 20.000. En comparación, la convolución FFT tiene una orden de magnitud más rápida, y una orden de magnitud más precisa (por ejemplo, 1 parte de 200.000).

La convolución FFT se reserva para cuándo se tiene una gran cantidad de datos para procesar y sé necesitar una núcleo del filtro extremadamente largo. Pensando en términos de señales de un millón de muestras y un núcleo de filtro de **miles** de puntos.

6. Filtros recursivos

Los filtros recursivos son una forma eficiente de lograr una larga respuesta de impulso, sin tener que realizar una convolución larga. Se ejecutan muy rápidamente, pero tiene menos ejecución y flexibilidad que otros filtros digitales. Los filtros recursivos son también llamados filtros de **Respuesta infinita del impulso (IIR),** ya que sus respuestas de impulso están compuestas de exponenciales descendientes. Esto los distingue de los filtros digitales realizados por convolución, llamada filtros de **Respuesta finita del impulso (FIR).** Este capítulo es una introducción de cómo operan los filtros recursivos filtros funcione, y como pueden ser diseñados. En los apartados sucesivos se presentan métodos de diseño más sofisticados.

6.1 El Método Recursivo

Si se necesita extraer información de x[ ]. Se busca filtrar x[ ] para obtener y[ ], que contiene la información que nos interesa. Para obtener la señal de salida y[n] hay que aplicar la siguiente formula:

$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + a_3 x[n-3] + \cdots$$

La fuente más obvia de información es la señal de entrada, esto es los valores: x[n], x[n -1], x[n - 2],... Esto no es ninguna cosa más que una sencilla convolución, con los coeficientes: a 0, a 1, a 2,... Formando el núcleo de la convolución. Si éste fue todo el profesor lo hiciese, no habría mucha necesidad para esta historia, o este capítulo. Sin embargo, hay otra fuente de información a la que el profesor tiene acceso. Utilizando los valores previamente calculados de la señal de salida: y[n – 1], y[n – 2], y[n – 3], el algoritmo quedará de la forma:

$$\begin{aligned} y[n] = {} & a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + a_3 x[n-3] + \cdots \\ & + b_1 y[n-1] + b_2 y[n-2] + b_3 y[n-3] + \cdots \end{aligned}$$

Ecuación 1.Ecuación de recursión. En esta ecuación, x[ ] es la señal de entrada, y[ ] es la señal de salida y los coeficientes son a y b.

La relación entre los coeficientes de recursión y las respuestas del filtro es dada por una técnica matemática llamado la transformada-z. Por ejemplo, la transformada-z puede servir para tales tareas tales como:
Mutar entre los coeficientes de recursión y la respuesta en frecuencia, combinar etapas en cascada y etapas en paralelo en un solo filtro, diseñar sistemas recursivos que limite los filtros analógicos, etc. Desgraciadamente, la transformada-z es muy matemática, y más complicados que la mayoría de usuarios de DSP están dispuestos a tratar. Éste es el reino de aquellos que están especializados en DSP.

Existen tres formas de encontrar los coeficientes de la recursión sin tener que entender la z-transformada.

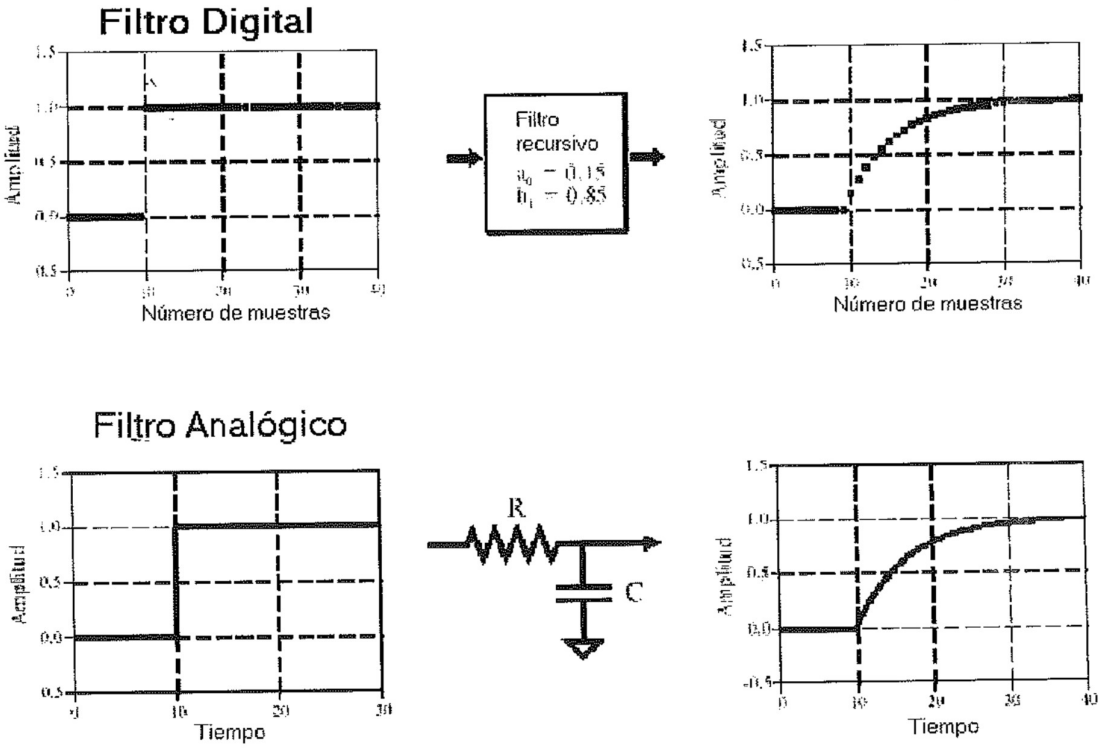


Figura 2.Filtro paso bajo de polo simple. Los filtros recursivos digitales pueden copiar a los filtros analógicos compuestos por resistencias y condensadores. Como se muestra en este ejemplo, un filtro recursivo de paso bajo de polo simple alisa el borde de una entrada paso, como en un filtro RC.

6.2 Filtros Recursivos de punto simple.

La figura 2 demuestra un ejemplo de lo que es llamado un filtro de paso-bajo de paso simple. Este filtro recursivo usa solo dos coeficientes, $a_0 = 0.15$ y $b_1 = 0.85$. Para este ejemplo, la señal de entrada es una función de paso. Como se debería esperar para un filtro de paso-bajo, la salida es una subida alisada para el nivel de estado seguro. Este paso bajo es el filtro recursivo completamente análogo a un filtro de paso bajo electrónico compuesto de una sola resistencia y un condensador.

La belleza del método recursivo está en su habilidad crear una ancha variedad de respuestas cambiando sólo algunos parámetros. Por ejemplo, la figura 3 muestra un filtro con tres coeficientes: a 0 = 0.93, a 1 = -0.93 y b 1 = 0.86.

Tal y como se muestra en las respuestas de paso similares, este filtro digital imita a un filtro de paso-alto RC.

Estos filtros recursivos de paso simple son algo que se debe conservar en la caja de herramientas DSP. Se puede usar entonces para tramitar señales digitales tal y como se usa en redes RC para gestionar señales analógicas. Esto incluye todo lo que se esperaría: supresión de CD, de ruido de alta frecuencia, formación de onda, de alisamiento, etc. Son fáciles para programar, rápidos de ejecutar, y producen pocas sorpresas.

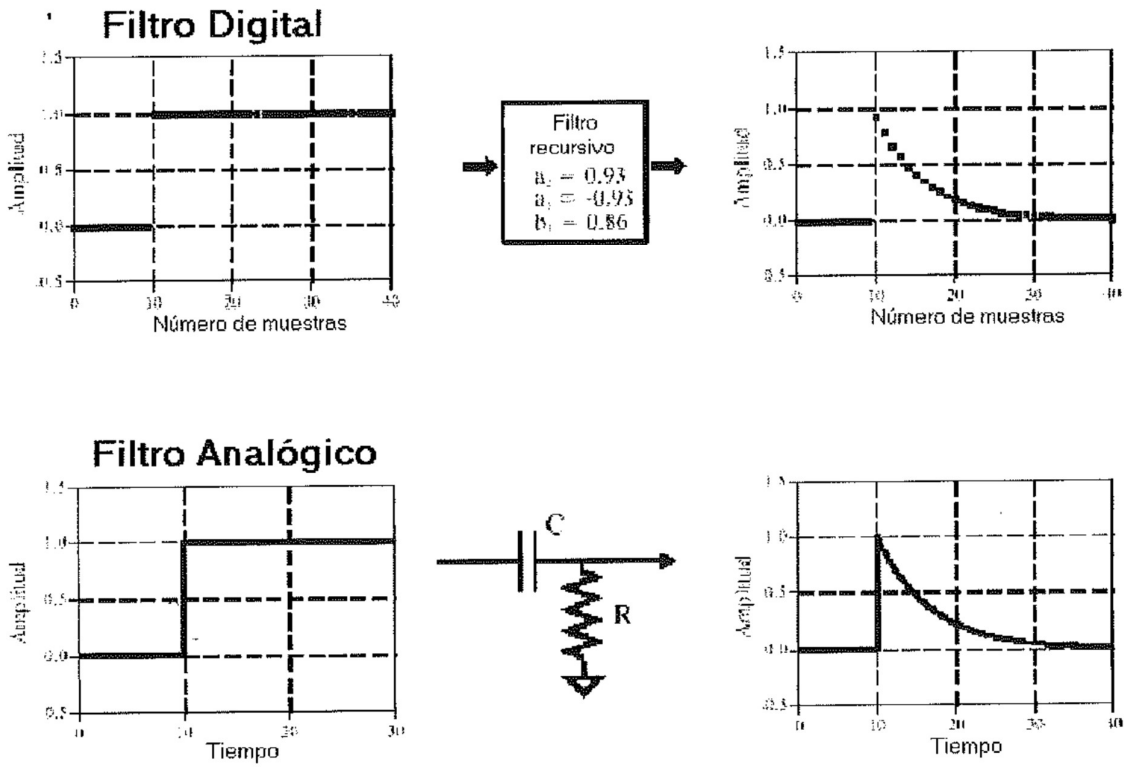


Figura 3.Filtro paso alto de polo simple. La selección del coeficiente proporcional puede también hacer al filtro recursivo idéntico al filtro paso alto RC. Estos filtros recursivos de polo simple pueden ser usados en DSP como son usados los circuitos RC en los filtros analógicos.

Los coeficientes son encontrados de estas ecuaciones de primer grado:

$$a_0 = 1 - x$$
$$b_1 = x$$

Ecuación 2.Filtro paso bajo de polo simple. La respuesta del filtro es controlada mediante el parámetro x, el cual toma un valor entre 0 y 1.

$$a_0 = (1 + x)/2$$
$$a_1 = -(1 + x)/2$$
$$b_1 = x$$

Ecuación 3.Filtro paso bajo de polo simple.

Las características de estos filtros están controladas por el parámetro, x, un valor entre el cero y uno. Físicamente, x es la cantidad de caída entre muestras adyacentes. Por ejemplo, x tiene 0.86 en la figura 3, queriendo decir que el valor de cada muestra en la señal de salida es 0.86, el valor de la muestra antes de eso. El superior valor de x, la más lenta caída. Hay que tener en cuenta que el filtro se vuelve inestable si la x es mayor que uno. Esto es, cualquier valor no nulo en la entrada incrementará la salida hasta que suceda él rebose.

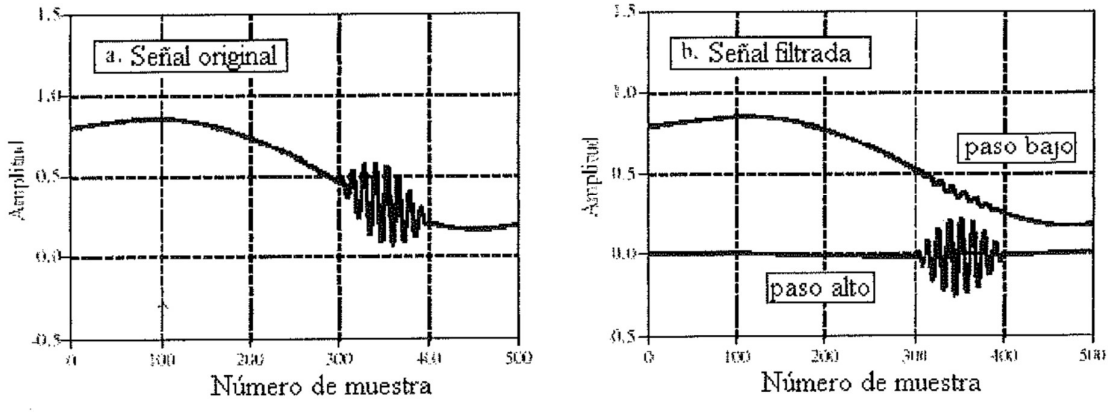


Figura 4.Ejemplo de filtros recursivos de polo simple. En (a), una ráfaga de alta frecuencia produce una lenta variación de la señal. En (b), un filtro pasa bajo de polo simple y otro paso alto son usados para separar los dos componentes. Los filtros de paso bajo usan x = 0.95, mientras que los paso alto x = 0.86.

El valor de x puede ser especificado directamente, o encontrado desde la constante deseada de tiempo del filtro. Tal como R x C número de segundos lleva un circuito RC a decaer un 36.8 % de su valor final, d es el número de muestras que lleva a un filtro recursivo a decaer a este mismo nivel:

$$x = e^{-1/d}$$

Ecuación 4.Constante de tiempo de los filtros de polo simple. Esta ecuación muestra la cantidad de caída entre muestras, x, con la constante de tiempo de los filtros, d, el número de muestras del filtro para decaer el 36.8%.

Por ejemplo, una caída de muestra a muestra de x = 0.86 corresponde a una constante de tiempo de d = 6.63 muestras (como se muestra en Figura 3).
Existe también una relación fija entre x y la frecuencia de corte -3dB, f e, del filtro digital.

$$X = e^{-2\pi fc}$$

Ecuación 5.Frecuencia de corte de los filtros de polo simple. La cantidad de caída entre muestras, x, es relacionada por la frecuencia de corte del filtro, Fc que es un valor entre 0 y 0.5.

Esto provee tres formas para encontrar los coeficientes "a" y " b", iniciando con la constante de tiempo, la frecuencia de corte, o justo directamente recogiendo x.

La figura 4 demuestra un ejemplo de uso de filtros recursivos de polo simple. En (a), la señal original es una curva suave, excepto una ráfaga de una onda sinusoidal de alta frecuencia. La figura (b) muestra la señal después de los pasos a través de los filtros de paso bajo y de paso alto. Las señales han estado separadas medianamente bien, pero no perfectamente, tal como si los circuitos sencillos RC fueron usados en una señal analógica.

La figura 5 muestra las respuestas de frecuencia de varios filtros recursivos de polo simple. Estas curvas son obtenidas haciendo a una función delta pasar por el filtro para encontrar la respuesta de impulso del filtro. El FFT es entonces usado para convertir la respuesta de impulso en la respuesta en frecuencia. En principio, la respuesta de impulso es infinitamente larga; Sin embargo, decae por debajo del ruido de rebose de precisión después de unas 15 o 20 constantes de tiempo. Por ejemplo, cuando el tiempo constante del filtro es d = 6.63 muestras, la respuesta de impulso se puede contener en unas 128 muestras.

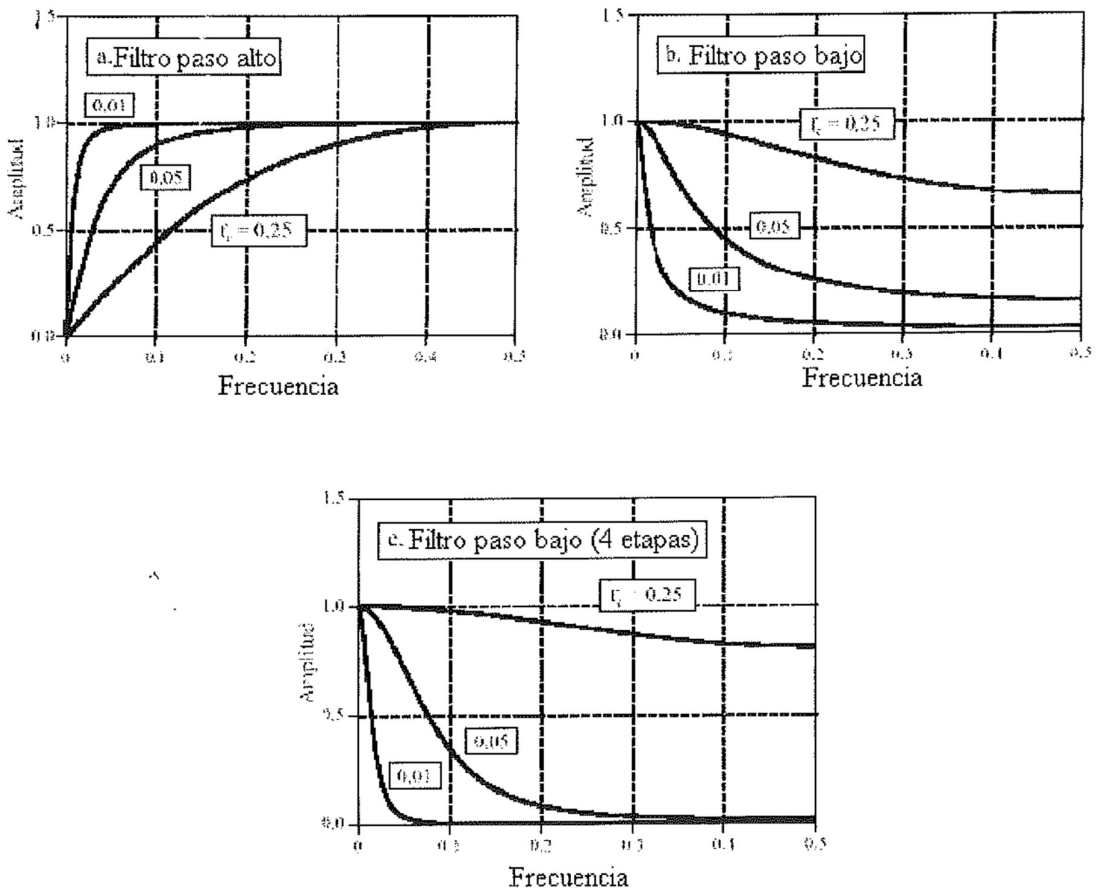


Figura 5. Respuesta en frecuencia de polo simple. Las figuras (a) y (b) muestra la respuesta en frecuencia de filtros recursivos de polos simple paso alto y paso bajo, respectivamente. La figura (c) muestra la respuesta en frecuencia de una cascada de cuatro filtros paso bajo. La respuesta en frecuencia de los filtros recursivos no es siempre la que cabe esperar, especialmente si el filtro es enviado a límites extremos. Por ejemplo, la gráfica de F e = 0,25 en (c) es un tanto inusual. Muchos factores son, incluyendo: aliasing, ruido de rebose, y la respuesta de fase no lineal.

El rasgo principal en la figura *5* es que los filtros recursivos de polo simple tienen poca habilidad para separar una banda de frecuencias de otra. En otras palabras, funcionan adecuadamente en el dominio de tiempo, y pobremente en el dominio de frecuencia. La respuesta de frecuencia puede ser mejorada ligeramente mediante varias etapas en cascada. Esto puede estar consumado de dos formas. La primera, que la señal puede ser hecha pasar a través del filtro varias veces. En segundo lugar, la transformada-z puede usarse para encontrar los coeficientes de recursión que combinan la cascada en una sola etapa. Ambas formas surgen efecto y son comúnmente usadas. En la figura (c) salta a la vista la respuesta en frecuencia de una cascada de cuatro filtros paso- bajo.

Aunque la atenuación de la banda de rechazo está significativamente perfeccionada, él rebose es todavía malo. Si se necesita mejorar el desempeño en el dominio de frecuencia, hay que mirar los filtros Chebyshev del siguiente capítulo.

El filtro pasa bajo de cuatro etapas es comparable con los filtros Blackman y Gaussiano, pero con una velocidad de ejecución mucho más rápida. Las ecuaciones de diseño para un filtro pasa-bajo de cuatro etapas de bajo son:

$$a_0 = (1-x)^4$$
$$b_1 = 4x$$
$$b_2 = -6x^2$$
$$b_3 = 4x^3$$
$$b_4 = -x^4$$

Ecuación 6. Filtro paso bajo de cuatro etapas. Estas ecuaciones proveen los coeficientes "a" y "b" para una cascada de cuatro filtros paso bajo de polo simple. La relación entre x y la frecuencia de corte viene dada mediante la ecuación 5, poniendo 14,445 en vez de 2D .

6.3 Filtros de banda estrecha

Una necesidad común en la electrónica y DSP es aislar una banda estrecha de frecuencias de una señal de ancha de banda más ancho. Por ejemplo, si se quiere eliminar una interferencia de 50 Hz en un sistema de instrumentación, o aislar los tonos de la llamada en una red telefónica. Dos tipos de respuestas de frecuencia están disponibles: El **paso de banda** y el **rechazado de banda** (también llamado un **filtro notch**).

En la figura 6 salta a la vista la respuesta en frecuencia de estos filtros, con los coeficientes de recursión provistos por las siguientes ecuaciones:

$$
\begin{aligned}
a_0 &= 1 - K \\
a_1 &= 2(K - R)\cos(2\pi f) \\
a_2 &= R^2 - K \\
b_1 &= 2R\cos(2\pi f) \\
b_2 &= -R^2
\end{aligned}
$$

Ecuación 7. Filtro de paso de banda. Un ejemplo de la respuesta en frecuencia es mostrado en la figura 6-a. Para usar estas ecuaciones, lo primero que hay que hacer es seleccionar la frecuencia central, f,y el ancho de banda BW. Ambos están expresados como una fracción de la tasa de muestreo, y se encuentran entre O y 0,5. Después se calcula R, K y por último los coeficientes de recursión.

$$a_0 = K$$
$$a_1 = -2K\cos(2\pi f)$$
$$a_2 = K$$
$$b_1 = 2R\cos(2\pi f)$$
$$b_2 = -R^2$$

En donde:

$$K = \frac{1 - 2R\cos(2\pi f) + R^2}{2 - 2\cos(2\pi f)}$$

$$R = 1 - 3BW$$

Ecuación 8. Filtro de rechazo de banda. Este filtro es comúnmente llamado filtro notch. Un ejemplo de la respuesta en frecuencia es mostrado en la figura 6-b.

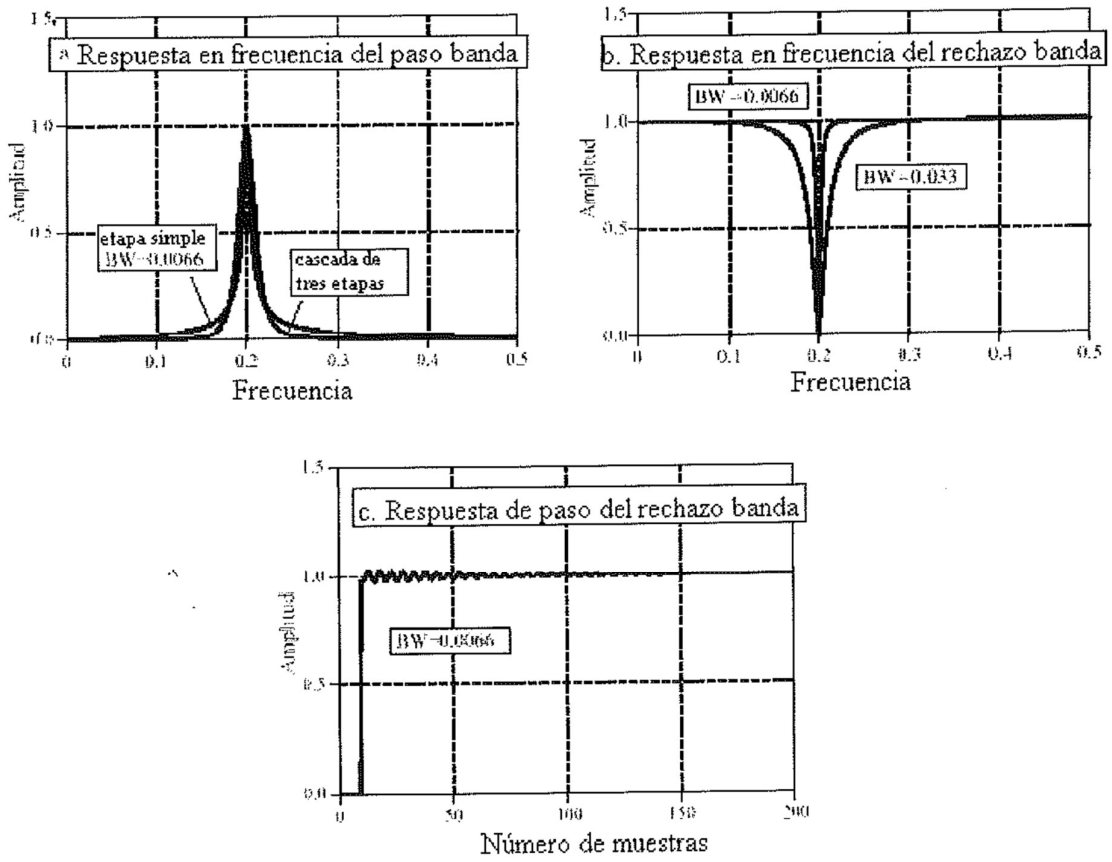


Figura 6.Características de filtros de banda estrecha. La figura (a) y (b) muestran la respuesta en frecuencia de varios filtros de paso banda y de rechazo banda. En (c) se muestra la respuesta en frecuencia de los filtros de rechazo de banda. El filtro de rechazo de banda (notch) es comúnmente usada para eliminar la señal de 50 Hz e interferencias similares de la onda de forma codificada de en el dominio del tiempo.

Dos parámetros deben ser seleccionados antes de usar estas ecuaciones: **F**, la frecuencia central, y **BW**, el ancho de banda (medido en una amplitud de 0.707). Ambos están expresados como un fragmento de la frecuencia de muestreo, y por consiguiente debe estar entre O y 0.5. De estos dos valores especificados, se calculan las variables intermedias: **R** y **K**, y entonces los coeficientes de recursión.

Tal y como se muestra en (a), el filtro del paso de banda tiene relativamente largas colas extendidas desde el pico principal. Esto puede ser mejorado poniendo en cascada varias etapas. Como las ecuaciones del diseño tardan realmente mucho, es más simple implementar esta cascada filtrando la señal varias veces, en vez de tratando de encontrar los coeficientes necesitados para un solo filtro.

La figura (b) demuestra ejemplos del filtro de rechazo de banda. El ancho de banda más estrecho que se puede obtener con precisión simple es aproximadamente 0.0003 de la frecuencia de muestreo. Por encima de esto, la atenuación de la notch disminuirá. La figura (c) muestra la respuesta de paso del filtro de rechazo de banda. Allí es más notable él rebose y la llamada, pero su amplitud es bastante pequeña. Esto permite al filtro eliminar la interferencia de la banda estrecha (50 Hz) con sólo una distorsión menor para la forma de onda del dominio del tiempo.

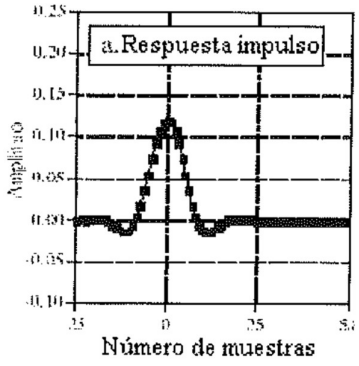
6.4 La respuesta de fase

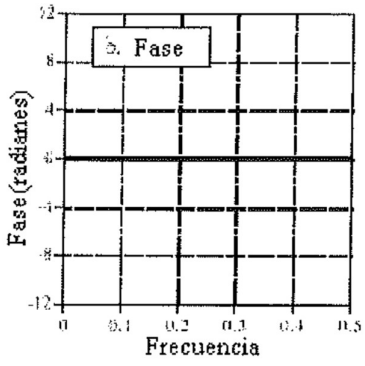
Existen tres tipos de **respuesta de fase** que un filtro puede tener: **Fase cero**, **fase lineal**, y **fase no lineal**. Un ejemplo de cada uno de estos se demuestra en la figura 7. Tal y como se muestra en (a), el filtro de fase de cero es caracterizado por una respuesta de impulso, esto es simétrico alrededor de la muestra cero. La condición real no tiene importancia, ya que las muestras tomadas por la negativa son una imagen idéntica a las muestras positivas.

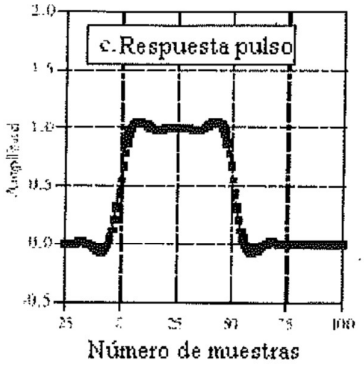
Cuando la transformada de Fourier es tomada de esta forma de onda simétrica, la fase será enteramente cero, tal y como se muestra en (b).

La desventaja del filtro de fase de cero es que requiere del uso de índices negativos, lo cual puede ser inconveniente para trabajar con él. El filtro de fase lineal es un camino alrededor de éste. La respuesta de impulso en (d) es idéntica a la mostrada en (a), con la excepción de haber sido desplazada para usar sólo muestras positivas. La respuesta de impulso es todavía simétrica entre la izquierda y la derecha; Sin embargo, la posición de simetría ha sido desplazada de cero. Este desplazamiento resulta en la fase, (e), siendo una **línea recta**, dando explicación a su nombre: **Fase lineal.** La pendiente de esta línea recta es directamente proporcional a la cantidad del desplazamiento.
Como el desplazamiento en la respuesta de impulso no hace nada pero produce un cambio idéntico en la señal de salida, el filtro de fase lineal es equivalente al filtro de fase cero para la mayoría de usos.

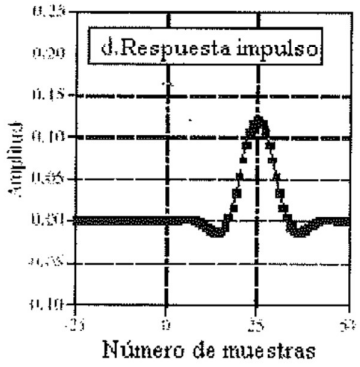
Filtro de Fase Cero

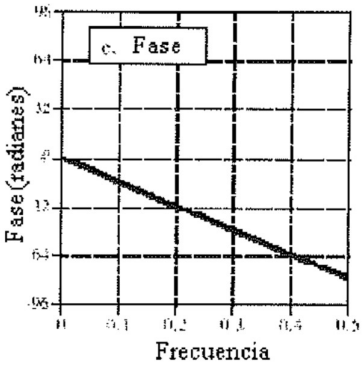


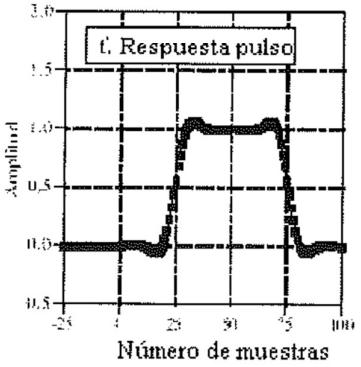




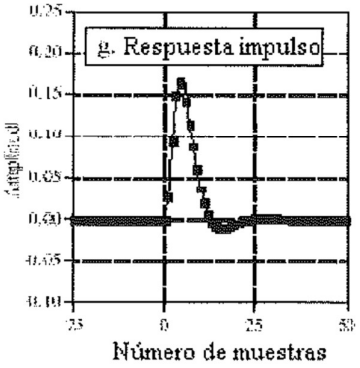
Filtro de Fase Lineal

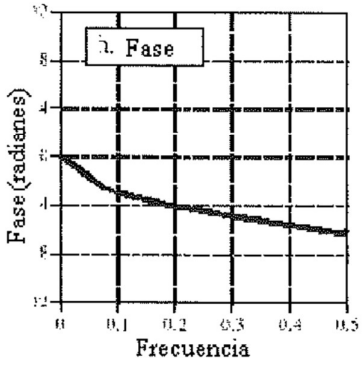






Filtro de Fase No Lineal





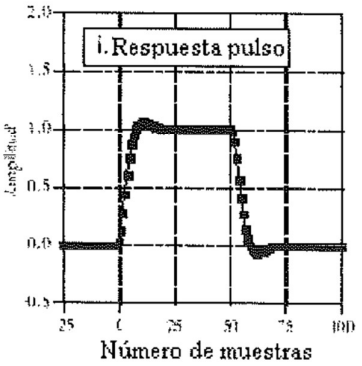


Figura 7. Filtros de fase cero, lineal y no lineal Un filtro de fase cero tiene una respuesta de impulso que tiene simetría entre izquierda y derecha en la muestra cero, tal y como se muestra en (a). De esto resulta una respuesta en frecuencia que tiene una fase enteramente compuesta de ceros, tal y como se muestra en (b). Las respuesta de impulso de fase cero son deseables ya que sus repuesta en frecuencia son simétricas entre arriba y abajo, haciendo los extremos de izquierda y derecha que sean iguales, tal y como se muestra en (c). Los filtros de fase lineal tienen simetría izquierda derecha menos en torno a cero, tal y como se muestra en (d). Esto resulta en una fase que es lineal, una línea recta, tal y como se muestra en (c). La respuesta de pulso de fase lineal, mostrada en (f), tiene todas las ventajas de la respuesta de pulso de fase cero. En comparación, las respuesta en impulso de los filtros de fase no lineal no es simétrica entre la izquierda y la derecha, en (g), y la fase no muestra líneas recta, como en (h). Lo peor es que los extremos izquierdo y derecho de la respuesta de pulso no son iguales, tal y corno se muestra en (i).

La figura (g) demuestra una respuesta de impulso que no es simétrica entre la izquierda y el derecho. Correspondientemente, la fase, (h), no es una línea recta. En otras palabras, tiene una fase **no lineal.** No hay que confundir los términos: Fase **no lineal y lineal** con el concepto de **linealidad** de sistema. Aunque ambos usan la palabra lineal, no están relacionados.

Las figuras (c), (f), y (i) son las **respuestas de pulso** de cada uno de los tres filtros. La respuesta de pulso no es nada más que una respuesta de paso positiva seguida de una respuesta de paso negativa. La respuesta de pulso se usa aquí porque muestra lo que ocurre a los bordes de subida y de bajada de una señal. Aquí está la parte importante: Los filtros de cero y de fase lineal tienen el mismo aspecto en los bordes derecho e izquierdo, mientras los filtros de fase no lineal tienen diferentes. Muchas aplicaciones no pueden tolerar que los bordes izquierdo y derecho tengan aspecto diferente. Un ejemplo es el display de un osciloscopio, en dónde esta diferencia podría ser malinterpretada como un rasgo de la señal que está siendo medida. Otro ejemplo está en el procesado de vídeo.

Es fácil hacer a un filtro FIR (respuesta finita de impulso) tener una fase lineal. Esto es porque la respuesta de impulso (el núcleo del filtro) es enseguida **especificado** en el proceso del diseño. Hacer que el núcleo del filtro tenga la simetría correcta es todo lo que se requiere. Esto no es el caso de los filtros IIR (recursivos), porque los coeficientes de recursión son lo que es especificado, no la respuesta de impulso. La respuesta de impulso de un filtro recursivo no es simétrica entre la izquierda y la derecha, y por consiguiente tiene una **fase no lineal**.

Los circuitos analógicos tienen este mismo problema con la respuesta de fase.

En un circuito compuesto de reóstatos y condensadores, si la entrada siempre ha sido cero, entonces la salida también siempre será cero. Cuando un impulso es aplicado a la entrada, los condensadores rápidamente se cargan con algún valor y entonces comienza exponencial mente a decaer a través de las resistencias. La respuesta de impulso (esto es, La señal de salida) es una combinación de estas diversas exponenciales decadentes. La respuesta de impulso no puede ser simétrica, porque la salida es cero antes del impulso, y la caída exponencial nunca vuelve a tomar un valor de cero otra vez. Los diseñadores de los filtros analógicos acometen este problema con el **filtro Bessel**. El filtro Bessel es diseñado para tener tanta fase lineal como sea posible; Sin embargo, está muy lejos del desempeño de los filtros digitales. La habilidad de proveer una fase lineal **exacta** es una clara ventaja de filtros digitales.

Afortunadamente, hay una forma simple de modificar los filtros recursivos para obtener una **fase cero** La figura 8 demuestra un ejemplo de cómo trabaja éste. La señal de entrada que va ser filtrada es mostrada en (a). La figura (b) muestra la señal después de que haya sido filtrado por un filtro paso bajo de polo simple. Como este es un filtro de fase no lineal, los bordes izquierdo y derecho no son iguales; Son versiones invertidas una de otra. Como previamente se desc1ibe, este filtro recursivo es implementado comenzando en la muestra 0 y operando hacia la muestra 150, calculando cada muestra a través del camino.

Ahora, hay que suponer esto en lugar de trasladarse de la muestra 0 hacia la muestra 150, se empieza en la muestra 150 y se mueve hacia la muestra 0. En otras palabras, cada muestra de la señal de salida es calculada desde las muestras de entrada y de salida a la derecha de la muestra con que se está trabajando. Esto quiere decir que la ecuación de recursión, la 1, es cambiada por:

$$y[n] = a_0 x[n] + a_1 x[n+1] + a_2 x[n+2] + a_3 x[n+3] + \cdots + b_1 y[n+1] + b_2 y[n+2] + b_3 y[n+3] + \cdots$$

Ecuación 9. Ecuación de recursión inversa. Esta es como la ecuación 1, a excepción que la señal es filtrada de derecha a izquierda en vez de izquierda a derecha.

La figura (c) demuestra el resultado de este **filtrado inverso**. Esto es análogo a hacer pasar una señal analógica a través de un circuito electrónico RC mientras se ejecuta el tiempo al **revés**.

Filtrado en dirección contraria no produce ningún beneficio de por sí; La señal filtrada todavía no tiene los bordes derecho e izquierdo iguales. La magia ocurre cuando se combinan los filtrados directos e inversos. La figura (d) resulta de filtrar la señal en la dirección directa y entonces filtrar otra vez en Ja dirección contraria. Esto produce un filtro recursivo de fase de cero. De hecho, cualquier filtro recursivo puede ser convertido a de fase de cero con esta técnica de filtrado bidireccional. El único inconveniente para este desempeño mejorado es un factor de dos en la complejidad de tiempo de ejecución y de programa

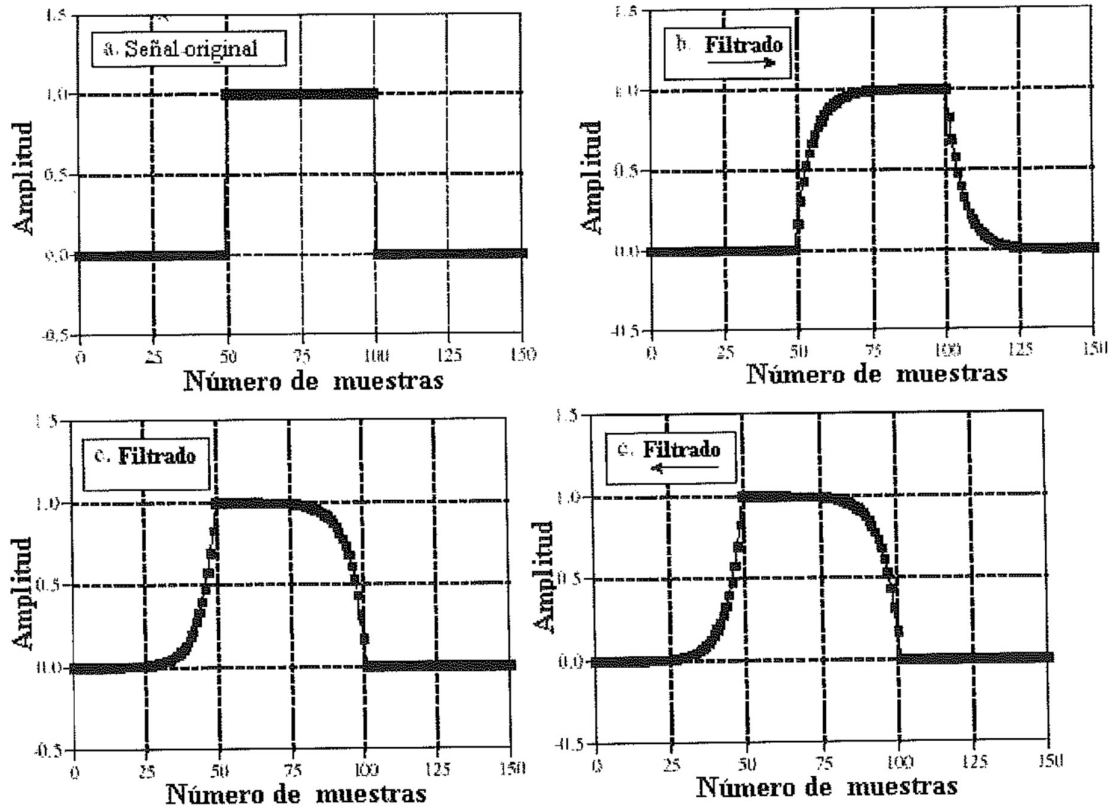


Figura 8. Filtrado recursivo bidireccional. Una señal de pulso rectangular es mostrada en (a). La figura (b) muestra la señal después de que haya sido filtrada con un filtro paso bajo recursivo de polo simple, pasado de derecha a izquierda. En (c) la señal ha sido 'procesada de la misma manera, a excepción del desplazamiento de filtro de izquierda a derecha. La figura (d) muestra la señal después de que haya sido filtrada con ambos filtros. Algún filtro recursivo puede estar hecho usando esta técnica. La magnitud de la respuesta en frecuencia es la misma para cada dirección, mientras que las fases son opuestas en signo. Cuando las dos instrucciones son combinadas, la magnitud llega a ser **cuadrada,** mientras la fase cancela a **cero.** En el dominio del tiempo, esto corresponde a convolucionar la respuesta de impulso original con una versión desplazada de derecha a izquierda de sí misma. Por ejemplo, la respuesta impulso de un solo filtro de paso bajo de un solo punto es una exponencial unilateral. La respuesta de impulso del filtro bidireccional correspondiente es una exponencial unilateral que decae a la derecha, convolucionada con una exponencial unilateral que decae a la izquierda.

Avanzando con el aparato matemático, se obtiene una exponencial bilateral que decae tanto a la izquierda y como a la derecha, con la misma constante de decaída como el filtro original.

Algunas aplicaciones sólo tienen una porción de la señal en la computadora en un tiempo particular, así como los sistemas que alternativamente introducen los datos de entrada y de salida en una base permanente. El filtrado bidireccional puede ser usado en estos casos combinado con el método de adicción de superposición descrita anteriormente. Cada segmento necesitará ser rellenado con ceros en la izquierda y la derecha para permitir la expansión durante el filtrado bidireccional.

6.5 Usando enteros.

El punto flotante de precisión simple es ideal para implementar estos filtros recursivos sencillos. El uso de integrales es posible, pero es mucho más difícil. Hay dos problemas principales. El primero, el error de rebose del número limitado de bits puede degradar la respuesta del filtro, o incluso hacerla inestable. En segundo lugar, los valores fraccionales de los coeficientes de recursión deben ser manipulados con integrales. Una forma a acometer este problema es expresar cada coeficiente como una fracción. Por ejemplo, 0.15 se convierte en 19/128. En lugar de multiplicar a las 0.15, se multiplica por 19 y después se divide por 128. Otra forma es reemplazar las multiplicaciones con tablas.

Por ejemplo, un ADC de 12 bits produce muestras con un valor entre 0 y 4095. En lugar de multiplicar cada muestra por 0.15, se hace a las muestras pasar por una tabla de 4096 muestras de longitud. El valor obtenido de la tabla es igual a 0.15 veces el valor entero de la tabla. Este método es muy rápido, pero ello requiere una memoria adicional; Se necesita una tabla para coeficiente.

7. Filtros Chebyshev.

Los filtros Chebyshev se usan para separar una banda de frecuencias de otra. Aunque no pueden hacer juego con el desempeño del filtro de ventanamiento síncrono, son más que adecuados para muchas aplicaciones. El atributo principal de los filtros Chebyshev es su velocidad, típicamente más que un orden de mayor de magnitud que los que ventanamiento síncrono. Ésta es la causa por la que ellos son llevados por **recursión** en vez de **convolución**. El diseño de estos filtros se basa en una técnica matemática llamado la **transformada-z**. Este apartado presenta la información necesitada para usar filtros Chebyshev sin adentrarse en matemáticas avanzadas.

7.1 Respuestas Chebyshev y Butterworth.

La respuesta Chebyshev es una estrategia matemática para lograr un **rebose** más rápido permitiendo el rizado de la respuesta en frecuencia. Los filtros analógicos y digitales que son usados en este acercamiento se llaman filtros de **Chebyshev**. Por ejemplo, los filtros de Chebyshev son usados para la conversión analógico-a-digital y la digital-a-analógico. Estos filtros son nombrados debido al uso de los polinomios de Chebyshev, desarrollados por el matemático ruso Pafnuti Chebyshev (1821-1894).

En el ejemplo 1 salta a la vista la respuesta de frecuencia de filtros de paso- bajo Chebyshev con rizado en la banda de paso de: 0%, 0.5 % y 20. Como el rizado aumenta (malo), él rebose llega a ser más fino (bueno). La respuesta Chebyshev es un intercambio óptimo entre estos dos parámetros. Cuando el rizado es del 0 %, el filtro es llamado bobina máxima o filtro Butterworth (después S. Butterworth, un ingeniero británico describe esta respuesta en 1930). Una rizado del 0.5 % es a menudo una buena elección para filtros digitales. Esto se combina con la precisión típica y la exactitud de la electrónica analógica que pasa a través de la señal.

Los filtros Chebyshev discutidos en este apartado son llamados filtros de **grado 1**, esto significa que el rizado solo se permite en la banda de paso. En contraste los filtros Chebyshev de **grado 2** solo tienen rizado en la banda de paso. Los filtros de grado 2 se usan rara vez. Existe un diseño importante llamado **filtro elíptico**, el cual tiene rizado tanto en la banda de paso como en la de rechazo.

Los filtros elípticos proveen un rebose más rápido para un número dado de polos, pero son mucho más difíciles de diseñar. El filtro elíptico es frecuentemente la primera elección de los diseñadores profesionales de filtros, en la electrónica analógica y DSPs.

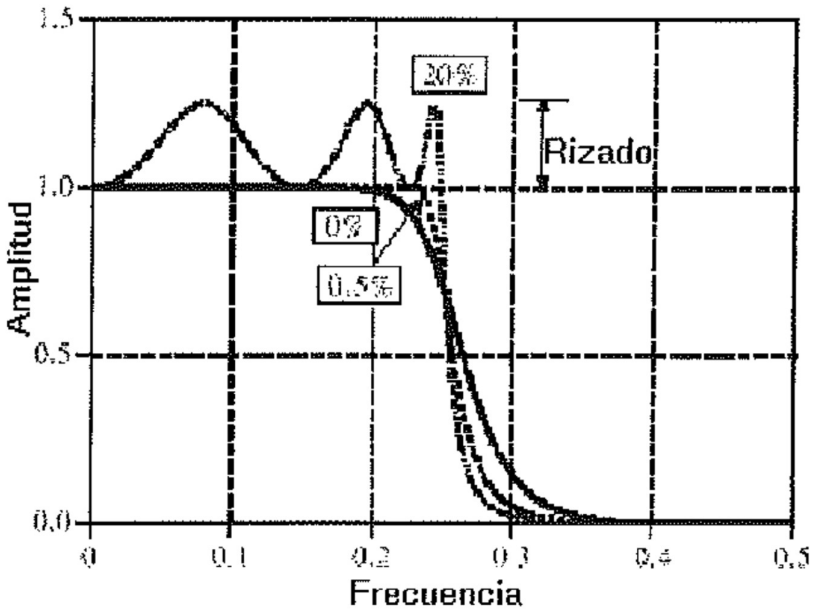


Figura 1. La respuesta Chebyshev. Los filtros Chebyshev proporcionan un rebose rápido permitiendo un rizado en la banda de paso. Cuando el rizado se pone al 0%, es llamado allanamiento máximo ó filtro Butterworth.

Considerando usar un rizado del 0,5% en el diseño; esta banda de paso no llano es tan pequeña que no puede verse en este gráfico, pero él rebose es mucho más rápido que en el Butterworth.

7.2 Diseñando el Filtro.

Se deben seleccionar cuatro parámetros para diseñar un filtro Chebyshev: (1) una respuesta del paso alto ó del paso bajo, (2) la frecuencia de corte, (3) el porcentaje del rizado en la banda de paso, y (4) el número de polos. Hay dos respuestas si se pregunta lo que es un **polo**.

I. La transformada de Laplace y la transformada-z son formas matemáticas de n1ptura de una respuesta de impulso en sinusoidales y exponenciales decadentes. Esto está hecho expresando las características del sistema como un polinomio complejo dividido por otro. Las raíces del numerador son llamadas **ceros**, mientras las raíces del denominador son llamadas **polos**. Como los polos y los ceros pueden ser números complejos, es común decir que tienen una "posición" en el plano complejo. Los sistemas elaborados tienen más polos y ceros que los simples. Los filtros recursivos son diseñados por la primera selección de la posición de los polos y los ceros, y entonces encontrando los coeficientes de recursión apropiados (o los componentes analógicos). Por ejemplo, los filtros Butterworth tienen polos que están situados en un círculo del plano complejo, mientras que en un filtro Chebyshev están situados sobre una elipse.

2. Mientras más polos haya en un filtro, mejor trabaja el filtro.

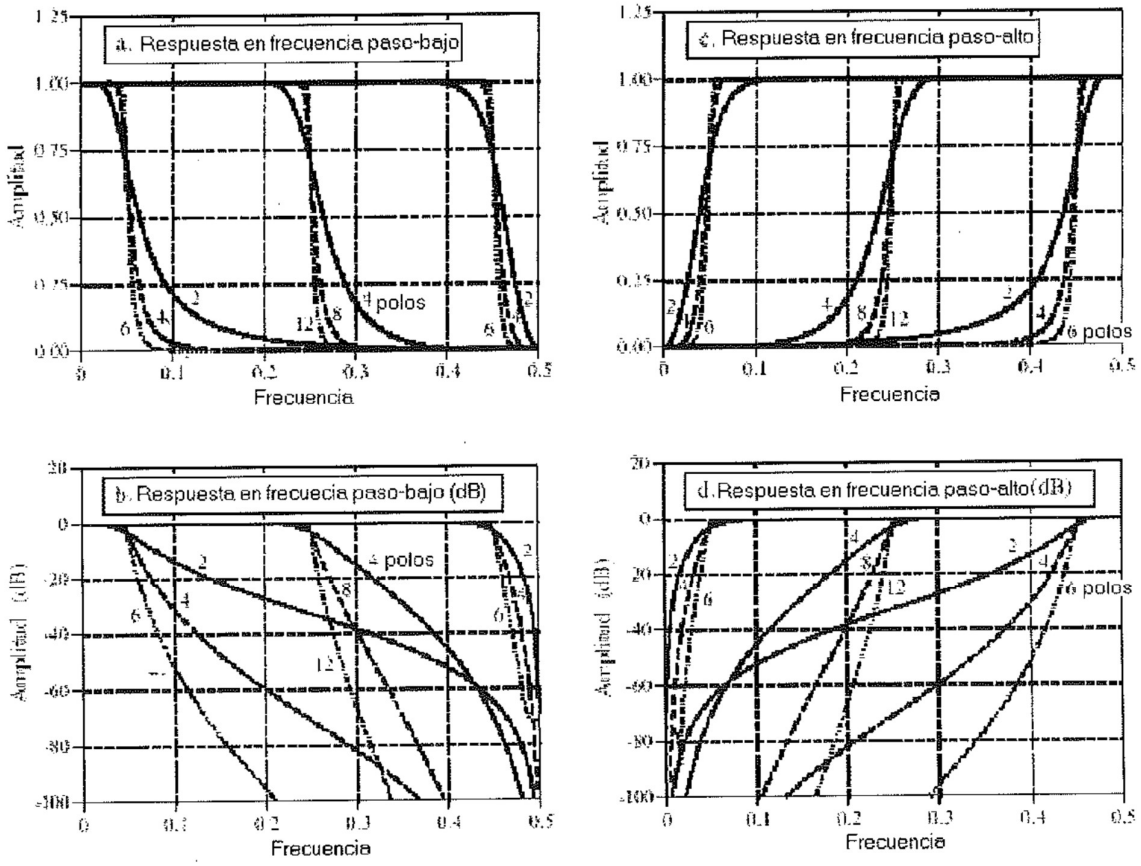


Figura 2. Respuesta en frecuencia Chebyshev. Las figuras (a) y (b) muestran la respuesta en frecuencia de filtros paso-bajo Chebyshev con un rizado del 0,5%, mientras que (c) y (d) muestran las correspondientes respuestas de filtro paso-alto.

El caso es que se pueden usar estos filtros eficazmente sin conocer sus principios matemáticos. El diseño de filtros es una especialidad. En la práctica real, más ingenieros, científicos y programadores piensan en términos de la respuesta 2, que en la **1**.

En el ejemplo 2 se muestra la respuesta en frecuencia de varios filtros Chebyshev con rizado del 0.5 %. Para el método aquí usado, el número de polos debe estar parejo. La frecuencia de corte de cada filtro es medida donde se cruza la amplitud 0.707 (- 3dB). Los filtros con una frecuencia de corte cerca de 0 o 0.5 tienen un rebose más afilados que los filtros en el centro del rango de frecuencia. Por ejemplo, un filtro de dos polos con **F c** = 0.05 tiene aproximadamente el mismo rebose que uno de cuatro polos con **F c** = 0.25.
Menos polos pueden ser usados cerca de 0 y 0.5 debido al ruido de rebose.

Existen dos formas de descubrir los coeficientes de recursión sin usar la transformada-z. Primero, usando una tabla. Las tablas 1 y 2 proveen los coeficientes de recursión para los filtros paso-bajo y paso-alto con rizado del 0.5 %. Si sólo se necesita un diseño rápido y de poca calidad, entonces se copia los coeficientes apropiados para el programa.

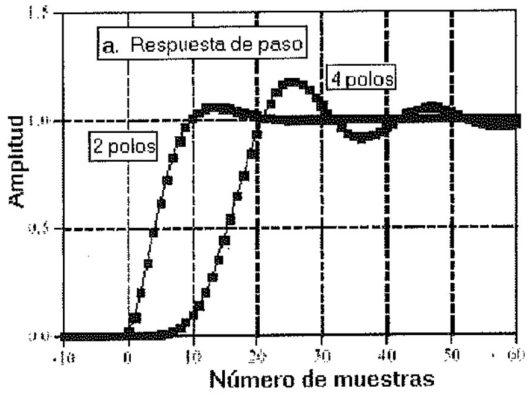
Existen dos problemas con las tablas usadas para el diseño de filtros digitales. El primero, que las tablas tiene una limitada elección de parámetros. Por ejemplo, la tabla 1 sólo provee 12 frecuencias de corte diferentes, un máximo de 6 polos por filtro, y **ninguna** elección para el rizado del paso de banda. El diseño del filtro no puede ser **optimizado,** sin la habilidad de selección de parámetros de un rango de valores continuo. En segundo lugar, los coeficientes deben ser manualmente transferidos de la tabla al programa. La desventaja es que esto es un gran consumidor de tiempo.

En lugar de usar valores tabulados, hay que considerar el incluir una subrutina en el programa que calcule los coeficientes. La buena noticia es que el programa es relativamente simple en su estructura. Después de que los cuatro parámetros del filtro sean introducidos, el programa saca los coeficientes "a" y "b" en los arrays A[ ] y B [ ]. Las malas noticias son que el programa llama a una subrutina complicada. Seis variables introducen la rutina, cinco variables la dejan, y quince variables temporales (más índices) son usadas.

7.3 Sobre impulso de la respuesta de paso.

Los filtros Butterworth y Chebyshev tienen un sobrepaso del 5 al 30 % en sus respuestas de paso, aumentando al aumentar el número de polos. En el ejemplo 3a se muestra la respuesta de paso para dos ejemplos de filtros Chebyshev. La figura (b) muestra algo que es único para los filtros digitales y no tiene comparación en la electrónica analógica: La cantidad de sobre impulso de la respuesta de paso depende en pequeño grado pequeño en la frecuencia de corte del filtro. Los excesivos sobre impulsos de respuesta de paso son resultado de la optimización del filtro Chebyshev para el **dominio de frecuencia** a expensas **del dominio de tiempo.**

En la tabla **1** se observa un filtro Chebyshev de paso-bajo con rizado del 0,5%, y en la tabla 2 se obse1va un filtro. Chebyshev de paso-alto con rizado del 0,5%.



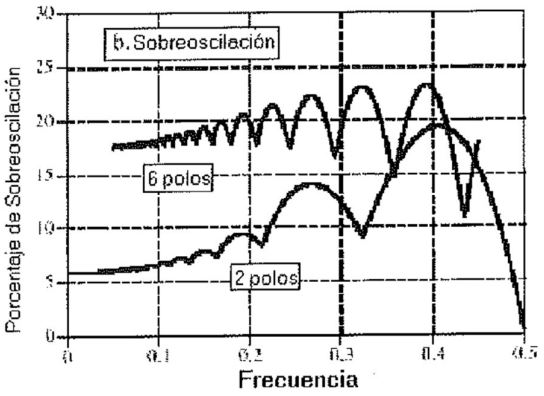


Figura 3. Respuesta de paso Chebyshev. La sobre oscilación en la respuesta de paso de los filtros Chebyshev está entre el 5% y el 30%, dependiendo del número de polos, tal y como se muestra en (a), y la frecuencia de corte, tal y como se muestra en (b). La figura (a) es para una frecuencia de corte 0,05, y puede ser escalado para otras frecuencias de corte.

7.4 La estabilidad

La limitación principal de los filtros digitales realizados por convolución es el **tiempo de** ejecución. Es posible lograr casi cualquier respuesta del filtro. Los filtros recursivos están en oposición. Son muy rápidos; Sin embargo, están limitados en la ejecución. Por ejemplo, considerando 6 polos, un rizado de 0.5%, un filtro paso bajo con una frecuencia de corte de 0.01. Los coeficientes de recursión para este filtro pueden ser obtenidos de la tabla 3:

a0= 1.391351E-10
a1= 8.348109E-10 b1= 5.883343E+00
a2= 2.087027E-09 b2= -l.442798E+01
a3= 2.782703E-09 b3= l.887786E+01
a4= 2.087027E-09 b4= -l.389914E+01
a5= 8.348109E-10 b5= 5.459909E+00
a6= l.391351E-10 b6= -8.939932E-01

Mirando estos coeficientes se observa que los coeficientes "b" tiene un valor absoluto de aproximadamente diez. Usando precisión simple, el ruido de rebose de cada uno de estos números es aproximadamente una diezmillonésima parte del valor, por ejemplo 10^{-6}. Ahora hay que mirar a los coeficientes "a", con un valor de aproximadamente 10^{-9}. Algo está obviamente mal aquí. La contribución de la señal de entrada (por los coeficientes "a") será 1000 veces más pequeño que el ruido de la previamente calculada señal de salida (mediante los coeficientes "b"). Este filtro no operará. Abreviadamente, alrededor el ruido de rebose limita el número de polos que puede ser usado en un filtro. El número real dependerá ligeramente del rizado y si es un filtro paso alto o paso bajo. Los números aproximados para precisión simple son:

Frecuencia de corte	0.02	0.05	0.10	0.25	0.40	0.45	0.48
Polos máximos	4	6	10	20	10	6	4

Tabla 3. Número máximo de polos de precisión simple.

El desempeño del filtro comenzará a disminuir cuando este límite es acometido; La respuesta de paso mostrará mayor sobre impulso, la atenuación de la banda de rechazo será pobre, y la respuesta en frecuencia tendrá un rizado excesivo. Si el filtro es empujado demasiado lejos, ó hay un error en los coeficientes, la salida probablemente oscilará hasta que ocurra una sobrecarga.

Hay dos formas de extender el número máximo de polos que pueden ser usados. La primera, usar doble precisión. Esto requiere usar doble precisión en el cálculo de coeficiente igualmente (incluyendo el valor para la D).

El segundo método es implementar el filtro por etapas. Por ejemplo, un filtro de seis polos se puede poner como una cascada de tres etapas de dos polos cada uno. Sin embargo, el filtro es más estable si es llevado a cabo como las tres etapas originales separadas. Esto requiere conocer los coeficientes "a" y "b" para cada una de las etapas.

8. Comparación de filtros.

Una cuestión muy importante es conocer cuál es el filtro que se ha de escoger. Este apartado es una comparación uno a uno entre filtros. Se va a realizar diferentes comparaciones. En la primera, se comparan los filtros **digitales** con los analógicos para ver cuál es la tecnología más conveniente. En la segunda, se compara el filtro de **ventanamiento síncrono** con el del **Chebyshev** para encontrar el mejor de los filtros de **dominio en frecuencia.** Por último, se compara el desplazamiento corriente con el filtro de polo para encontrar el mejor de los filtros de **dominio en el tiempo.**

8.1 Filtros Analógicos contra Digitales.

La mayoría de señales digitales se originan de la electrónica analógica. Si la señal necesita ser filtrada, cabe preguntarse si debe realizarse un filtrado analógico antes de la digitalización, ó un filtro digital después.

Sea el cometido proveer un filtro paso bajo de 1 kHz. Como filtro analógico se usa un filtro Chebyshev de seis polos con rizado de 0.5 dB (6 %), esto puede estar construido con 3 amplificadores operacionales, 12 resistencias, y 6 condensadores. Como filtro digital se usa un filtro de ventanamiento síncrono. La señal analógica es digitalizada con una tasa de muestreo de 10 kHz, haciendo la frecuencia de corte 0.1 en la escala de la frecuencia digital. La longitud del ventanamiento síncrono esta seleccionado para 129 puntos, proveyendo el mismo (90 % a 10 %) apagado del rebose que el filtro del analógico. En el dibujo 1 se muestra la frecuencia y las respuestas de paso para estos dos filtros.

Comparando estos dos filtros, tal y como se muestra en (a) y (b), se observa que el filtro analógico tiene un rizado del 6 % en la banda de paso, mientras que el filtro digital es perfectamente plano (dentro de 0.02 %). El diseñador de filtros analógicos podría argumentar la opinión que la onda puede ser **seleccionada** en el diseño; Sin embargo, esto se entiende mal. El aplanamiento realizable con filtros analógicos está limitado por la exactitud de sus resistencias y sus condensadores. Aun si una respuesta Butterworth es diseñada (por ejemplo, O % de rizado), los filtros de esta complejidad crearán un rizado residual, quizás del, **1**%. Por otra parte, el aplanamiento realizable con los filtros digitales está primordialmente limitado por el

error del rebose, haciéndoles centenares de veces más planos que los analógicos.

En la respuesta de frecuencia en una escala larga, tal y como se muestra en (c) y (d) se observa que el filtro digital es claramente mejor en él rebose y la **atenuación de la banda de rechazo**. Aunque el desempeño de los filtros analógicos es mejorado añadiendo etapas adicionales, no se puede comparar al filtro digital. Por ejemplo, si se necesita mejorar estos dos parámetros por un factor de 100. Esto se puede realizar con modificaciones simples para el ventanamiento síncrono, pero es virtualmente imposible para los circuitos analógicos.

La respuesta de paso de los dos filtros se demuestra en (e) y (f). La respuesta de paso de los filtros digitales es simétrica entre las porciones inferiores y superiores del paso, esto es, tiene una fase lineal. La respuesta de paso de los filtros analógicos no es simétrica, es decir, tiene una fase no lineal.
Finalmente" los sobre impulsos de los filtros analógicos son aproximadamente un 20 % a un lado del paso, mientras que para el filtro digital es un 10 % a ambos lados del paso.

Aun así, existen todavía muchas aplicaciones en donde los filtros analógicos deberían, o deben, son usados. Esto no está relacionado con el desempeño real del filtro (es decir, lo que entra y lo que sale), sino que con las ventajas generales que los circuitos analógicos tienen sobre las técnicas digitales.
La primera ventaja es la **velocidad**: Los digitales son lentos; Los analógicos son rápidos. Por ejemplo, una computadora personal sólo puede filtrar datos a eso de 10.000 muestras por segundo, usando convolución FFT, y amplificadores operacionales simples pueden funcionar de 100 kHz a 1 MHz, de 10 a 100 veces más rápido que el sistema digital.

La segunda ventaja es el **rango dinámico**. El **rango dinámico** de la amplitud es la proporción entre las señales más largas que pueden por un sistema, y el ruido inherente del sistema. Por ejemplo, un ADC de 12 bits tiene un nivel de saturación de 4095, y un ruido de cuantificación de rms de 0.29 números digitales, para un rango dinámico de casi 14000. En contraste, un amplificador operacional estándar tiene una tensión de saturación de casi 20 voltios y un ruido interno de casi 2 microvoltios, para un rango dinámico de unos diez millones.

Otra ventaja es la frecuencia de rango dinámico. Por ejemplo, es fácil de diseñar un circuito de amplificador operacional para manipular simultáneamente frecuencias entre 0.01 Hz y 100 kHz (siete décadas). Cuando esto es probado con un sistema digital, la computadora se satura de datos. Por ejemplo, el muestreo en 200 kHz, lleva 20 millones de puntos a captar un ciclo completo a 0.01 Hz. Se puede ver que la respuesta en frecuencia de los filtros digitales es casi siempre expuesta en una escala **lineal** de frecuencia, mientras que los filtros analógicos son usualmente exhibidos con una frecuencia **logarítmica.** Esto es debido a la que los filtros digitales necesitan una escala lineal para demostrar su excepcional desempeño de filtrado, mientras que los filtros analógicos necesitan la escala logarítmica para demostrar su enorme rango dinámico.

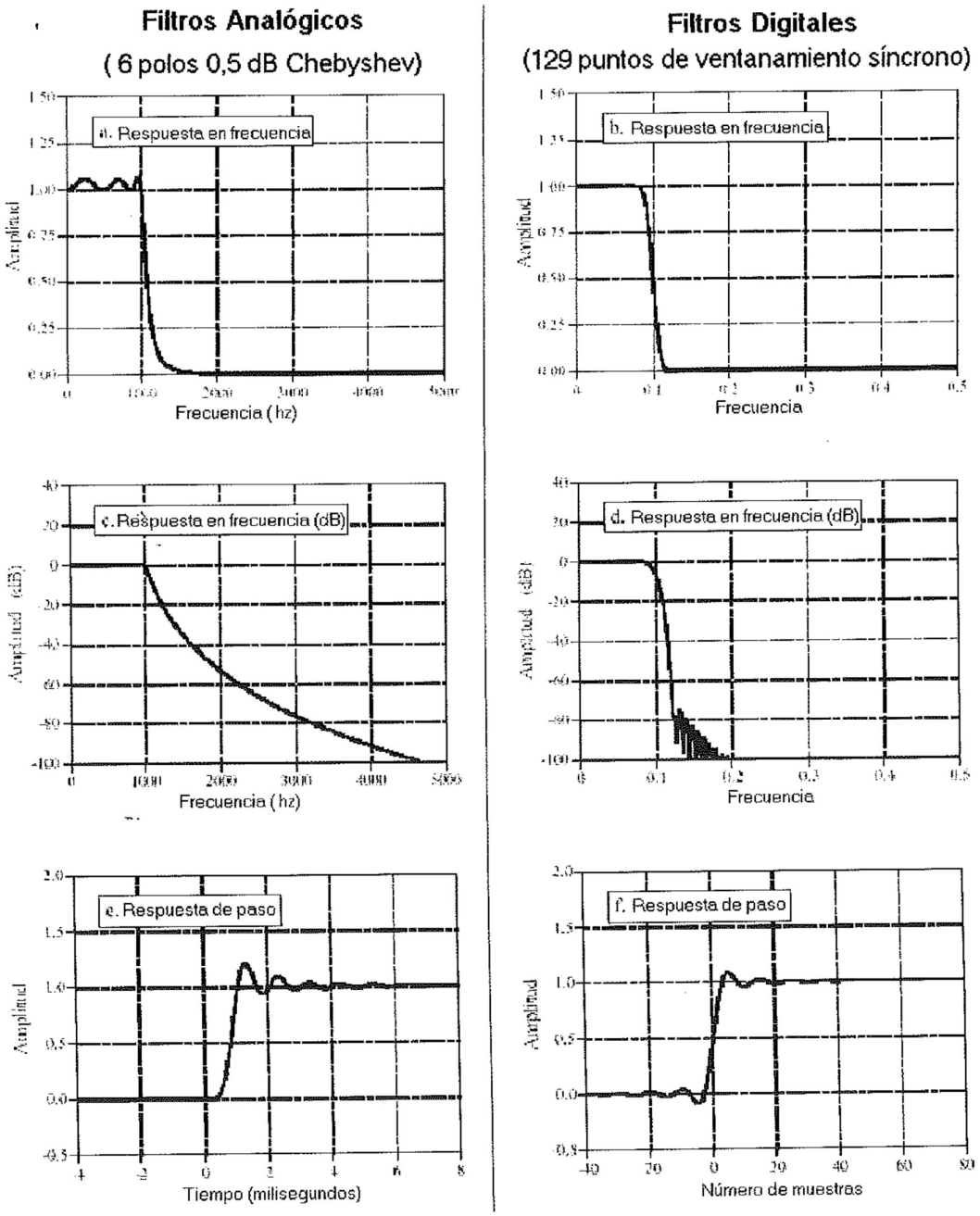


Figura I. Comparación de filtros analógicos y digitales. Los filtros digitales tienen mejor comportamiento en muchas áreas, tales como: Rizado de la banda de paso, (a) frente a (b), rebose y atenuación de la banda de rechazo, (c) frente a (d), y simetría de la respuesta de paso, (e) frente a (f). El filtro digital en este ejemplo tiene una frecuencia de corte de 0,1 del ratio de muestreo de 10 kHz. Esto provee una comparación con la frecuencia de corte de 1 kHz del filtro analógico.

8.2 Filtros de Ventana m iento síncrono contra Chebyshev.

Ambos filtros son diseñados para separar una banda de frecuencias de otra. El ventanamiento síncrono es un filtro FIR implementado por convolución, mientras el Chebyshev es un filtro IIR ejecutado por recursión. ¿Cuál es el mejor filtro digital en el dominio de frecuencia?.

El filtro recursivo es una onda de 0.5 % de rizado, filtro paso bajo Chebyshev de 6 polos. Una comparación justa es complicada ya que la respuesta en frecuencia de Chebyshev se altera con la frecuencia de corte. Si se usaremos una frecuencia de corte de 0.2, y se selecciona un núcleo del filtro del ventanamiento síncrono para tener 51 puntos. Esto hace que ambos filtros tengan el mismo rebose del 90 % al 10 %, tal y como se muestra en la figura 2 (a).

El filtro recursivo tiene un rizado del 0.5 % en la banda de paso, mientras que el ventanamiento síncrono es plano. Sin embargo, fácilmente se podría colocar el rizado del filtro recursivo al 0 % si fuese necesario. La figura 2b muestra que el ventanamiento síncrono tiene una atenuación de la banda de rechazo muy mejor que el Chebyshev.

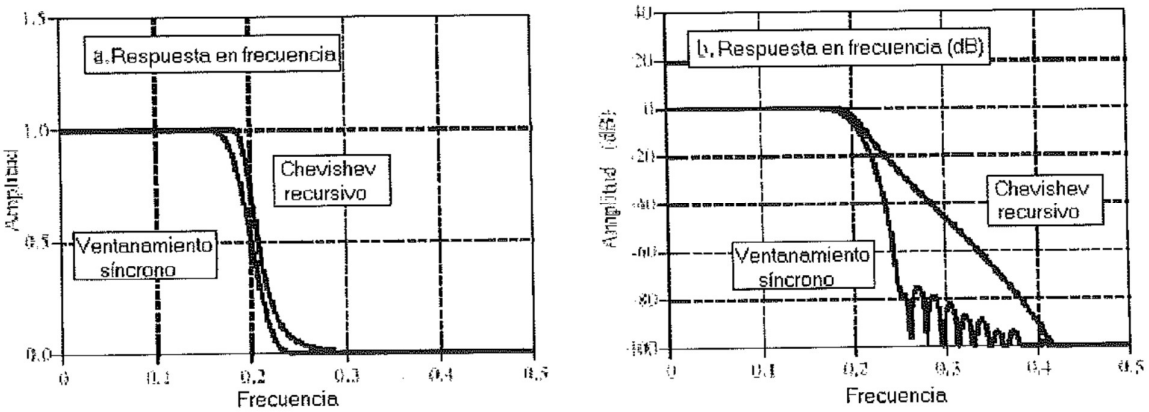


Figura 2. Ventanamiento síncrono y respuestas de frecuencia Chebyshev. Las respuestas en frecuencia son mostradas para filtros de ventanamiento síncrono de 51 puntos y un filtro recursivo Chebyshev con rizado del 0,5% de 6 polos. El ventanamiento síncrono tiene mejor atenuación de la banda de rechazo, pero también trabajará en

aplicaciones de comportamiento moderadas. La frecuencia de corte de ambos filtros es 0,2, medida en una amplitud del 0,5 para el ventanamiento síncrono, y 0,707 para el recursivo.

En el dibujo 3 se muestra la respuesta de paso de los dos filtros. Ambos son malos, como se debería esperar para filtros de dominio en frecuencia. El filtro recursivo tiene una fase no lineal, pero esto puede corregirse con filtrado bidireccional.

Hasta ahora, no hay mucha diferencia entre estos dos filtros; Uno u otro trabajará cuando el desempeño moderado sea necesario. La diferencia esencial está en dos asuntos críticos: El **desempeño máximo** y **la velocidad**. El ventanamiento síncrono es una central eléctrica, mientras el Chebyshev es rápido y ágil. Suponiendo que se tiene un problema realmente difícil de separación de frecuencia, esto es, se necesita aislar en señal de 100 milivoltios en 61 Hz que está en una línea de conducción eléctrica de 120 voltios y 60 Hz.

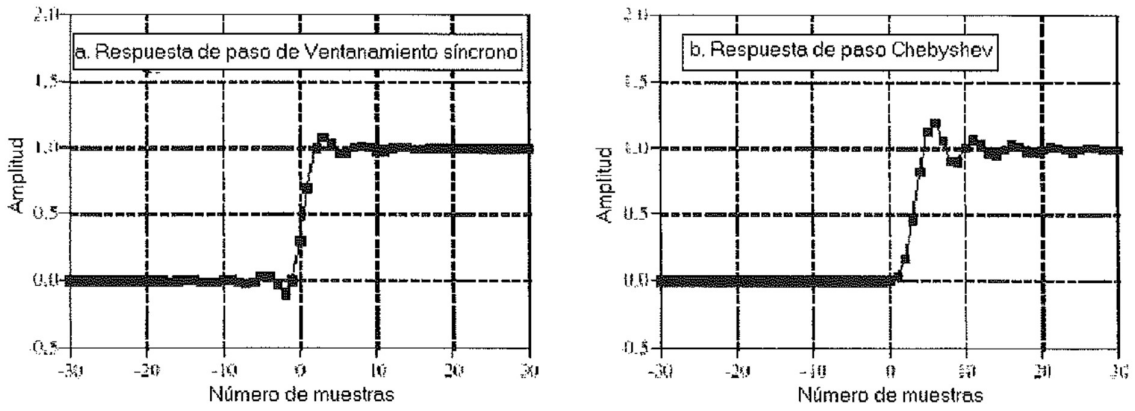


Figura 3. Respuestas de paso de Ventanamiento síncrono y Chebyshev. Las respuestas de paso se muestran para un filtro de Ventanamiento síncrono de 51 puntos y un filtro recursivo Chebyshev con rizado del 0,5% de 6 polos. Cada uno de estos filtros tiene una frecuencia de corte de 0,2. El Ventanamiento síncrono tiene una respuesta de paso ligeramente mejor porque tiene menos sobre oscilación y una fase cero.

En, el dibujo 4 se muestra cómo se asemejan estos dos filtros cuando se necesita un desempeño máximo. El filtro recursivo es un Chebyshev de 6 polos con rizado del 0.5 %. Éste es el número máximo de polos que pueden ser usados a una frecuencia de corte de 0.05 con precisión simple. El ventanamiento síncrono usa un núcleo de filtro con 1001 puntos, formada al Convolucionar con un núcleo de filtro de ventanamiento síncrono de 501 puntos consigo mismo. Tal y como se ha visto anteriormente, esto provee una atenuación mayor de la banda de rechazo.

Estos dos filtros se asemejan cuando es necesario el desempeño máximo. El ventanamiento síncrono es mejor que el Chebyshev. Aun si el filtro recursivo estuviera perfeccionado (más polos, implementación multietapa, doble precisión, etc.), no es ninguna mejora para el desempeño del FIR. Existen fuertes límites en el desempeño máximo que los filtros recursivos pueden proveer. El ventanamiento síncrono, en contraste, puede ser excitado a niveles muy grandes. Esto es, proveer a los resultados a los que se esperan. Que trae a colación el segundo asunto crítico: **La velocidad.**

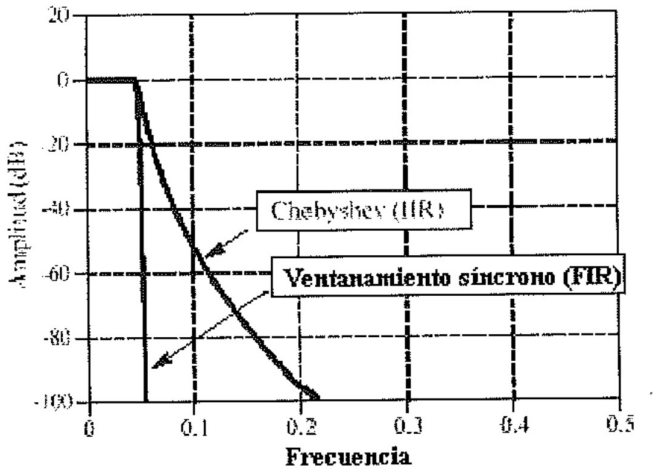


Figura 4. Máxima comportamiento de filtros FIR e IIR. La respuesta en frecuencia del ventanamiento síncrono puede ser virtualmente cualquier forma requerida necesitada, mientras que el filtro recursivo Chebyshev es muy limitado. Este gráfico compara la respuesta en frecuencia de un filtro recursivo Chebyshev de seis polos con un filtro de ventanamiento síncrono de 1001 puntos.

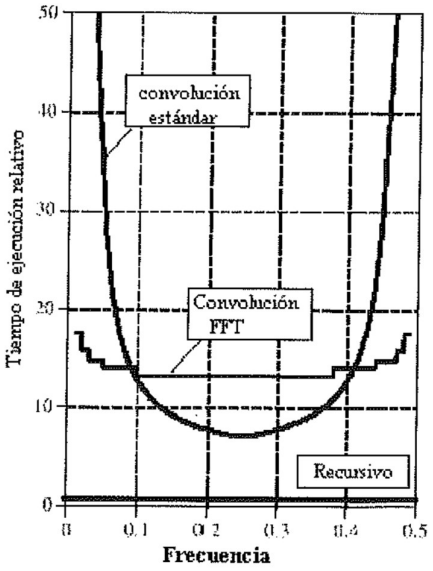


Figura 5. Comparación entre velocidades de ejecución de FIR y IIR. Estas curvas muestran los tiempos de ejecución relativos para un filtro de ventanamiento síncrono comparado con un filtro recursivo Chebyshev de seis polos. Las curvas son mostradas para implementar los filtros FIR tanto con el algoritmo de convolución estándar como el FFT. El tiempo de ejecución del ventanamiento síncrono se incrementa en frecuencias altas y bajas porque el filtro del núcleo debe ser mayor para guardarse con el mayor comportamiento de los filtros recursivos a estas frecuencias. En general, los filtros IIR tienen una orden de magnitud más rápida que los filtros FIR de comportamiento comparable.

En el dibujo 5 se muestra cuánto más lento es el ventanamiento síncrono para ejecutar, comparado con un filtro recursivo de seis polos. Como el filtro recursivo tiene un rebose más rápido a frecuencias bajas y altas, la longitud del núcleo de ventanamiento síncrono debe estar hecha más larga para ajustar con el desempeño. Esto da explicación sobre el aumento del tiempo de ejecución para las frecuencias del ventanamiento síncrono cercanas a 0 y 0.5.

8.3 Desplazamiento común contra polo simple.

Esta comparación es entre filtros en el dominio del tiempo. El filtro de desplazamiento común es un filtro de nueve puntos. El otro es un filtro recursivo de polo simple que usa técnica bidireccional. Para lograr una respuesta comparable de frecuencia, el filtro de polo simple usará una caída de muestra a muestra de x = 0.70. La respuesta en frecuencia de cada filtro es demostrada en la figura 6. Ninguna es muy impresionante, ya que la separación en frecuencia no es para lo que estos filtros son usados.

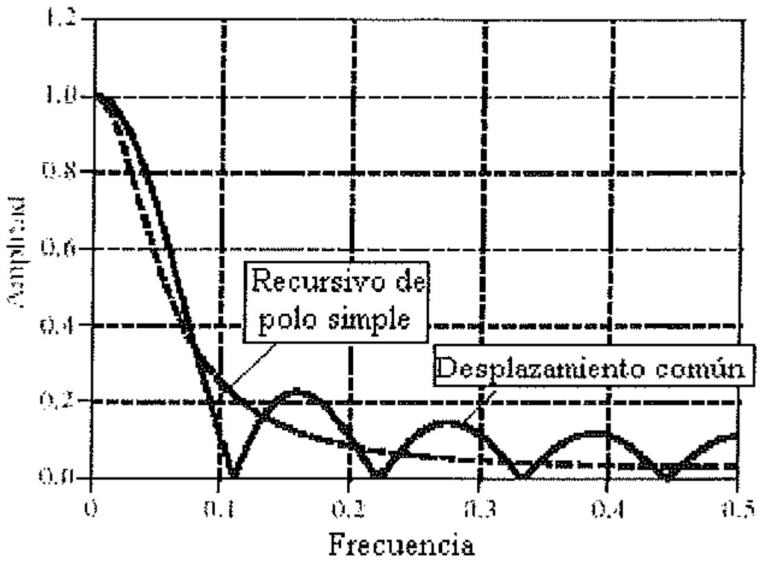


Figura 6. Respuestas en frecuencia de desplazamiento común y polo simple. Ambos filtros tienen una respuesta en frecuencia pobre, tal y como se debe esperar para filtros en el dominio del tiempo.

En el dibujo 7 se muestra las respuestas de paso de los filtros. En (a), la respuesta de paso de desplazamiento común es una línea recta, la forma más rápida de desplazamiento de un nivel a otro. En (b), la respuesta de paso del filtro recursivo es más lisa, lo cual puede ser mejor para algunas aplicaciones.

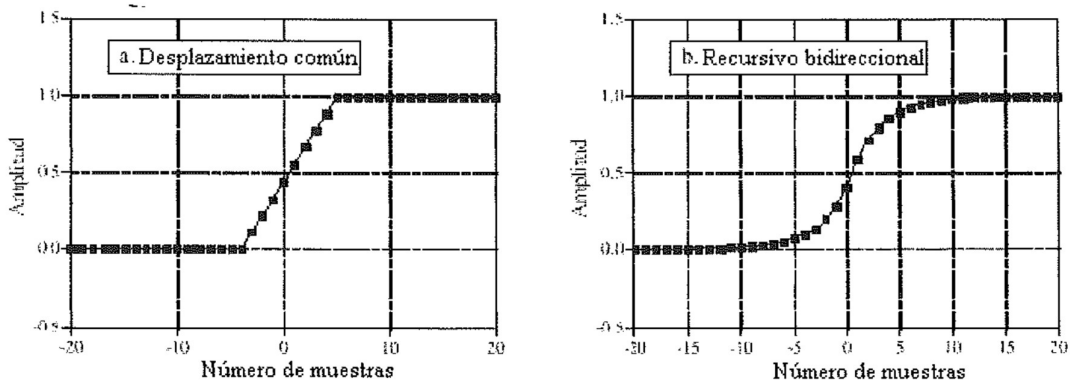


Figura 7. Respuesta de paso del filtro de desplazamiento común y el bidireccional de polo simple. La respuesta de paso del desplazamiento común ocurre sobre un número pequeño de muestras, mientras que la respuesta de paso del filtro de polo simple es más plana.

Estos filtros están muy igualados en términos del desempeño y a menudo la elección entre los dos está debida a una preferencia personal. Sin embargo, existen dos casos en donde un filtro es superior a otro. Estos se basan en el intercambio entre el **tiempo de desarrollo** y el de **ejecución.** En primer lugar, se quiere disminuir el tiempo de desarrollo y se está dispuesto a aceptar un filtro más lento. Por ejemplo, se podría tener la necesidad de filtrar algunos miles puntos en el mismo instante. Como el programa entero se ejecuta en sólo algunos segundos, no tiene sentido gastar tiempo optimización el algoritmo. Es casi seguro que el punto flotante será usado. La elección es usar el filtro de desplazamiento común ejecutado por convolución, o un filtro recursivo de polo simple. Será ligeramente más fácil programar y modificar, y se ejecutará mucho más rápido.

El segundo caso es justo el sentido contrario; El filtro debe funcionar tan rápido como sea posible y se está dispuesto a gastar un tiempo adicional de desarrollo para obtenerlo. Por ejemplo, este filtro podría ser una parte de un producto comercial, con el potencial para que funcione millones de veces. Probablemente se usen enteros para la velocidad posible más alta. La elección del filtro es el desplazamiento común ejecutado por **recursión,** ó el filtro recursivo de polo simple implementado con tablas ó matemáticas de entero. El mejor es el filtro de desplazamiento común, el cual se ejecutará más rápido y no será susceptible para los problemas de desarrollo y ejecución de la aritmética de entero.

B DSP TMS320C50.

B.1 STARTER KIT DEL DSP TMS320C50.

1. Licencia del Hardware y del software del DSK.

El fabricante garantiza la funcionalidad del circuito del DSK habiendo probado y examinado en fábrica el hardware del circuito, no obstante se recomienda comprobar la tarjeta antes de realizar cualquier cambio. El realizar cambios va a invalidar cualquier clase de garantía.

2. Introducción al KIT de los diseñadores del C50.

-Descripción del DSP.

El kit de desarrollo del DSK C50 está diseñado para ser utilizado desde un IBM PC/ AT o compatible. La interfaz utilizada está conectada a través de un puente de comunicaciones serie Rs- 232.
En las especificaciones del hardware está previsto que se pueda aumentar el módulo del DSK con memoria adicional.

-Componentes del kit.

El kit debe contener los siguientes materiales:

- La tarjeta del circuito del DSK.
- El procesador de TMS320C50 DSP.

Si se vuelve a apagar la interrupción global se permite que el pedacito o desenmascare INT2 de la característica runtime del alto no trabaje. Asimismo, sí se reescribe la tabla del vector para la TRAMA o INT2 el núcleo de la depuración no podrá funcionar correctamente. De todo lo almacenado del registro del bloque B2 de la CPU, las primeras 384 palabras están reservadas por el núcleo de la depuración de comunicaciones de SARAM, es por ello que los vectores de la interrupción del DSK han determinado que se comience en Ox800. Esto significa que el valor de IPTR en el registro de PMST es 1, y este valor no se puede modificar.

El programa de DSK5D restaurará el REAJUSTE, INT2, los vectores de la interrupción de la TRAMA por lo que no hay que preocuparse por sobrescribir estos vectores.

El núcleo del programa empieza en 0x840, cosa que hay que tener en cuenta para que los vectores de interrupción no sobrescriban el programa en esta dirección.

El programa debe comenzar después de Ox980. Se recomienda salir de 0xa00.

-Tabla del vector de interrupción

Interrupción RINT del contador de tiempo: B RECIBE; 0A
El puerto serie recibe la interrupción XINT: B TRANX·; 0C;
El puerto serie transmite la interrupción TRNT: B TREC·; 0E';
El puerto serie de TDM recibe la interrupción TXNT: B TTRANX·; 10;
El puerto serie de TDM transmite interrupción INT4: B ISR4; 12; INT4 externo;

Como ya hemos mencionado anteriormente el código del Núcleo del programa viene entre (00840H - 00980H)

-Comandos del DSK5D y del menú de ayuda

Se pueden ver dentro de DSK5D pulsando ' H ' (ayuda).
Esta lista se puede también enviar a un archivo pulsando ' F ' (archivo) en el menú de ayuda. Lo que sigue es el listado de salida.

8. DSP STARTER KIT DEBUGGER (DEPURADOR)

H: Exhibe pantalla de ayuda
Bxxxxx: Selecciona los baudios: 4800, 9600, 19200 por defecto, 38400, o
57600.
El COMx: Puerto serie: X = **1** por defecto o x = 2
Exxxx: Define el punto de entrada, dónde el xxxx es la dirección en hexadecimal.
I: Selecciona nivel de lógica para DTR-RESET (el incumplimiento > inverso) L: Selecciona el largo de la pantalla del EGA/ VGA (43 o 50)
S: Selecciona el largo predeterminado (25) de la pantalla

-Comandos

F1 Información de Ayuda
F2 Imprime pantalla para ' screen.sm
F3 Directorio de Despliegue
F4 sin uso
F5 ejecuta para breakpoint - > de la misma forma como XG
F6 sin uso
F7 sin uso
F8 ejecuta un paso, al igual FI O, XS, Barra de espacio, return
F9 sin uso
F11 y ShiftFI Ventana de ensamblado
FI2 y ShiftF2 Trace ON/OFF

BA Añade Breakpoint
BD Borra Breakpoint
BE Habilita/ Deshabilita Breakpoint
BL Lista de Breakpoint
C.. Copia/ Mueve
CDD Copia Bloque de Datos a otra área de Datos
CDP Copia Bloque de Datos a área de programa
CPD Copia Bloque de programa a área de Datos
CPP Copia Bloque de programa a otra área de programas

DD Muestra memoria Data DB Exhibe los Breakpoints
DF. Despliegue Format: Ver subórdenes
DFB Muestra Datos en formato grande, (exp = 16, mantisa = 32/Q30)

DFP Muestra Datos en formato double
DFE Muestra Datos en formato abruptamente flotador (exp = 16, mant = 16/Ql4)
DFF Muestra Datos en flotador del formato
DFI Muestra Datos en el entero del formato DFL Muestra Datos en formato largo
DFO Muestra Datos en octal del formato
DFP Visualiza datos en formato packed string
DFQ Visualiza datos en formato firmó a Ql 5
DFS Muestra Datos en format stimg
DFU Muestra un Dato en formato el entero sin integrar
DFT Exhibe un Dato en formato el flotador largo (exp= 32, mant= 64/Q62) DFX Muestra Datos en hexadecimal del formato
DM. La ventana de despliegue de Memoria: Ver subórdenes
DMA Mues,,tra memoria en formato: Dirección determinada
DMB Muestra memoria en formato: Grande (exp = 16, mantisa 32/Q30)
DMD Muestra memoria en formato: El doble (IEEE)
DME Muestra memoria en formato: Ponga en sh01t float (exp = 16, mantisa 16/Q14)
DMF Muestra memoria en formato: Float (IEEE) DMI Muestra memoria en formato: El entero DML Muestra memoria en formato: Long
DMO Muestra memoria en formato: El octal DMQ Muestra memoria en formato: Ql 5
DMU Muestra memoria en formato: El int sin firmar
DMT Muestra memoria en formato: El flotador largo (exp = 32, mant 64/Q62)
DMX Muestra memoria en formato: Para hexadecimal DP Muestra La memoria de Programa
DS Muestra información de registro Status DV Muestra la Versión

FD Rellena la memoria de datos con un valor
FP Rellena la memoria de datos con un valor

Help Despliegue de ayuda

I Inicializa el punto de entrada del programa
L.. Los datos de carga (el programa, los datos formateados de un archivo) LC Carga archivo Coff: El formato común del archivo del objeto

LD, Carga archivo DSK: Formato de archivo DSK
LP Carga el formato del Programa: 4 dígitos hex por línea LF. Carga formato
LFB Carga en formato Bfloat: 1 Bfloat por línea
LFD Carga en formato Double: 1 número del doble IEEE por línea
LFE Carga en formato Efloat: 1 Efloat por línea
LFI Carga en Formato entero: 1 int por línea
LFL Carga en formato largo: 8 dígitos hex por línea
LFF Carga Flotador Format: 1 número float IEEE por línea

LFP Carga en formato packed charter string
LFQ Carga en formato Ql5: Un número Ql 5 por línea
LFS Carga en formato Char string: 1 Carácter por la palabra
LFT Carga en formato Tfloat: 1 float largo por línea
LFU Carga ,,formato sin firmar: 1 entero sin firmar por línea
LFX Carga en formato Hex: 4 dígitos hex por línea

MR Modifica Registro los del 320C50: PC, ACCU, ARx, etc.
MP Modifica la memoria de programa
MD Modifica la memoria de datos
Q Salir del programa (o ctrl Q)
P Carga el contador del programa: Hecho de despliegue ModifyRegister
R Resetea el 320c50
S.. Salva .el programa, los datos de un archivo
SD Salva la memoria de datos a archivo SF Guardando el Formato
SP Guarda memoria de Programa a archivo: 4 dígitos hex por línea
SFB Salva a Formato Bfloat: 1 float grande por línea
SFI Salva a Formato entero: 1 int por línea
SFD Salva a Formato Double: 1 double IEEE por línea
SFE Salva a Formato Efloat: 1 float especial por línea
SFF Salva a Formato Float: 1 número float IEEE por línea
SFL Salva a Formato largo: 8 dígitos hex por línea
SFP Salva a Formato de Paquete de cadena de caracteres
SFQ Salva a Formato Ql5: 1 número Ql 5 por línea
SFS Salva a cadena de caracteres Format: 1 cadena de caracteres por la palabra
SFT Salva a Formato Tfloat: 1 float largo por línea
SFU Salva a Formato entero finnar: 1 int por línea
SFX Salva a Formato Hex: 4 dígitos hex por línea

V, Verifica Flag ON/ OFF - por defecto = ON para (Carga...)

W.. Definición de ventana de reloj
WA Ventana del reloj: Definición de la dirección
WD Observa ventana y suprime el número del reloj
WM Observa ventana y modifica la dirección de reloj
WF Formato de la ventana: Ver subórdenes
WFB Formato de la ventana: Grande (exp = 16, mant 32/Q30)
WFD Formato de la ventana: Doble
WFE Formato de la ventana: Corto float (exp = 16, mant 16/Ql4)
WFF Formato de la ventana: Float
WFI Formato de la ventana: Entero
WFP Formato de la ventana: packed string
WFQ Formato de Ja ventana: Q15 (signo. fragmento)
WFS Formato de la ventana: stirng
WFU Formato de la ventana: Entero sin firmar
WFT Formato de la ventana: Largo (exp = 32, mant 64/Q62)

XA ejecuta y va a dirección definida
XC ejecuta con llamada a una función
XF ejecuta una función: CALL o CALA
XG ejecuta el ir a Breakpoint
XL ejecuta y Linea: A ramas como' call ' para una dirección
XQ ejecuta y abandona el depurador
XN ejecuta ' n ' pasos simples
XR ejecuta y coITe libre (ignorando cambiar a XF)
XS ejecuta un solo paso como: F8, FlO, baITa de espacio, return

-Formatos **de punto** flotante:

Doble IEEE de 64 bits formato de punto flotante doble estándar
Bfloat ' float grande / 16 bit de exponente + 32 bits de exponente de mantisa
: 2 el complemento / signed mantisa = Q30
.Tfloat ' float largo / 32 bit de exponente + 64 bits de exponente de mantisa : 2 el complemento / signed mantisa = Q62
Efloat float realzado / 16 bit de exponente + 16 bits de exponente de mantisa: 2 el complemento / signed mantisa Ql4

-Definición de Registros:

PC Contador del Programa: 16 bits
ACCU Acumulador: 32 bit con acarreo en STI
ACCB Buffer del acumulador : 32 bit con almacenamiento temporal en ACCU
PREG Registro del producto : 32 bit para multiplicaciones de 16* 16 bits TRGO Multiplicando temporal: 16 bit para las instrucciones de multiplicación e instrucciones especiales
TRG 1 Registros temporales 1: 5 bit para el cambio dinámico
TRG2 Registros temporales 2: 4 bit para bit del puntero en testeo de bits
Ari Registros Auxiliares: 16 bit con i = 0..7 usados como contador y puntero
STO Registro de estado O: 16 bit
ST1 Registro de estado 1: 16 bit
PMST Registro de estado: 16 bit
STCKi Registro de empilamamiento:16 bit con n = 0..7 usado como empilador
Hardware. El depurador debe usar un empilador que este a nivel de sus aplicaciones.
DRR Registro de recepción datos en dirección 0: 16 bit para puerto serie
DXR Registro de transmisión datos en dirección 1: 16 bit para el puerto srie
TIM Registro cronometrador (Timer) en dirección 24: 16 bit
PRD Registro de Periodo en dirección 25: 16 bit
IMSK Registro de interrupción del enmascaramiento en dirección 4: 6 bits para enmascarar 6 interrupciones
GREG Registro Global en dirección 5: 8 bits para definir la memoria de datos como global
SPC Registro de control de puerto serie en dirección 22
ARP STO: Puntero auxiliar de registro: 3 bits
INUN STO: modo de interrupción (Habilita entero global): 1 bit DP STO: Puntero de la página de datos: 9 bits
ARB STI : Buffer del puntero del Registro auxiliar: 3 bits
CNF STI: Configuración interna Prog/ Datos: 2 bits
PM STI : PREG para el modo de cambio del ACCU: 2 bits

C STI : Bit de acarreo: 1 bit
TC STI: Test/Control: 1 bit
HM STI : Selección modo sostenido: 1 bit
TXM SPC: Bit del modo FSX: 1 bit
FO SPC: Control del puerto serie (modo 8/16 bit): 1 bit

Hay que tener en cuenta lo siguiente:

1) IMR = 4 (INT2) y EINT están listos para el depurado (debugger) y así posibilitar que se pueda ejecutar un programar. No hay que sobrescribir el bit IMR = 4 y usar la instrucción lógica apropiada para interrumpir el enmascaramiento. Tampoco hay que cambiarlas posiciones Ox804 y Ox805, ya que están reservadas para INT2. Si otro bit IMR de los 4 es activado, entonces esta Intem1pción es ejecutada durante pasos simples (singel steps)

Si un breakpoint es colocado en un vector de interrupción (0x802 ...) y entonces es ejecutado con pasos simples encajará en la estructura de las interrupciones ya que EINT es ejecutado automáticamente.

2) En el Bloque B2 (la memoria de datos 0x60 - 0x7f) está reservada para el uso del depurador (debugger), luego no hay que sobrescribir estas pos1c10nes.

3) La semilla del debugger está ubicada de Ox840-0x980 en SARAM configurado como la memoria Programa y Datos, por lo que no hay que modificar los bits de la RAM y OVLY.

4) La instrucción TRAP la posición de memoria Ox822 (el vector trap) es reservada para el debugger

El depurador(debugger) ubica el vector de interrupción en Ox800 y pone IPTR = 1, por lo que no hay que modificar el valor de ejecución del 1IPTR en los interruptores usados, los cuales requieren de los vectores adicionales tienen que ser colocados dentro del espacio de direcciones Ox800-0x83f de la siguiente forma:

-Mapa de memoria:

Ox802-0x803 - INT 1
Ox804-0x805 - INT 2 - usado para el depurador (debugger) Ox806-0x807 - INT 30x808-0x809-Cronómetro
Ox80a-Ox80b - Receptor del puerto Serie Ox80c-Ox80d - Transmisor del puerto Serie Ox80e-Ox80f - Receptor del puerto serie TDM
Ox810-0x81 I - Transmisor del puerto serie TDM Ox812-0x813 - INT40x814-0x820 - reservado
Ox822-0x823 - TRAP - usado por el depurador (debugger)

0x24-0x825 – NMI0x826-0x827 - reservado
0x828-0x83f - Definido por el usuario

También hay que tener en cuenta que el ARP (el puntero auxiliar de registro) está listo para AR5 con ejecutar la rama vectorial de intem1pción. El contenido original de ARP es salvado para los 3 MSBs (bits más significativos) en STI y aquí se puede restaurar y extractar.

9. CONJUNTO ENSAMBLADOR DE LOS DISEÑADORES DE DSPs.

El Conjunto ensamblador de los diseñadores de DSPs (DSK5A.EXE) es muy simple y fácil para usar como ensamblador de la familia de procesadores del TMS320C5x. Sólo las características más esenciales de un ensamblador han sido incorporadas para conseguir el ensamblador tan fácil de usar como sea posible. Este esfuerzo ha sido realizado para mantener el DSK5A tan compatible como sea posible.

10. VISIÓN GENERAL.

El ensamblador del DSK5A difiere de muchos otros ensambladores en que no pasa por una fase de lineado para crear un archivo de salida. En lugar de eso el DSK5A usa directivas especiales para ensamblar código para una dirección absoluta ' en la mosca '. Una tabla de etiquetas genera al final de archivo de la lista el cual le puede dar la dirección absoluta de todas las etiquetas. La fuerza de este acercamiento es que estos pequeños y simples programas pueden ser creados rápidamente y fácilmente.
No esta previsto que se acoplen ejecutables o bibliotecas posteriormente.

11. OPCIONES DE LA LÍNEA DE COMANDO PARA INVOCAR AL DSK5A.

El ARCHIVO del DSK5A [.ASM]>[KL] asm “ code 2
| | |
| | Pre-ensambla ' código ' como un include
| +- Genera FILE.LST del archivo de la lista
+--- conserva salida. Ignorando ERRORES

ARCHIVOS ABIERTOS: FILE.DSK: Archivo de salida del DSK5A para el usar con DSK5D
FILE. UNP: El objeto temporal (automáticamente suprimido)
FILE.INC: El archivo usado al usar la opción del asm

Ejemplo: Para Ensamblar TEST.ASM para el TEST.DSK y generar un TEST.LST del archivo del listado

C:\DSK DSK5A TEST.ASM - L

Opción: - K. Si se da esta opción al DSK5A, intentará generar unos irregardless en el archivo de salida que nos informe de cualquier error o advertencia. Por defecto un archivo que contenga algún ERROR es suprimido.

- L Esta opción genera un archivo intermedio conteniendo los códigos de operación no resueltos. Los símbolos no resueltos están resueltos cuando el archivo entero ha sido leído. Como estos símbolos no se guardan en el archivo intermedio del objeto no son lincables.

- asm. Esta opción es un rasgo que permite al usuario definir declaraciones del ensamblador de la línea de comando. El código del ensamblador en este caso es terminado en ambos extremos con una cita doble. El carácter inmediatamente después de la primera cita es colocado en la primera columna. Las declaraciones ASM que están escritas para un archivo include FILE.INC sobrescribiendo el archivo previo usando ese nombre.

Ejemplo:

DSK5A TEST.ASM L ASM"FFT .SET 256 "ASM" .entry 0a00h "

Genera el archivo "TEST.INC" que contiene

La columna 1

|

FFT .SET 256

.0a00h de entrada

12 ARCHIVOS FUENTE del DSK5A.

-Editores:

Los archivos fuente del ensamblador del DSK5A pueden ser creados con casi cualquier editor de ASCII. Los caracteres admitidos son cualquier
carácter alfanumérico, espacio blanco y cualquier etiqueta. Con lo que no puede operar el DSK5A son archivos creados directamente para procesadores de texto que usan caracteres especiales de control para formatear. Usualmente estos editores pueden generar archivos abstractos del texto de ASCII, aunque no por defecto.

-Comentarios.

Existen dos formas para comentar una línea con el ensamblador del DSK5A. La primera y forma preferible es usar un punto y coma ";". Un punto y coma puede estar situado en cualquier parte de una línea y todo lo que haya a continuación se ignora. La segunda forma a añadir un comentario es usando un asterisco. En este caso el asterisco siempre debe estar en la primera columna. Si no lo está, entonces - se asumirá - que está en parte de un código de operación y probablemente generará un error.

-Etiquetas:

Todas las etiquetas deben estar situadas a en la primera columna y deben empezar con una letra o un subrayado"-". Después de la primera letra se pueden mezclar con números y poner como se quiera. Un dos puntos opcional ":", lo cuál es normalmente utilizado en otros ensambladores como un finalizado, no es necesario. Una etiqueta puede tener hasta 8 caracteres de longitud. Cuando un código de

operación o directiva hace referencias a una etiqueta, el valor de la dirección donde la etiqueta está ubicada en la memoria es substituido. La única excepción es el código de operación LDP, que carga los nueve bits más altos de la dirección.

-Códigos de operación:

Los códigos de operación deben comenzar después de la primera columna o después de un espacio en blanco o carácter siguiendo a una etiqueta. La sintaxis usada por los códigos de operación específicos del TMS320C5x puede ser encontrada en el TMS320C5x Users Guide. Las reglas que se aplican son dadas en los ejemplos siguientes.

Primero, es necesario resolver TODOS los campos de un código de operación. Si un campo del código de operación tal como el campo de cambio en un SACL el código de operación queda fuera, entonces el siguiente campo encontrado se convierte en el valor para ese campo y el siguiente campo no es rellenado. Si un campo no es especificado, entonces su valor se asume que es nulo y el código de operación es procesado de acuerdo con ello.

En segundo lugar, el análisis de la expresión no es realizado. Acciones como la adición de dos valores, multiplicaciones, cambios etcétera ... no son realizadas. Sólo la sustitución directa de variables definidas está permitida.

Ejemplo

```
        .ps
        LAR     AR7,RESULT
        MAR     *, AR7
        WRONG: SACL     *,AR7   ; llega a ser SACL *7 AR7 usado para
                                ; desplazar)
RIGHT: SACL  *,0,AR7; todos los campos especificados
OK:    SACL *       ; últimos dos campos no especificados
Word
.ds
RESULT .word O
```

Esto ocurre porque el valor numérico de AR7 se convierte en 7, el cuál cuándo se coloca en el código de operación B (en el campo equivocado) se convierte en * BRO + en el campo de modificación de AR.

NOTA: Los códigos de operación impropiamente especificados no siempre generan warnmgs.

-Directorios:

.entry [valor]

Cuando se encuentra esta directiva, el valor dado se convierte en la dirección de puesta en marcha del contador del programa cuando el archivo es cargado. Si no se da ningún valor la dirección de la memoria de programa actual determinado por la sección .ps ó .text se convierte en la dirección de inicio. Si más de una directiva .entry es encontrada en el archivo, el último .entry se convierte en la puesta en marcha dirección del código.

.end

Esta directiva termina el proceso de ensamblado haciendo cualquier código que se encuentre a continuación sea irrelevante. Hay que tener en cuenta que un .end inapropiadamente situado dentro de un archivo .include puede terminar el ensamblador prematuramente.

Label .byte valor1 , valor2, ... '

Esta directiva coloca uno ó más bytes dentro de las palabras de memoria consecutiva. Los 8 MSBs se llenan con Os. El valor puede ser numérico o carácter constante. La sustitución variable no es soportada.

LABl .byte - 1,2,0Ah, ' A ';Donde ' A ' es convertido a carácter de ASCII.

label . int valor1, [valor2,]
label .word valor1,[valor2,]

Las directivas .int y .word colocan a uno o más valores de 16 bits en la memoria. La constante de carácter es también legal. La sustitución variable no es soportada.

LAB I .int 0, - 1,2,0ABCDh, ' B '

label .long valor1,[valor2,]

La directiva .long coloca uno o más valores de 32 bits o la constantes de carácter en la memoria. La palabra menos significativa (LSW) es la primera almacenada.

```
LAB 1   .long 0, - 123456,234567,01234ABCDh, ' C '
```

```
label          .float    valor1, [valor2]...
label          .double   valor], [valor2}...
label          .efloat   valor], [valor2]...
label          .bjloat   valor], [valor2}...
label          .float    valorl, [valor2]...
```

Estas directivas convierten valores numéricos en diversos formatos de punto de flotante. Los formatos son:

.float IEEE-784 de 32 bits en formato de punto flotante
.doublé IEEE de 64 bits en formato de punto flotante
efloat Este formato usa un exponente de 16 bits con complemento a 2 y una mantisa de 16 bits con complemento a 2. El punto decimal de mantisa está entre los bits 14 y 15 haciendo el ler lugar explícito. Esto es no es insinuado como en los formatos IEEE en donde normalmente se suprime. En IEEE esto se suprime para permitir una precisión mayor desde que la mantisa esta siempre entre los valores 1,0 y 1,9999.
Para usar en DSPs un número IEEE necesitaría ser ' desempacado ' antes de que los miembros individuales sean intervenidos. Como la ejecución de la velocidad resultante va a ser sacrificada.

En los formatos de números .efloat, .bfloat y .tfloat se mantienen en su forma desempacada para mayor eficiencia.

. *fl.oat* es parecido a .efloat excepto que tiene exponente de 16 bit y mantisa de 32 bits..tfloat parecido a .efloat excepto que tiene 32 bit en el exponente y 64 bits en la mantisa.

NOTA: Éste es un formato pensado para el TMS320C40 y es sólo incluido para la compatibilidad. No se halla en disposición para un procesador de 16 bits

label .string valorl,[valor2,]...

Esta directiva coloca bytes en la memoria en un formato empacado en el orden que son encontrados. Si una palabra no está completamente llena, entonces el LSB de arrastre se rellenan con Os. El valor puede ser numérico, constante de carácter o cadena de caracteres. La sustitución variable no es soportada.

LABI .string -1,2,0Ah,"ABC",'D' ;"ABC" se convierte a 3 caracteres ASCII.

label ..space valor

Esta directiva es usada para insertar valores de bits en el segmento actual, ya sea programa o datos. Cada palabra en el espacio se llena con 0s y las palabras no completamente llenas son rellenadas con ceros adicionales.

Ejemplo:

```
    b past  ;
    .space 040h   ; Rellena 4 palabras con 0´s
past: nop
```

label .text y *label .data*

Estas directivas seleccionan que la salida del ensamblador vaya al programa o espacio de datos respectivamente. La dirección previamente usada (el programa o espacio de datos) es guardada en memoria facilitando mantener código o datos después de un descanso.

var .set valor

Esta directiva se usa para asignar un valor a una variable. Esta directiva es útil para establecer valores a todo lo largo del alcance del código ensamblado. Por ejemplo, .set puede ser usado para poner la carga y direcciones con elmmódulo del DSK s de entrada de programa o establecer valores para las variables comúnmente usadas como los registros del mapa de memoria .

.Ps [valor] y *.ds [valor]*

Estas directivas se usan para inicializar el programa y direcciones de carga de datos. Pueden aparecer las múltiples veces, haciendo posible para seleccionar cualquier cosa no grabada para las aberturas que están producidas. También, si las direcciones traslapan, entonces es posible sobrescribir código o datos con estas directivas. Si el valor es omitido, entonces la última dirección es usada como en .text o .data.

. mmregs

Estos generarán los nombres simbólicos para los registros del C50 y colocarlos .n las tablas de símbolos. Es equivalente a ejecutar

```
IMR .set 4
GREG .set 5
```

.qxx y *.lqxx / * enformato Q 32 bits * I*

Estas directivas se usan para generar integradores fraccionales en complemento a 2 y valores de entero largo cuyo punto decimal es desplazado xx lugares del LSB.

Ejemplo:

```
.ds             0400h
. Q15           0.25, 0.5, 0.75
        <- 15 lugares
0400 00000h; #0.000000000000000b
0401 02000h;  #0.01 0000000000000b
0402 04000h;  #0.100000000000000b
0403 06000h;  #0.110000000000000b
```

.Include "file.ext" y *.copy ':file.ext"*

Estas directivas se usan para encadenar juntos archivos facilitando así construir programas grandes. Hay que tener en cuenta que porque no hay fase de lineado del ensamblador DSK entonces las definiciones múltiples de etiquetas y las variables generarán errores y advertencias. El Anidado de los archivos incluidos de hasta 8 niveles está permitido antes de terminar el ensamblador.

Ejemplo:

```
.include    "MMRS.ASM"   ; incluye un archivo definiendo los mmregs
.include    "LINK.ASM"   ; incluye un archivo definiendo la carga
                         ; y direcciones de entrada de programas
Include     "VECTS.ASM"  : incluye una tabla de vector
      ; AQUÍ METER CÓDIGO
```

.if valor, .else y *.endif*

Estas directivas se usan para las secciones de ensamblado condicional de código. El segmento .if es ensamblado si el valor es cero. El valor puede ser numérico ó una variable predeterminada (.set). Hay que tener en cuenta que como el DSK5A no tiene una fase de lineado y es un ensamblador de un paso, las variables deben ser inicializadas ANTES de usarlas. De otra manera su valor será cero. También hay que tener en cuenta que el análisis de expresión no está soportada.

Ejemplo:

```
ELOOP    .set   1
         .if    ELOOP
  SELF    B     SELF
         .else
         NOP
         .endif
```

.El liston y *.listoff*

Estas directivas son algunas veces útiles en depurar el código. Están acostumbradas a pasar a disposición de la línea de comando - opción L que puede encender el listado de la salida. El listado es siempre escrito para un archivo cuyo nombre está derivado de la fuente originaria con una extensión de archivo .LST.

NOTA: .listoff también desactiva el desensamblado ya que este rasgo está en parte del archivo de LIST.

13. ARCHIVOS DE SALIDA DEL DSK.

El archivo de salida DSK soporta direcciones de 16 bits en programa y espacio de datos permitiendo la colocación directa de código del tiempo de ejecución en la tarjeta.

```
+- ,caracteres DSK:      La K designa archivo de inicio
|                         1 designa punto de entrada de código
          |                      9 designa dirección de carga
        |  +-- DSK nombre de archivo de revisión y salida
        |  |                          +-- Continua la suma de control
        /\ [----------------]                        |       +-- fin de línea
        K_DSKA_1.01_TEST.DSK                         |       |
        1FB007FBOOF                                  /\      /\
        9FBOOB5588B55A9B55AAB55ABB55ACB55ADB55AEB55AF7
           AD3CF
        9FB08BFFA8BFBO  IB67EOB60F07C379F
        904001\100001\100001\1123471234F

                    +-- B designa la carga del espacio del programa
                       M designa la carga del espacio de datos

        +-- Fin de  registro de archivo
```

B.2 SET DE INSTRUCCIONES DEL TMS320C5x.

1. GRUPOS FUNCIONALES

- MEMORIA DEL ACUMULADOR
- REGISTROS AUXILIARES Y PUNTERO DE LA MEMORIA DE DATOS
- UNIDAD LÓGICA PARALELA (PLU)
- TREGO, PREGO Y MULTIPLICACIÓN
- DE RAMA Y DE LLAMADA
- I/O Y OPERACIÓN DE MEMORIA DE DATOS
- DE CONTROL

2. MEMORIA DEL ACUMULADOR

ABS: Valor absoluto del acumulador; bit de carry cero.

ADCB: Suma el buffer del acumulador y el bit de carry al acumulador.

ADD: Suma el valor de la memoria de datos, con desplazamiento a la izquierda, al acumulador.
Suma el valor de la memoria de datos, con desplazamiento a la izquierda de 16, al acumulador.
Suma el inmediato corto al acumulador.
Suma el inmediato largo, con desplazamiento a la izquierda, al

acumulador. ADDB: Suma el buffer del acumulador al acumulador.

ADDC: Suma el valor de la memoria de datos y el carry al acumulador con extensión de signo suprimida.

ADDS: Suma el valor de la memoria de datos al acumulador, con extensión de signo suprimida.

ADDT: Suma el valor de la memoria de datos, con desplazamiento a la izquierda especificado por TREG1, al acumulador.

AND: Hace un AND al valor de la memoria de datos con los bytes bajos del acumulador; hace cero los bytes altos del acumulador.

Hace un AND inmediato largo, con desplazamiento a la izquierda, con el acumulador.

Hace un AND inmediato largo, con desplazamiento a la izquierda de 16, con el acumulador.

ANDB: Hace un AND al buffer del acumulador (ACCB) con el acumulador (ACC).

BSAR: Hace un desplazamiento de almacenamiento del acumulador a la derecha.

CMPL: Complementa a 1 el acumulador.

CRGT: Almacena el acumulador en el buffer del acumulador si ACC >
ACCB.

CRLT: Almacena el acumulador en el buffer del acumulador si ACC <
ACCB.

EXAR: Intercambia el buffer del acumulador (ACCB) con el acumulador (ACC).

LACB: Carga el ACC al ACCB.

LACC: Carga el valor de la memoria de datos, con desplazamiento a la izquierda, al ACC.

Carga el inmediato largo, con desplazamiento a la izquierda, al ACC.

Carga el valor de la memoria de datos, con desplazamiento a la izquierda de 16, al ACC.

LACL: Carga el valor de la memoria de datos a los bytes bajos del ACC; hace cero los bytes altos del ACC.

Carga el inmediato corto a los bytes bajos del ACC; hace cero los bytes altos del ACC.

LACT: Carga el valor de la memoria de datos, con desplazamiento a la izquierda especificado por TREG 1, al ACC.

LAMM: Carga el contenido del registro mapeado en memoria a ACCL; hace cero el ACCH.

NEG: Niega (complemento a 2) el ACC. NORM: Normaliza el ACC.

NOR: Hace un OR con el valor de la memoria de datos y el ACCL.
Hace un OR con el inmediato largo, con desplazamiento a la izquierda, y el ACC.
Hace un OR con el inmediato largo, con desplazamiento a la izquierda de 16,
y el ACC.

ORB: Hace un OR con el ACCB y el ACC.

ROL: Rota el ACC a la izquierda un bit.

ROLB: Rota el ACCB y el ACC a la izquierda un bit.

ROR: Rota el ACC a la derecha un bit.

RORB: Rota el ACCB y el ACC a la derecha un bit.

SACB: Almacena el ACC en el ACCB.

SACH: Almacena el ACCH, con desplazamiento a la izquierda, en la localización de la memoria de datos.

SACL: Almacena el ACCL, con desplazamiento a la izquierda, en la localización de la memoria de datos.

SAMM: Almacena el ACCL en el registro mapeado en memoria.

SATH: Desplaza con almacenamiento el ACC a la derecha 0 ó 16 bits, según especifica el TREG 1.

SATL: Desplaza con almacenamiento el ACC a la derecha, según especifica el TREG1.

SBB: Resta el ACCB del ACC.

SBBB: Resta el ACCB y la inversión lógica del bit de carry, del ACC.

SFL: Desplaza el ACC a la izquierda un bit.

SFLB: Desplaza el ACCB y el ACC a la izquierda un bit.

SFR: Desplaza el ACC a la derecha un bit.

SFRB: Desplaza el ACCB y el ACC a la derecha un bit.

SUB: Resta el valor de la memoria de datos, con desplazamiento a la izquierda, del ACC.
Resta el valor de la memoria de datos, con desplazamiento a la izquierda de 16, del ACC.
Resta el inmediato corto del ACC.
Resta el inmediato largo, con desplazamiento a la izquierda, del ACC.

SUBB: Resta el valor de la memoria de datos y la inversión lógica del bit de carry del ACC con extensión de signo suprimida.

SUBC: Resta condicional.

SUBS: Resta el valor de la memoria de datos del ACC con extensión de signo suprimida.

SUBT: Resta el valor de la memoria de datos, con desplazamiento a la izquierda especificado por el TREG 1, del ACC.

XOR: Hace un OR exclusivo del valor de la memoria de datos y el ACCL. Hace un OR exclusivo del inmediato largo, con desplazamiento a la izquierda de 16, y el ACC.
Hace un OR exclusivo del inmediato largo, con desplazamiento a la izquierda, y el ACC.

XORB: Hace un OR exclusivo del ACCB y el ACC.

ZALR: Hace cero el ACCL y carga el ACCH con redondeo.

ZAP: Hace cero el ACC y el PREG.

3. REGISTROS AUXILIARES Y PUNTERO DE LA MEMORIA DE DATOS.

ADRK: Suma el inmediato corto al registro auxiliar (AR).

CMPR: Compara el AR con el registro de comparación del registro auxiliar (ARCR) como especifican los bits CM.

LAR: Carga el valor de la memoria de datos al ARx.
Carga el inmediato corto al ARx.
Carga el inmediato largo al ARx.

LDP: Carga el valor de la memoria de datos a los bits del puntero de la memoria de datos.
Carga el inmediato corto a los bits del puntero de la memoria de datos.

MAR: Modifica el AR.

SAR: Almacena el ARx en la localización de la memoria de datos.

SBRK: Resta el inmediato corto del AR.

4. UNIDAD LÓGICA PARALELA (PLU).

APL: Hace un AND con el valor de la memoria de datos y el registro de manipulación de bit dinámico, y almacena el resultado en la localización de la memoria de datos. Hace un AND con el valor de la memoria de datos y el inmediato largo, y almacena el resultado en la localización de la memoria de datos.

CPL: Compara el valor de la memoria de datos con el registro de manipulación de bit dinámico (DBMR). Compara el valor de la memoria de datos con el inmediato largo.

OPL: Hace un OR con el valor de la memoria de datos y el DBMR, y almacena el resultado en la localización de la memoria de datos.

Hace un OR con el valor de la memoria de datos y el inmediato largo, y almacena el resultado en la localización de la memoria de datos.

SPLK: Almacena el inmediato largo en la localización de la memoria de datos.

XPL: Hace un OR exclusivo del valor de la memoria de datos con el DBMR, y almacena el resultado en la localización de la memoria de datos.
Hace un OR exclusivo del valor de la memoria de datos con el inmediato largo, y almacena el resultado en la localización de la memoria de datos.

LPH: Carga el valor de la memoria de datos al byte alto del registro de producto (PREG).

LT: Carga el valor de la memoria de datos al registro de temporización TREGO. '

5. TREGO, PREG Y MULTIPLICACIÓN.

LTA: Carga el valor de la memoria de datos a TREGO; suma PREG, con desplazamiento especificado por los bits PM, al ACC.

LTD: Carga el valor de la memoria de datos a TREGO; suma PREG, con desplazamiento especificado por los bits PM, al ACC, y mueve datos.

LTP: Carga el valor de la memoria de datos a TREGO; almacena PREG, con desplazamiento especificado por los bits PM, en el ACC.

LTS: Carga el valor de la memoria de datos a TREGO; resta PREG, con desplazamiento especificado por los bits PM, del ACC.

MAC: Suma PREG, con desplazamiento especificado por los bits PM, al ACC; carga el valor de la memoria de datos a TREGO; multiplica el valor de la memoria de datos por el valor de la memoria del programa y almacena el resultado en PREG.

MACD: Suma PREG, con desplazamiento especificado por los bits PM, al ACC; carga el valor de la memoria de datos a TREGO; multiplica valor de la memoria de datos por el valor de la memoria del programa y almacena el resultado en PREG; y mueve los datos.

MADD: Suma PREG, con desplazamiento especificado por los bits PM, al ACC; carga el valor de la memoria de datos a TREGO; multiplica el valor de la memoria de datos por el valor especificado en BMAR y almacena el resultado en PREG; y mueve los datos.

MADS: Suma PREG, con desplazamiento especificado por los bits PM, al ACC; carga el valor de la memoria de datos a TREGO; multiplica el valor de la memoria de datos por el valor especificado en BMAR y almacena el resultado en PREG.

MPY: Multiplica el valor de la memoria de datos por TREGO y almacena el resultado en PREG.
Multiplica el inmediato corto por TREGO y almacena el resultado en PREG. Multiplica el inmediato largo por TREGO y almacena el resultado en PREG.

MPYA: Suma PREG, con desplazamiento especificado por los bits PM, al ACC; multiplica el valor de la memoria de datos por TREGO y almacena el resultado en PREG.

MPYS: Resta PREG, con desplazamiento especificado por los bits PM, del ACC; multiplica el valor de la memoria de datos por TREGO y almacena el resultado en PREG.

MPYU: Multiplica el valor de la memoria de datos sin signo por TREGO y almacena el resultado en PREG.

PAC: Carga PREG, con desplazamiento especificado por los bits PM, al ACC.

SPAC: Resta PREG, con desplazamiento especificado por los bits PM, del ACC.

SPH: Almacena PREG (bytes altos), con desplazamiento especificado por los bits PM, en la localización de la memoria de datos.

SPL: Almacena PREG (bytes bajos), con desplazamiento especificado por los bits PM, en la localización de la memoria de datos.

SPM: Establece bits de modo de desplazamiento de producto (PM). ·

SQRA: Suma PREG, con desplazamiento especificado por los bits PM, al ACC; carga el valor de la memoria de datos a TREGO; hace raíz del valor y almacena el resultado en PREG.

SQRS: Resta PREG, con desplazamiento especificado por los bits PM, del ACC; carga el valor de la memoria de datos a TREGO; hace raíz del valor y almacena el resultado en PREG.

ZPR: Hace cero al PREG.

6. INSTRUCCIONES DE RAMA Y DE LLAMADA

B: Rama sin condición a la localización de la memoria del programa.

BACC: Rama a la localización de la memoria de programa especificada por ACCL.

BACCD: Rama de retardo a la localización de la memoria de programa especificada por ACCL.

BANZ: Rama a la localización de la memoria de programa si AR no es cero.

BANZD: Rama de retardo a la localización de la memoria de programa si AR no es cero.

BCND: Rama condicional a la localización de la memoria de programa.

BCND: Rama condicional de retardo a la localización de la memoria de programa.

BD: Rama incondicional de retardo a la localización de la memoria de programa.

CALA: Llamada a subrutina direccionada por ACCL.

CALAD: Llamada de retardo a subrutina direccionada por ACCL.
CALL: Llamada a subrutina sin condiciones.
CALLD: Llamada de retardo a subrutina sin condiciones.

CC: Llamada a subrutina condicional.

CCD: Llamada de retardo a subrutina condicional.

INTR: Interrupción software que ramifica el control del programa a la localización de la memoria de programa.

NMI: Interrupción no enmascarada que deshabilita globalmente las interrupciones (INUN= 1).

RET: Retomo de subrutina.

RETC: Retorno de subrutina condicionada.

RETCD: Retorno con retardo de subrutina condicionada. RETD: Retomo con retardo de subrutina.

RETE: Retomo de subrutina con switch de contexto que habilita globalmente las interrupciones (INUN=O).

RETI: Retomo de subrutina con switch de contexto.

TRAP: Interrupción software que ramifica el control del programa a la localización de la memoria de programa 22h.

XC: Ejecuta la siguiente instrucción condicionada.

7. I/O Y OPERACIÓN DE LA MEMORIA DE DATOS.

BLDD: Movimiento de bloque de datos a la memoria de datos.
Movimiento de bloque de datos a la memoria de datos con dirección destino inmediata larga.

Movimiento de bloque de datos a la memoria de datos con dirección fuente en el registro de direccionamiento de movimiento de bloque (BMAR).
Movimiento de bloque de datos a la memoria de datos con dirección destino en el registro de direccionamiento de movimiento de bloque (BMAR).

BLDP: Movimiento de bloque de datos a la memoria de programa con dirección destino en BMAR.

BLPD: Movimiento de bloque del programa a la memoria de datos con dirección fuente en BMAR.
Movimiento de bloque del programa a la memoria de datos con dirección fuente en la inmediata larga.

DMOV: Mueve datos en la memoria de datos.

,,
IN: Entrada de datos del puerto I/O a la localización de la memoria de datos.

LMMR: Carga el valor de la memoria de datos al registro mapeado en memoria.

OUT: Salida de datos de la localización de la memoria de datos al puerto I/O.

SMMR: Almacena el registro mapeado en memoria en la localización de la memoria de datos.

TBLR: Transfiere datos del programa a la memoria de datos con dirección fuente en el ACCL.

TBLW: Transfiere datos de los datos a la memoria de programa con dirección destino en el ACCL.

8. INSTRUCCIONES DE CONTROL.

BIT: Prueba un bit.

BITT: Prueba un bit especificado por TREG2.

CLRC: Borra el bit de modo overflow (OVM).

Borra el bit de modo extensión de signo (SXM).
Borra el bit de modo hold (HM).
Borra el bit de test / control (TC).
Borra el bit de carry (C).
Borra el bit de control de configuración (CNF).
Borra el bit de modo interrupción (INUN).
Borra el pin de señalización externa (XF).

IDLE: Inactiva hasta la interrupción de desenmascaramiento o reseteo.

IDLE2: Inactiva hasta la interrupción de desenmascaramiento o reseteo modo de baja potencia.

LST: Carga el valor de la memoria de datos al registro de estado STO.
Carga el valor de la memoria de datos al registro de estado ST1.

NOP: No operación.

POP: Hace un pop del límite de la pila al ACCL; hace cero al ACCH.

POPD: Hace un pop del límite de la pila a la localización de la memoria de datos.

PSHD: Hace un push del valor de la memoria de datos al límite de la pila.

PUSH: Hace un push del ACCL al límite de la pila.

RPT: Repite la siguiente instrucción especificada por el valor de la memoria de datos .Repite la siguiente instrucción especificada por el inmediato corto. Repite la siguiente instrucción especificada por el inmediato largo.

RPTB: Repite el bloque de instrucciones especificado por BRCR.

RPTZ: Borra el ACC y el PREG; repite la siguiente instrucción especificada por el inmediato largo.

SETC: Establece el bit de modo overflow (OVM).
Establece el bit de modo extensión de signo (SXM).
Establece el bit de modo hold (HM).

Establece el bit de test / control (TC).
Establece el bit de carry (C).
Establece el pin de señalización externa (XF) a alto.
Establece el bit de control de configuración (CNF).
Establece el bit de modo interrupción (INUN).

SST: Almacena el STO en localización de la memoria de datos.
Almacena el ST1 en localización de la memoria de datos.

B.3 USOS GENERALES DEL TMS320C50.

1. USOS GENERALES DEL TMS320C50

Esta guía de usuario proporciona aplicaciones para la generación del TMS320C5x procesadores de señales digitales (DSPs) de punto fijo de la familia del TMS320. El DSP ' C5x provee un desempeño mejorado sobre las generaciones ' C1 y ' C2x al mantener compatibilidad ascendente de código de la fuente entre los dispositivos.

La unidad procesadora central C5x (CPU) se basa en la CPU ' C25 e incorpora realces adicionales en la arquitectura que permiten al dispositivo ir el doble de rápido que los dispositivos C2x. Se espera una expansión futura y realces que eleven el desempeño y el rango de aplicaciones de los DSPs ' C5x.

La generación C5x de DSPs CMOS estáticos consta de los siguientes dispositivos:

Dispositivo	**RAM en chip**	**ROM en chip**
TMS320C50/LC50	palabras de 10K	palabras de 2K
TMS320C51/LC51	palabras de 2K	palabras de 8K
TMS320C52/LC52	palabras de 1K	palabras de 4K
TMS320C53/LC53	palabras de 4K	palabras de 16K
TMS320C53S/LC53S	palabras de 4K	palabras de 16K
TMS320LC56	palabras de 7K	palabras de 32K
TMS320C57S	palabras de 7K	palabras de 2K
TMS320LC57	palabras de 7K	palabras de 32K

1.1 Repaso a la familia TMS320

La familia del TMS320 consta de dos tipos de DSPs de un solo chip: punto flotante de 32 bits y punto fijo de 16 bits. Estos DSPs poseen la flexibilidad operacional de controladores de alta velocidad y la capacidad numérica de procesadores de array.

Respecto al apartado "1.1.2. Aplicaciones típicas del TMS320", hay una lista detallada de aplicaciones de la familia del TMS320. Las siguientes características hacen a esta familia la elección ideal para una gran variedad de procesar aplicaciones:

- Las instrucciones son muy flexibles
- Flexibilidad operacional inherente
- Alta velocidad
- Innovación, diseño de arquitectura paralela
- Coste-Efectividad

1.1.1 Historia, Desarrollo, y Ventajas de los DSPs TMS320.

En 1982, Texas Instruments presentó el TMS32010, el primer DSP de punto fijo en la familia del TMS320. Antes del fin de año, la revista Electronic Products recompensó al TMS32010 con el título "Producto de Año ". El TMS32010 se convirtió en el modelo para generaciones futuras del TMS320.

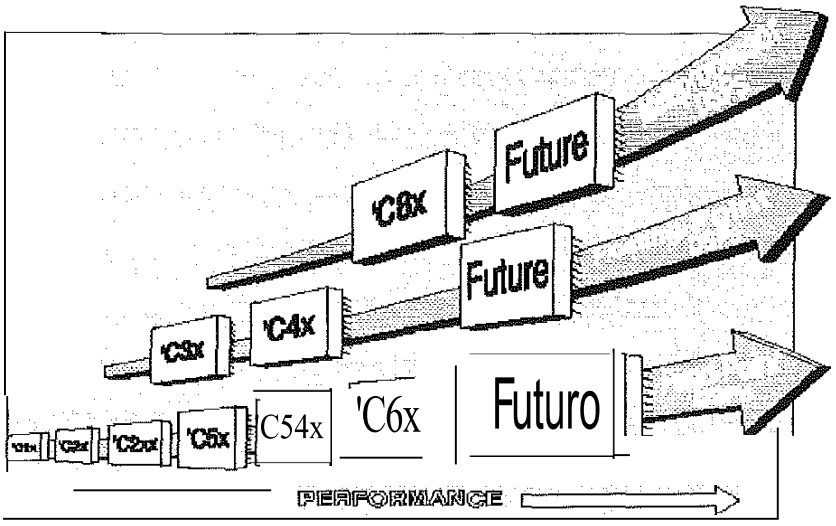
Hoy, la familia del TMS320 consta de estas generaciones (la Figura 1- 1): 'C1x, ' C2x, ' C2xx, ' C5x, ' C54x, y ' C6x DSPs de punto fijo; los DSPs de punto flotante 'C3x y 'C4x; Y los DSPs multiprocesador C8x.

La figura 1-1 ilustra las ganancias de desempeño que la familia del TMS320 ha hecho en el tiempo con generaciones sucesivas. El código de la fuente es compatible de una generación de punto fijo para la siguiente generación de punto fijo (excepto para el ' C54x), y de una generación de punto flotante para la siguiente generación de punto flotante. La compatibilidad ascendente conserva la generación de software de su inversión, por lo tanto provee una manera conveniente y eficiente en base a costos para un desempeño superior, sistema DSP más versátil.

Cada generación de dispositivos TMS320 tiene una CPU y una variedad de memoria en el chip y configuraciones periféricas para dispositivos derivados en vías de desarrollo. Estos dispositivos derivados satisfacen una gran variedad de necesidades en el mercado mundial de electrónica.

Cuándo la memoria y los periféricos están integrados en un procesador, el costo global del sistema se acorta enormemente, y se ahorra en espacio para la placa del circuito.

Evolución de la familia TMS320:



1.1.2. Aplicaciones Típicas del TMS320.

La familia TMS320 de DSPs ofrece mejores y más ajustables avances para los problemas tradicionales de procesado 'de la señal, como codificación de voz, filtrado, y la codificación de errores. Además, la familia del TMS320 da soporte a las aplicaciones complicadas que a menudo requieren operaciones múltiples para ser realizadas simultáneamente. La figura 1-2 muestra muchas de las aplicaciones típicas de la familia del TMS320.

Aplicaciones típicas para la familia TMS320.

Automoción
Control de manejo adaptable
Frenos antideslizantes
Teléfonos móviles
Radios digitales
Control del motor
Posicionamiento global
Navegación
Análisis de vibración
Comandos de voz

Consumo
Radios/ TVs digitales
Juguetes educativos
Sintetizadores de música
Herramientas mecánicas
Detectores de radar
Contestadores automáticos de estado sólido
Control de la unidad de disco

Control
Control de impresoras láser
Control de motores
Control de robótica
Servocontrol

Objetivos generales
Filtrado adaptable
Convolución Correlación
Filtrado digital
Transformadas de Fourier
Transformadas Hilbert
Generación de formas de ondas
Ventanamiento

Gráficos/ Imagen
Rotación 3-D
Animación/ mapas digitales
Procesamiento homomórfico
Realce de imagen
Compresión/ transmisión de imagen
Visión automatizada
Estaciones de trabajo

Industria
Control numérico
Monitorización de línea de energía Robótica
Acceso de seguridad

Instrumentación

Filtrado digital
Generación de funciones
Lazos cerrados en fase
Procesamiento sísmico
Análisis del espectro
Análisis transitorio

Medicina

Equipo diagnóstico
Monitorización fetal
Monitorización del paciente
Equipo de ultrasonido

Militar

Procesamiento de la imagen
Guía de misiles
Navegación
Procesamiento de radar
Modems de radiofrecuencia
Comunicaciones de seguridad
Procesamiento de sonar

Telecomunicaciones

Modems de 1200 a 19200-bps
Ecualizadores adaptables
Teléfonos móviles
Multiplexado del canal
Encriptación de datos
Interpolación de habla digital (DSI)
Asistentes digitales personales (PDA)
DUNF codificación/ decodificación
Cancelación de eco

Fax

Repetidores de la línea
Teléfonos
Comunicaciones del espectro

Conferencia por video
Sistemas de comunicaciones personales (PCS)

Voz/ habla

Realce del habla
Reconocimiento del habla
Síntesis del habla
Correo de voz
De texto a voz

1.2 Repaso al TMS320C5x.

La generación C5x consiste en los DSPs ' C50, ' C51, ' C52, ' C53, ' C53S, ' C56, ' C57, y 'C57S, que son fabricados por la tecnología de circuito integrado CMOS.

Su diseño arquitectónico se basa en el ' C25. La flexibilidad operacional y la velocidad del ' C5x son el resultado de combinar una avanzada arquitectura Hardware (que tiene buses separados para la memoria de programa y memoria de datos), una CPU con lógica específica en la aplicación del hardware, periféricos en chip, memoria en el chip, y un equipo de instrucciones altamente especializado. El ' C5x es diseñado para ejecutar hasta 50 millones de instrucciones por segundo (MIPS). Los aparatos derivados que combinan la CPU ' C5x con configuraciones periféricas y memoria en el chip personalizadas pueden ser desarrollados para aplicaciones especiales en el mercado mundial de la electrónica.

Los dispositivos C5x ofrecen estas ventajas:

- El realzado diseño arquitectónico del TMS320 para el aumento de la eficacia y la versatilidad
- El-diseño de su arquitectura modular para el desarrollo rápido de dispositivos derivados
- Avanzada tecnología de procesamiento de circuito integrado para aumentar la eficacia y el consumo bajo de energía
- Compatibilidad de código de fuente con los DSPs' Clx, ' C2x, y C2xx para programas rápidos y fáciles de realizar
- Reforzado equipo de instrucciones para los algoritmos más rápidos y para las operaciones de lenguaje de alto nivel optimizadas
- Reducido consumo de energía y aumento de la dureza de la radiación por las nuevas técnicas de diseño estático.

La tabla 1-1 lista las características principales de los DSPs 'C5x. La tabla muestra la capacidad de la RAM en chip y la ROM en chip, número de puertos de entrada/salida serie y paralelo, requisitos de la alimentación, tiempo de ejecución de un ciclo de la máquina, y los tipos de paquete disponibles con cuenta total de pines. Usando la tabla 1-1 se puede escoger el mejor DSP 'C5x para una aplicación determinada.

TMS320 Device	ID	On-Chip Memory (16-bit words) DARAM†	SARAM†	ROM	I/O Ports Serial	Parallel‡	Power Supply (V)	Cycle Time (ns)	Package Type
'C50	PQ	1056	9K	2K§	2¶	64K	5	50/35/25	132 pin BQFP=
'LC50	PQ	1056	9K	2K§	2¶	64K	3.3	50/35/25	132 pin BQFP=
'C51	PQ	1056	1K	8K§	2¶	64K	5	50/35/25/20	132 pin BQFP=
'C51	PZ	1056	1K	8K§	2¶	64K	5	50/35/25/20	100 pin TQFP=
'LC51	PQ	1056	1K	8K§	2¶	64K	3.3	50/35/25	132 pin BQFP=
'LC51	PZ	1056	1K	8K§	2¶	64K	3.3	50/35/25	100 pin TQFP=
'C52	PJ	1056	—	4K§	1	64K	5	50/35/25/20	100 pin QFP=
'C52	PZ	1056	—	4K§	1	64K	5	50/35/25/20	100 pin TQFP=
'LC52	PJ	1056	—	4K§	1	64K	3.3	50/35/25	100 pin QFP=
'LC52	PZ	1056	—	4K§	1	64K	3.3	50/35/25	100 pin TQFP=
'C53	PQ	1056	3K	16K§	2¶	64K	5	50/35/25	132 pin BQFP=
'C53S	PZ	1056	3K	16K§	2	64K	5	50/35/25	100 pin TQFP=
'LC53	PQ	1056	3K	16K§	2¶	64K	3.3	50/35/25	132 pin BQFP=
'LC53S	PZ	1056	3K	16K§	2	64K	3.3	50/35/25	100 pin TQFP=
'LC56	PZ	1056	6K	32K	2#	64K	3.3	50/35/25	100 pin TQFP=
'C57S	PGE	1056	6K	2K§	2#	64K‖	5	50/35/25	144 pin TQFP=
'LC57	PBK	1056	6K	32K	2#	64K‖	3.3	50/35/25	128 pinTQFP=

† Dual-access RAM (DARAM)
‡ Single-access RAM (SARAM)
§ ROM bootloader available
¶ Includes time-division multiplexed (TDM) serial port
# Includes buffered serial port (BSP)
‖ Includes host port interface (HPI)
= 20 × 20 × 3.8 mm bumpered quad flat-pack (BQFP) package
= 14 × 14 × 1.4 mm thin quad flat-pack (TQFP) package
= 14 × 20 × 2.7 mm quad flat-pack (QFP) package
= 20 × 20 × 1.4 mm thin quad flat-pack (TQFP) package
‖ Sixteen of the 64K parallel I/O ports are memory mapped.

1.3 Características principales del TMS320C5x.

Las características principales de los DSPs ' C5x están listadas debajo.

Donde un rasgo es exclusivo para un dispositivo particular, el nombre del dispositivo es incluido entre paréntesis y anotado después de ese rasgo.

- La compatibilidad: El código de fuente es compatible con los dispositivos ' C1x, ' C2x, y ' C2xx

- Velocidad: El tiempo de ejecución de instrucción de punto fijo de un solo ciclo es 20-/25 /25 /35 150 ns (50/40/28.6/20 MIPS)

- Alimentación.
  * tecnología estática CMOS 3.3 V y *5* V con dos modos
  * Control de consumo con las instrucciones IDLE1 e IDLE2 para los dos modos

- Memoria.
  * espacio de memoria direccionable externo máximo de palabras de

224K y 16 bits (programa de palabras de 64K, datos de palabras de 64K, entrada/ salida de palabras de 64K, y memoria global de palabras de 32K)

  * RAM de datos en chip de doble acceso de 1056 palabras x D 16 bits.
  * RAM de programa/ datos en chip de simple acceso de palabras de 9K D x 16 bits.('C50)
  * ROM de boot en chip de simple acceso de palabras de 2K D x 16 bits ('C50,'C57S)
  * RAM de datos/ programa en chip de simple acceso de palabras de 1 K x 16 bits ('C51)
  * ROM de programa en chip de simple acceso de palabras de 8K x O 16 bits ('C51)
  * RQM de programa en chip de simple acceso de palabras de 4K D x 16 bits ('C52)
  * RAM de programa/ datos en chip de simple acceso de palabras de 3K D x 16 bits ('C53, 'C53S)
  * ROM de programa en chip de simple acceso de palabras de I6K x 0 16 bits ('C53, 'C53S)
  * RAM de programa/ datos en chip de simple acceso de palabras de 6K x O 16 bits ('LC56, 'C57S, 'LC57)

*

* ROM de programa en chip de simple acceso de palabras de 32K x 16 bits ('LC56, 'LC57)

- Unidad central de proceso (CPU)

* La unidad lógica aritmética central (CALU) consiste en lo siguiente:

+ La unidad lógica aritmética (ALU) de 32 bits, el acumulador de 32 bits (ACC), y el buffer del acumulador de 32 bits (ACCB)

+ multiplicador paralelo de 16 bits por 16 bits con una capacidad de producto de 32 bits

+ Desplazamiento y almacenamiento de datos a izquierda y derecha de 0 a 16 bits y desplazamiento de datos incremental de 64 bits

* Unidad lógica paralela de 16 bits (PLU)

* Unidad aritmética de registro auxiliar (ARAU) para direccionamiento indirecto

* Ocho registros auxiliares

- Control de programa

* Pila hardware de 8 niveles

*4 operaciones de distribución profunda para retardo de rama, llamada, e instrucciones de retorno.

* Once registros sombra para almacenar registros estratégicos controlados por la CPU durante una rutina de servicio de interrupción (ISR)

*Operaciones de retención extendida para acceso de memoria directo externo (DMA) de la memoria externa o la RAM en chip

*Dos buffers circulares direccionados indirectamente para direccionamiento circular

- Grupo de instrucciones

* Instrucciones de multiplicar/ acumular en un solo ciclo

* Operaciones de repetición de una sola instrucción y de bloque

* Instrucciones de movimiento de bloques de memoria para programas
mejores y gestión de datos

* Instrucciones de carga y almacenamiento de registros de mapeado de memoria.

* Instrucciones de rama y de llamada condicional

* * Ejecución retardada de las instrucciones de rama y llamada

* * Retomo rápido de instrucciones de intern1pción

* Modo de direccionamiento indexado

* Modo de direccionamiento indexado en inversión de bit para transformadas de Fourier (FFTs) de radio 2

Periféricos en chip

* Puertos paralelo I/O de 64K (16 pue1tos I/O están mapeados en

memoria)

* Dieciséis generadores de estado de espera programables en software para programa, datos, y espacios de memoria I/O

* Temporizador de intervalos con período, control, y registros de conteo para parada, arranque y reseteo del software

* Generador de reloj de lazo cerrado en fase (PLL) con oscilador interno o fuente de reloj externa

* Múltiple opción de reloj PLL (xi , x2, x3, x4, x5, x9, dependiendo del dispositivo)

* Interfaz de puerto serie síncrona full-dúplex para comunicación directa entre el ' C5x y otro dispositivo serie.

* Puerto serie (' C50, ' C51, ' C53) de multiplexado de división por tiempo (TDM).

* Puerto serie de buffer (BSP) (' LC56, ' C57S, ' LC57).

* Interfaz de puerto paralelo de 8 bits (HPI) (' C57, ' C57S).

- Test/ emulación.

* Lógica de emulación basada en sean en chip.

* Estándar IEEE JTAG con lógica de sean 1149.1 (' C50, ' C5I , ' C53, ' C57S).

- Paquetes.

* Paquete (' C52) de 100 pines quad flat-pack (QFP).

* Paquete (' C51, ' C52, ' C53S, ' LC56) de 100 pines thin quad flat- pack (TQFP).

* Paquete TQFP de 128 pines (' LC57).

* Paquete de 132 pines bumpered quad flat-pack (BQFP), (' C50, ' C51, ' C53)

* Paquete TQFP de 144 pines (' C57S).

2.APLICACIONES SOFTWARE.

Los dispositivos 'C5x mantienen compatibilidad de código de fuente con generaciones 'C1x y 'C2x y tienen características de arquitectura que mejoran su realización y versatilidad. Un set de instrucciones ortogonal se magnifica por instrucciones nuevas que soportan hardware adicional y manejan datos y registros mapeados en memoria. Otras características incluyen una unidad lógica paralela independiente (PLU) para realizar operaciones Booleanas, un buffer de acumulador (ACCB) de 32 bits, y un set de registros que proporcionan capacidades de cambio de contexto latencia cero para interrumpir rutinas en servicio. La RAM de acceso dual en chip y el set de registros mapeados en memoria son realzados.

Este capítulo explica el uso del set de instrucciones 'C5x con un énfasis particular en sus nuevas características y aplicaciones especiales. Para un completo estudio de las directivas del ensamblador utilizadas en los ejemplos de este capítulo, consultar la guía de usuario de herramientas del lenguaje ensamblador del TMS320C1x/C2x/C2xx/C5x.

2.1 **Inicialización del Procesador.**

Antes de ejecutar un algoritmo para el 'C5x, hay que inicializar el procesador.

Generalmente, la inicialización tiene lugar en cualquier momento en que se reinicia el procesador. El procesador es reiniciado aplicando un nivel de bajo a la entrada RS; los bits del puntero del vector de interrupción (IPTR) del registro de estado de modo del procesador (PMST) son borrados, mapeando los vectores a la página 0 en el espacio de la memoria de programa. Esto quiere decir que el vector reiniciado siempre reside en la posición de memoria de programa 0. Esta posición normalmente contiene una instrucción de rama para dirigir la ejecución de programa a la rutina de inicialización de sistema. Una reanudación del hardware bona todas las señalizaciones de interrupción pendientes y pone el bit de modo de interrupción (INUN) en ST0, deshabilitando por tanto todas las interrupciones. Un reseteo del hardware también inicializa varios bits de estado y registros periféricos.

Para configurar el procesador después del reseteo, las siguientes funciones internas deben ser inicializadas:

- El procesador de núcleo mapeado en memoria y los registros de control periféricos.
- Estructura de interrupción (bit INTM).
- Control de modo (bits OVM, SXM, PM, AVIS, NDX, TRM).
- Control de memoria (bits RAM, OVLY, CNF).
- Registros auxiliares y puntero del registro auxiliar (ARP).
- Puntero de la página de memoria de datos (DP).

El OVM (el modo de overflow), TC (señal de test/ control), IMR (registro de máscara de interrupción), puntero auxiliar de registro (ARP), el buffer de registro auxiliar (ARB), y el puntero de la página de memoria de datos (DP) no son inicializados mediante el reseteo.

El ejemplo 2-1 muestra la codificación para inicializar el ' C5x en el siguiente estado de la máquina y para la inicialización realizada después del reseteo del hardware:

- RAM de acceso único interna configurada como memoria de programa.
- Tabla cargada en la memoria de programa interna.
- Puntero de la tabla de vector de interrupción (IPTR).
- Bloques de RAM de acceso dual interna llenados con Os.
- Interrupciones habilitadas.

Ejemplo l. Inicialización del TMS320C5x

2.2 Interrupciones.

Los dispositivos 'C5x tienen cuatro interrupciones de usuario externas, enmascaradas, (INT1 - INT4) y una intem1pción no enmascarada (NMI) disponible para dispositivos externos. Las interrupciones internas se generan por los puertos serie, temporizador, y por las instrucciones de interrupción del software (INTR, TRAP y NMI). La estructura de interrupción está descrita en la guía del .usuario TMS320C5x.

Los dispositivos 'C5x son capaces de generar interrupciones software utilizando la instrucción INTR. Esto permite a cualquiera de las 32 rutinas de servicio de interrupción (ISRs) ser ejecutadas desde su software. Los primeros 20 ISRs son reservados para intem1pciones externas, interrupciones periféricas, e implementaciones futuras. Las 12 posiciones restantes en la tabla de vectores de interrupciones pueden ser definidas

por el usuario. La instrucción INTR puede llamar a cualquiera de las 32 interrupciones disponibles en los dispositivos 'C5x.

Cuando una interrupción es ejecutada, ciertos registros de la CPU se guardan automáticamente.

El PC es guardado en una pila hardware de 8 niveles, lo cual sirve también para llamadas de subrutina. Por lo tanto, la CPU soporta las llamadas de subn1tina dentro de un ISR mientras la pila de 8 niveles no sea excedida.

También, hay una pila de 1 nivel (o registro de sombra) para cada uno de los siguientes registros:

- Acumulador (ACC).
- Buffer del acumulador (ACCB).
- Registro de comparación de registro auxiliar (ARCR). .
- Registro de índice (INDX).
- Registro de estado de modo del procesador (PMST).
- Registro de producto (PREG).
- Registro de estado O (STO).
- Registro de estado 1 (ST1).
- Registro de temporización O (TREGO) para multiplicador.
- Registro de temporización 1(TREG 1) para contador de desplazamiento.
- Registro de temporización 2 (TREG2) para test de bit.

Cuando la red de interrupción es tomada, los contenidos de todos estos registros son llevados a una pila de nivel 1, a excepción del bit INUN en STO y el bit XF en ST1 . En una interrupción, el bit INUN está siempre preparado para deshabilitar las interrupciones. Los valores en los registros en el instante de la red de interrupción están todavía disponibles para el ISR pero están también protegidos en los registros de sombra. Los registros de sombra son copiados a los registros de la CPU cuando la instrucción RETI o RETE sea ejecutada. -Esta función permite a la CPU ser utilizada por el ISR sin requerir que lo alto del contexto se guarde y restituya en el ISR.

El Ejemplo 2 muestra el uso de la instrucción INTR. El programa principal establece registros auxiliares e invoca la interrupción definida por el usuario número 20.

Mientras el contexto es guardado automáticamente, el ISR está libre para usar cualquier de los registros guardados sin destruir las variables del programa de llamada. La rutina mostrada aquí utiliza la instrucción CRGT para encontrar el valor máximo de las 16 ejecuciones de la ecuación $Y = aX^2 + bX + c$. El AR1 indica los valores X, AR2 para los coeficientes, y AR3 para los resultados de la Y. Para devolver el resultado para la rutina de llamada, todos los registros se recuperan ejecutando una instrucción RETI.

El valor calculado se coloca en el acumulador, y un retorno estándar es ejecutado porque la pila está ya saltada.

Ejemplo 2. El uso de la instrucción INTR.

2.3 Pila Software.

El 'C5x tiene una pila hardware interna de 8 niveles que se utiliza para guardar y reemplazar direcciones para las subrutinas y los ISRs. Las instrucciones PUSH y POP pueden acceder a la pila hardware por el acumulador.

Dos instrucciones adicionales, PSHD y POPD, están incluidas en el set de instrucciones a fin de que la pila pueda almacenarse directamente y recuperada de la memoria de datos.

Una pila software puede ser implementada usando la instrucción POPD al principio de cada subrutina para guardar el PC en la memoria de datos.

Entonces, antes del retorno, un PSHD es utilizado para colocar el valor correcto al principio de la pila.

Cuando la pila tiene siete valores almacenados en ella, y dos o más valores son puestos en la pila antes de que otros valores desaparezcan, es necesaria una subrutina que expanda la pila. Una rutina para expandir la pila es mostrada en el Ejemplo 3. En este ejemplo, el programa principal almacena la pila, almacena la posición de memoria de inicio en AR2, e indica a la subrutina que empuje los datos de la memoria en la pila o salte los datos de la pila a la memoria. Si se carga un O en el acumulador antes de llamar la subrutina, entonces la subrutina empuja los datos de la memoria a la pila. Si el acumulador contiene un valor que no sea cero, entonces la subrutina salta los datos de la pila a la memoria.

Como la instrucción CALL utiliza la pila guardar el contador de programa, la subrutina salta este valor en el acumulador y utiliza la instrucción BACC para regresar al programa principal. Esto impide al contador de programa ser almacenado en una posición de memoria. La subrutina del Ejemplo 3 utiliza la instrucción BCNDD (rama condicional atrasada) para determinar si la operación debe ser guardada o restaurada.

Ejemplo 3. Operación de pila software

2.4 Operaciones lógicas y aritméticas

Las siguientes subdivisiones proporcionan ejemplos de operaciones lógicas y aritméticas.

2.4.1 Unidad lógica paralela (PLU)

La PLU proporciona un camino lógico directo a los datos de la memoria sin afectar los contenidos del acumulador o el registro del producto. La PLU permite la manipulación directa de bits en cualquier posición en el espacio de memoria de datos. El operando de la fuente puede ser un valor inmediato largo o el registro de manipulación de bit dinámico (DBMR). La utilización de un valor inmediato largo es particularmente efectiva en la inicialización de posiciones de memoria de datos, incluyendo los registros mapeados en memoria. El uso del DBMR como el operando de la fuente permite la computación del tiempo de ejecución de los operandos.

También disminuye el tiempo de ejecución de instrucción para un ciclo, lo cual puede ser importante para rutinas críticas en el tiempo.

El Ejemplo 4 y el Ejemplo 5 ilustran el uso del PLU para la inicialización y la operación lógica. La subrutina UNPACK (Ejemplo 4) extrae los bits individuales de una palabra simple y los almacena separadamente en un array. La subrutina PACK (Ejemplo 5) hace lo contrario de UNPACK tomando los bits de posiciones diferentes y los empaqueta en una palabra simple. En el Ejemplo 5, se inserta una instrucción NOP en el lazo del bloque de repetición. Un lazo del bloque de repetición debe ser por lo menos de tres palabras de longitud en los dispositivos 'C5x.

Ejemplo 4. Utilizando la PLU para desempaquetar.

Ejemplo 5. Utilizando la PLU para empaquetar.

2.4.2 Instrucciones Multicondicional

El 'C5x incluye instrucciones que prueban múltiples condiciones antes de pasar el control a otra pare del programa. Estas instrucciones son: BCND, BCNDD, CC, CCD, RETC, RETCD, y XC. Estas instrucciones pueden probar las condiciones listadas en la tabla 2-1 individualmente o en combinación con otras condiciones.

Tabla 2-1. Condiciones para Instrucciones de rama, llamada y retorno.

Mnemónicos	**Condición**	**Descripción**
EQ	ACC = O	Acumulador es igual a 0
NEQ	ACC D O	Acumulador no es igual a 0
LT	ACC < O	Acumulador menor que 0
LEQ	ACC D O	Acumulador menor o igual a 0
GT	ACC > O	Acumulador mayor que 0
GEQ	ACC D O	Acumulador mayor o igual a 0
NC	C = O	Bit de carry borrado
C	C = 1	Bit de carry puesto a set
NOV	OV = 0	No overflow del acumulador detectado
OV	OV = 1	Overflow del acumulador detectado
BIO	BIO está bajo	La señal BIO está baja
NTC	TC = 0	Señal Test/ control borrada
TC	TC = 1	Señal Test/ control puesta a set
UNC	nada	Operación incondicional

Se pueden combinar condiciones de cuatro grupos (tabla 2-2). Hasta cuatro condiciones pueden ser seleccionadas; sin embargo, cada una de estas condiciones deben ser de grupos diferentes. No se pueden tener dos condiciones del mismo grupo. Por ejemplo, se puede probar EQ y TC al mismo tiempo pero no NEQ y GEQ. Por ejemplo:

BCND BRANCH, LT, NOV, TC; Si ACC < O, no overflow; y el bit TC a set.

En este ejemplo, las condiciones LT (ACC _ O), NOV (OV = O), y TC (TC = 1) deben ser encontradas por la rama que va a ser tomada.

Tabla 2-2. Grupos para instrucciones multicondicionales

Group 1	Group 2	Group 3	Group 4
EQ	OV	C	TC
NEQ	NOV	NC	NTC
GT			BIO
LT			
GEQ			
LEQ			

Probar el estado de la señal TC es exclusivo para probar el pin BIO.

El código en el Ejemplo 6 prueba simultáneamente la señal de carry (C) y el bit de signo del acumulador para localizar un bit a cero (comienza desde el MSB) en una palabra de 64 bits. Esta palabra de 64 bits podría ser la salida del puerto serie donde el primer 0 indica el inicio.

Ejemplo 6. Utilizando condiciones múltiples con la instrucción BCND.

2.4.3 Búsqueda de algoritmos utilizando CRGT

El Ejemplo 7 muestra cómo encuentran las instrucciones CRGT y RPTB el valor máximo y su posición buscando entre un bloque de datos. El lazo que hay sobre ellas es minimizado utilizando la función de repetición de bloque. El acumulador se inicializa con el valor mínimo posible (8000h) antes de que se entre en el lazo principal de búsqueda.

Para encontrar el valor mínimo, la instrucción CRGT puede ser reemplazada por CRLT, y el acumulador se carga con el valor máximo posible (7FFFh) en lugar de con el más pequeño. El resto del código permanece igual.

2.4.4 Multiplicación de matriz utilizando lazos anidados

El 'C5x proporciona tres tipos diferentes de instrucciones para implementar lazos de código. La instrucción RPT (repetición de una sola instrucción) permite que la siguiente instrucción sea ejecutada N veces.

La instrucción RPTB (repite bloque) ejecuta repetidamente un bloque de instrucciones con la cuenta del lazo determinada por el registro contador de repetición de bloque (BRCR). La instrucción BANZ (rama si AR no es O) es otra forma de implementación para siguientes lazos con la cuenta especificada por un registro auxiliar.

Lazos anidados de tres niveles pueden ser implementados eficazmente por estas tres instrucciones con cada una controlando un lazo. El Ejemplo 8 muestra esta estructura anidada de código para multiplicar la matriz de N-por-N.

La utilización de la instrucción MADS (multiplica y acumula usando BMAR) sirve para cambiar las filas de la matriz A para calcular los elementos del producto de la matriz C.

Ejemplo 7. Utilizando las instrucciones CRGT y CRLT

Ejemplo 8. Utilizando lazos anidados

2.5 Buffers Circulares.

El direccionamiento circular es una característica imp01iante del set de instrucciones del 'C5x. Los algoritmos, como la convolución, correlación, y los filtros de respuesta de impulso finita (FIR) pueden hacer uso de buffers circulares en la memoria. El 'C5x soporta dos buffers concurrentes funcionando mediante los registros auxiliares. Cinco registros mapeados en memoria controlan la operación del buffer circular: CBSR1 , CBSR2, CBER1, CBER2, y CBCR.

Las direcciones de principio y de fin deben ser cargadas en los registros del buffer correspondiente (CBSRx y CBERx) antes de que el buffer circular sea habilitado. También, el registro auxiliar que actúa como un puntero para el buffer debe ser inicializado con el valor adecuado.

El Ejemplo 9 muestra el uso de un buffer circular para generar una onda sinusoidal digital. Una tabla de ondas sinusoidales de 256 palabras se carga en el bloque DARAM B1 de memoria interna de datos desde la memoria externa del programa. Para acceder a la DARAM interna se requiere solamente un ciclo de la máquina. El registro de la dirección de movimiento del bloque (BMAR) se carga con la dirección ROM de la tabla. La instrucción de movimiento de bloque mueve 256 muestras de la onda sinusoidal a la memoria interna de datos, la cual es entonces establecida como un buffer circular.

Las direcciones de principio y fin de este buffer circular son cargadas en los registros correspondientes (CBSR l y CBER 1). El registro auxiliar AR7 también se inicializa para el comienzo de la tabla de onda sinusoidal. La instrucción SAMM nos sirve para actualizar el AR7 porque todo los registros auxiliares están mapeados en memoria en la página de datos 0. Finalmente, el buffer circular # 1 se habilita y el AR7 se mapea a ese buffer. El otro buffer circular se deshabilita.

Ejemplo 9. El uso del direccionamiento circular

Si el tamaño de paso debe ser mayor que 1, hay que ver si una actualización para el registro auxiliar genera una dirección fuera del rango del buffer circular. Esto puede ocurrir si la misma tabla del seno se usa para generar ondas sinusoidales de frecuencias diferentes cambiando el tamaño de paso. El direccionamiento de módulo puede evitar tales problemas. Una forma simple para realizar un direccionamiento de módulo en los dispositivos 'C5x consiste en utilizar las instrucciones APL y OPL. Por ejemplo, para implementar el contador de módulo 256, primero hay que cargar el registro de manipulación de bit dinámico (DBMR) con 255 (valor máximo permitido); cuando el registro auxiliar es actualizado, se le hace un AND con el DBMR y un OR con la dirección de principio del buffer. La dirección de principio del buffer modulo-2 k debe tener Os en los k LSBs.

Por lo tanto, para direccionamientos de módulo 256, los primeros ocho LSBs del registro de inicio deben ser 0 (ver el Ejemplo 10).

Ejemplo 10. El direccionamiento de módulo 256

2.6 Lazos de repetición de instrucción simple (RPT).

El 'C5x proporciona dos tipos diferentes de instrucciones de repetición. La instrucción de repetición de bloque (RPTB) implementa los lazos de código que pueden tener desde 3 a 65 536 palabras de tamaño.

Estos lazos no requieren cualquier ciclo adicional para saltar desde el fin de bloque hasta la dirección de principio de bloque al final de cada iteración. Estos lazos por encima de cero se pueden interrumpir a fin de que puedan ser usados en procesados de fondo sin afectar la latencia de las tareas críticas en el tiempo.

Por otra parte, la instrucción simple de repetición (RPT) realiza la ejecución de la siguiente instrucción para proporcionar una alta velocidad en el modo de repetición. Un registro contador de repetición de 16 bits (RPTC) permite la ejecución de una sola instrucción 65 536 veces.

Cuando esta característica de repetición es utilizada, la instrucción que está siendo repetida es retomada sólo una vez. Como consecuencia de esto, muchas instrucciones multiciclo, como MAC/ MACD, BLDD/ BLDP, o TBLR/ TBLW, pasan a ser de un solo ciclo cuando sean repetidas.

Algunas de las instrucciones del 'C5x se comportan de manera diferente en la repetición de una sola instrucción para utilizar de un modo eficiente la arquitectura del bus múltiple del 'C5x. Las siguientes instrucciones pertenecen a esta categoría:
BLDD, BLDP, BLPD, IN, OUT, MAC, MACD, MADS, MADD, TBLR, TBLW,
LMMR, SMMR

Como la instrucción es retomada y cerrada internamente cuando está en el modo de repetición de una instrucción, el bus de programa es usado por estas instrucciones para leer o escribir un segundo operando en paralelo a las operaciones que están siendo usadas por el bus de datos.

Con la instrucción cerrada para una ejecución repetida, el contador del programa se carga con la dirección del segundo operando (el cuál puede estar en forma de datos, programa, o espacio I/O) y se incrementa en sucesivas ejecuciones para lectura/ escritura en posiciones sucesivas de memoria. Por poner un ejemplo, la instrucción MAC toma el multiplicando de la memoria de programa por el bus de programa.

Simultáneamente con la toma del bus de programa, el segundo multiplicando se toma de la memoria de datos por el bus de datos.

Además de estas tomas de datos, la preparación está hecha para accesos en el siguiente ciclo incrementando el contador de programa e indexando el registro auxiliar. La instrucción IN es otro ejemplo de una instrucción que se aprovecha de transferencias simultáneas de datos en el programa y en los buses de datos. En este caso, los valores de datos de posiciones sucesivas en el espacio *VO* pueden ser leídos y transferidos a la memoria de datos.

Desde el Ejemplo 11 hasta el 2-17 se demuestra la implementación de los movimientos de bloque de memoria a memoria en el 'C5x usando lazos de repetición de una instrucción simple (RPT). No hay instrucciones simples para mover datos de memoria a memoria.

Ejemplo 11. Movimiento de bloques de memoria a memoria utilizando RPT con BLDD.

Ejemplo 12. Movimiento de bloques de memoria a memoria utilizando RPT con BLDP.

Ejemplo 13. Movimiento de bloques de memoria a memoria utilizando RPT con BLPD.

Ejemplo 14. Movimiento de bloques de memoria a memoria utilizando RPT con TBLR.

Ejemplo 15. Movimiento de bloques de memoria a memoria utilizando RPT con TBLW.

Ejemplo 16. Movimiento de bloques de memoria a memoria utilizando RPT con SMMR.

Ejemplo 17. Movimiento de bloques de memoria a memoria utilizando RPT con LMMR.

2.7 Subrutinas.

El Ejemplo 18 muestra el uso de una subrutina para determinar la raíz cuadrada de un número de 16 bits. La rutina principal se ejecuta en el punto donde la raíz cuadrada del número es necesaria. En este punto, se realiza una llamada con retardo (CALLD) a la subrutina, transfiriendo el control a esa parte de la memoria del programa para la ejecución y regresando entonces la rutina de llamada mediante la instrucción de retorno con retardo (RETD) cuando la ejecución ha sido completada.

El Ejemplo 18 muestra varias características del set de instrucciones del 'C5x. En particular, se nota el uso de las instrucciones llamada con retardo (CALLD), retorno con retardo (RETD), y ejecución condicional (XC). Debido a la estructura de cuatro niveles en los dispositivos 'C5x, las instrucciones de rama normal requieren cuatro ciclos para ejecutarse. Utilizando ramas con retardo, sólo se requieren dos ciclos para la ejecución. La instrucción XC es útil donde sólo una o dos instrucciones van a ser ejecutadas condicionadamente.

En este ejemplo, se nota cómo XC se utiliza para evitar un ciclo adicional debido a la instrucción de rama. El uso de la instrucción XC también ayuda a conservar el tiempo de ejecución de una constante de rutina, a pesar de las condiciones de entrada. Esto es porque XC ejecuta NOPs en lugar de las instrucciones si no se dan las condiciones.

Se puede ver que la restauración se ha hecho con la instrucción LST para prevenir al ARP de ser sobrescrito. Si se utiliza el direccionamiento indirecto, entonces el orden es invertido.

Ejemplo 18. Cálculo de la raíz cuadrada utilizando la instrucción XC.

2.8 Aritmética de precisión extendida.

El análisis numérico, los cálculos en coma flotante, u otras operaciones pueden requerir la aritmética para ejecutarse con más que 32 bits de precisión. Desde que los dispositivos 'C5x son procesadores de punto fijo de 16/32 bits, el software se requiere para la precisión extendida de operaciones aritméticas. Las subrutinas que realizan las funciones de aritmética de precisión extendida para los 'C5x están provistas en los ejemplos de este capítulo. La técnica consiste en realizar la aritmética por partes.

El 'C5x tiene varias características que ayudan a hacer cálculos de precisión extendida más eficientes. Una .de las características es el bit de carry.

El bit de carry se ve afectado por todas las operaciones aritméticas del acumulador, incluidas la suma y la resta con el buffer del acumulador. Esto permite operaciones aritméticas de 32 bits de longitud utilizando el buffer del acumulador como el segundo operando.

El bit de carry se ve también afectado por las instrucciones de rotación y desplazamiento del acumulador.

También puede ser explícitamente modificado por el registro de estado de carga ST1 y las instrucciones del bit de control set/ reset.

Para una operación correcta, el bit de modo overflow debería ser reseteado (OVM = 0) a fin de que el resultado del acumulador no se cargue con el valor de saturación.

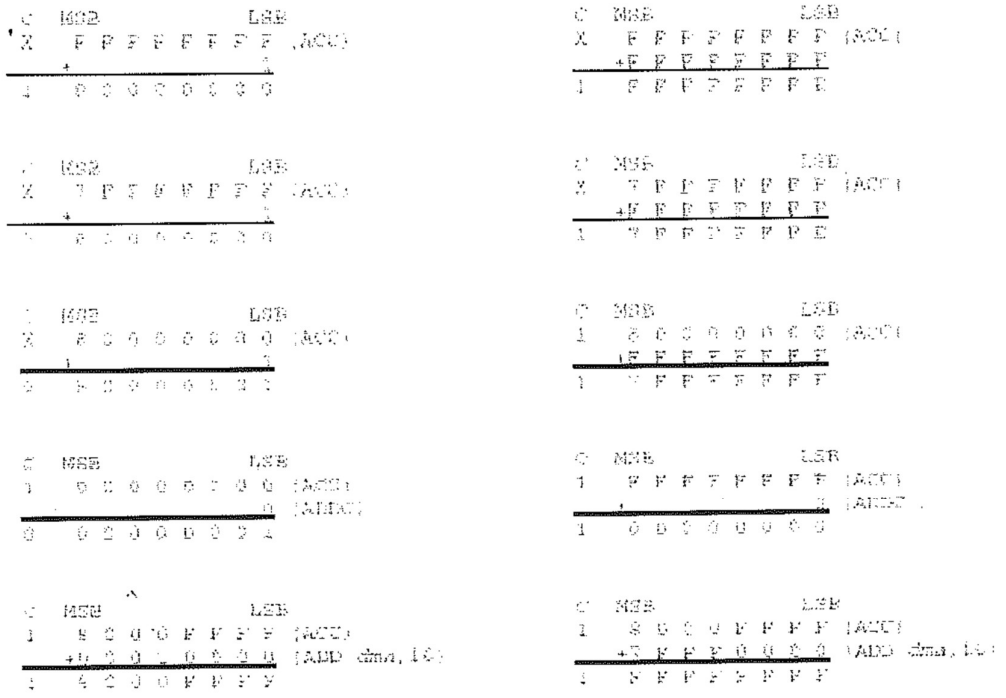
2.8.1 Suma

El bit de carry está a set (C = 1) cuando sea que el desplazador de escala de la entrada, el registro del producto (PREG), o el valor del buffer del acumulador añadido al contenido del acumulador genere un carry de salida del bit 31. De otra manera, el bit de carry se resetea (C = 0) porque el carry de salida del bit 31 es un O. Una excepción para este caso es la suma al acumulador con un desplazamiento de 16 instrucciones (ADD dma, 16), lo cual sólo puede poner a set el bit de carry. Esto permite al ALU generar un carry simple adecuado cuando la suma bien a la mitad inferior o a la mitad superior del acumulador

causa el carry. La figura 2-1 demuestra el significado del bit de carry para las sumas o adiciones.

El Ejemplo 19 muestra una implementación de dos números de 64 bits sumados a cada uno para obtener un resultado de 64 bits.

Figura 2-1. Suma de 32 bits.



Ejemplo 19. Suma de 64 bits.

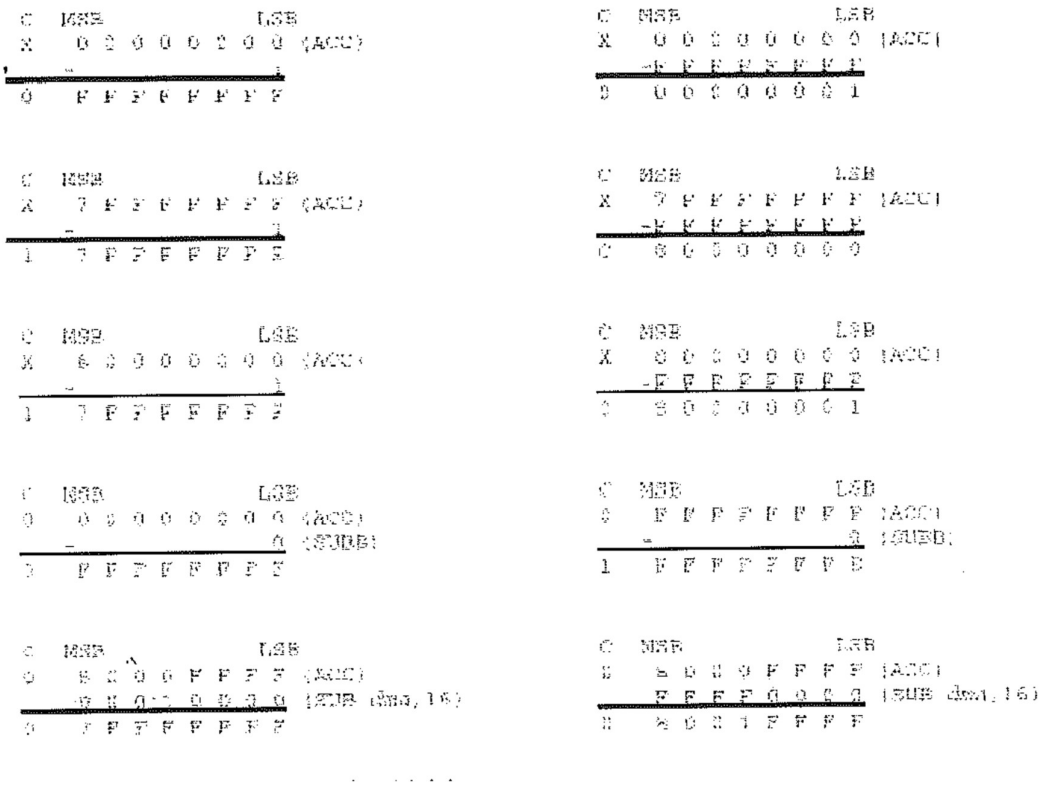
2.8.2 Resta

El bit de carry se resetea (C = O) cuando el desplazador de escala de la entrada, el PREG, o el valor del buffer del acumulador restado del contenido del acumulador genera una ayuda en el bit 31. De otra manera, el bit de carry está a set (C = 1) porque no se requiere ayuda en el bit 31. Una excepción para este caso es la instrucción SUB dma, 16, que sólo puede resetear el bit de carry. Esto permite al ALU generar un carry simple adecuado cuando la resta bien de la mitad inferior o de la mitad superior del acumulador causa la ayuda. La figura 2-2 demuestra el

significado del bit de carry para la resta.

El Ejemplo 20 muestra una implementación de dos números de 64 bits restados de cada uno. La ayuda se genera dentro del acumulador para cada una de las partes de 16 bits de la resta.

Figura 2-2. Resta de 32 bits



Ejemplo 20. Resta de 64 bits.

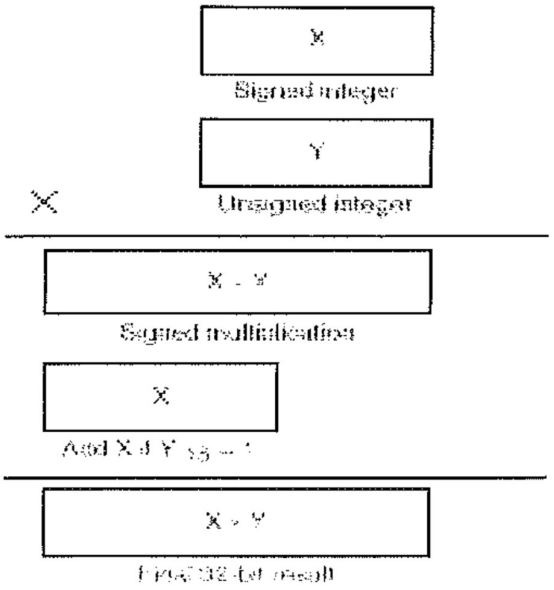
2.8.3 Multiplicación

Otra característica importante que ayuda en los cálculos de precisión extendida es la instrucción MPYU (multiplica sin signo). La instrucción MPYU permite que dos números de 16 bits sin signo sean multiplicados y el resultado de 32 bits se coloque en el PREG en un solo ciclo. La eficiencia se obtiene por generación de productos parciales de partes de 16 bits de un valor de 32 bits o mayor, en lugar de tener que desdoblar el valor en partes de 15 bits o más pequeñas.

La mayor eficiencia se obtiene usando el buffer del acumulador para fijar resultados parciales, en lugar de utilizar una posición temporal en la memoria de datos. La habilidad de los dispositivos 'C5x para desplazar el acumulador de 1 a 16 bits en sólo un ciclo es también útil para escalar y justificar operandos.

Para la multiplicación de enteros de 16 bits, en la que un operando es un entero con signo complemento a 2 y el otro es un entero sin signo, el algoritmo mostrado en la figura 2-3 puede ser utilizado.

Figura 2-3. Algoritmo de multiplicación de un entero de 16 bits.



Pasos requeridos:

1) Multiplicar dos operandos X e Y como si fueran enteros con signo.
2) Si el MSB del entero sin signo Y es 1, sumar X a la mitad superior del producto con signo de 32 bits.

El factor de corrección debe sumarse al resultado de la multiplicación con signo porque el peso del bit del MSB de cualquier entero sin signo de 16 bits es 2^{15}

Lo siguiente es una representación de un entero con signo X y un entero sin signo Y:

$$X = - 2^{15} X_{15} + 2^{14} X_{14} + 2^{13} X_{13} + ... + 2^{1} X_{1} + 2^{0} X_{0}$$

$$Y = - 2^{15} Y_{15} + 2^{14} Y_{14} + 2^{13} Y_{13} + ... + 2^{1} Y_{1} + 2^{0} Y_{0}$$

La multiplicación de X por Y será:

$$X*Y=X*(2^{15}y_{15}+ 2^{14}y_{14}+2^{13}y_{13}+\ldots+2^{1}y_{1}+2^{0}y_{0}) = 2^{15}y_{15}X + 2^{14}y_{14}X + 2^{13}y_{13}X+\ldots+2^{1}y_{1}X+2^{0}y_{0}X(1)$$

Sin embargo, si la X y la Y se consideran enteros con signo, su multiplicación queda así:

$$X*Y=X * (- 2^{15} y_{15} 1s + 2^{14} y_{14} + 2^{13} y_{13} + ... + 2^{1} y_{1} + 2^{0} Y_{0}) = - 2^{15} y_{15} X + 2^{14} y_{14} X + 2^{13} y_{13} X + ... + 2^{1} y_{1} X + 2^{0} Y_{0} X (2)$$

La diferencia entre las ecuaciones (1) y (2) y está en el primer término de la parte derecha de las dos ecuaciones.

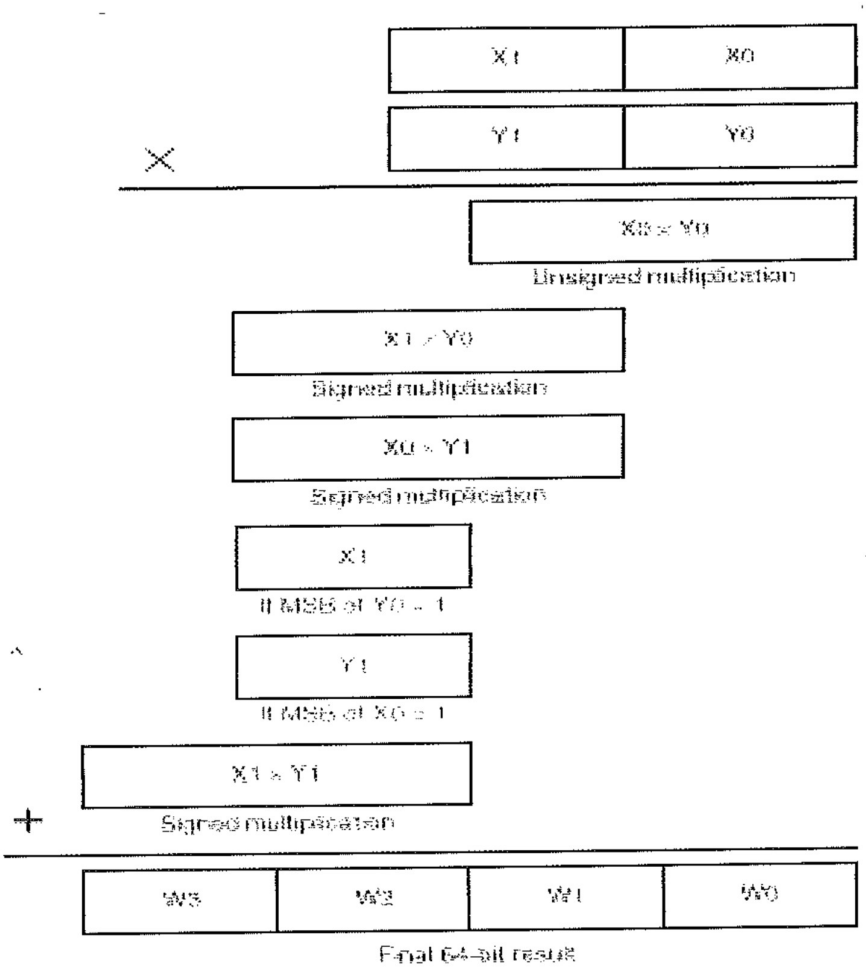
Por lo tanto, si añadimos el término de corrección, $2^{16}y_{15} X$, a la ecuación (2), el resultado sería idéntico que para la ecuación (1) y es el resultado correcto.

Este método de multiplicar un entero con signo por un entero sin signo puede utilizarse para implementar multiplicaciones de precisión extendida en el 'C5x. La figura 2-4 muestra un algoritmo de multiplicación de 32 bits basado en este método. El Ejemplo 21 implementa este algoritmo. El producto es un número entero de 64 bits .

Aquí se ve el uso de las instrucciones BSAR y XC.

El Ejemplo 22 realiza multiplicación fraccionada. Los operandos están en formato Q31, mientras el producto está en formato Q30.

Figura 2-4. Algoritmo de multiplicación de 32 bits



Ejemplo 21. Multiplicación de enteros de 32 bits.

Ejemplo 22. Multiplicación fraccional de 32 bits

2.8.4 División

La división entera y fraccionada se implementa en el 'C5x por restas repetidas ejecutadas con SUBC, una instrucción especial condicional de sustracción. Con un dividendo y un divisor positivos de 16 bits, la repetición de la orden SUBC 16 veces produce un cociente de 16 bits en el acumulador bajo y un resto de 16 bits en el acumulador alto.
SUBC implementa división binaria del mismo modo que se hace la división larga (figura 2-5). El dividendo se desplaza hasta que la resta del divisor

produce un resultado negativo. Para cada resta que no produce una respuesta negativa, se mete un 1 en el LSB del cociente y entonces se cambia de posición. El desplazamiento del resto y el cociente después de cada resta produce la separación del cociente y del resto en las mitades bajas y altas del acumulador, respectivamente.

El dividendo y el divisor deben ser positivos al usar la orden SUBC. Así, el signo del cociente debe estar determinado y el cociente calculado usando el valor absoluto del dividendo y el divisor.

La división de enteros puede ser implementada con la instrucción SUBC, como se muestra en el Ejemplo 23. Para la división de enteros, el valor absoluto del numerador debe ser mayor que el valor absoluto del denominador.

La división fraccionada también puede ser implementada con la instrucción SUBC como se muestra en el Ejemplo 24. Al implementar un algoritmo de división, es importante saber si el cociente puede ser representado como una fracción y el grado de precisión para el que el cociente va a ser calculado.

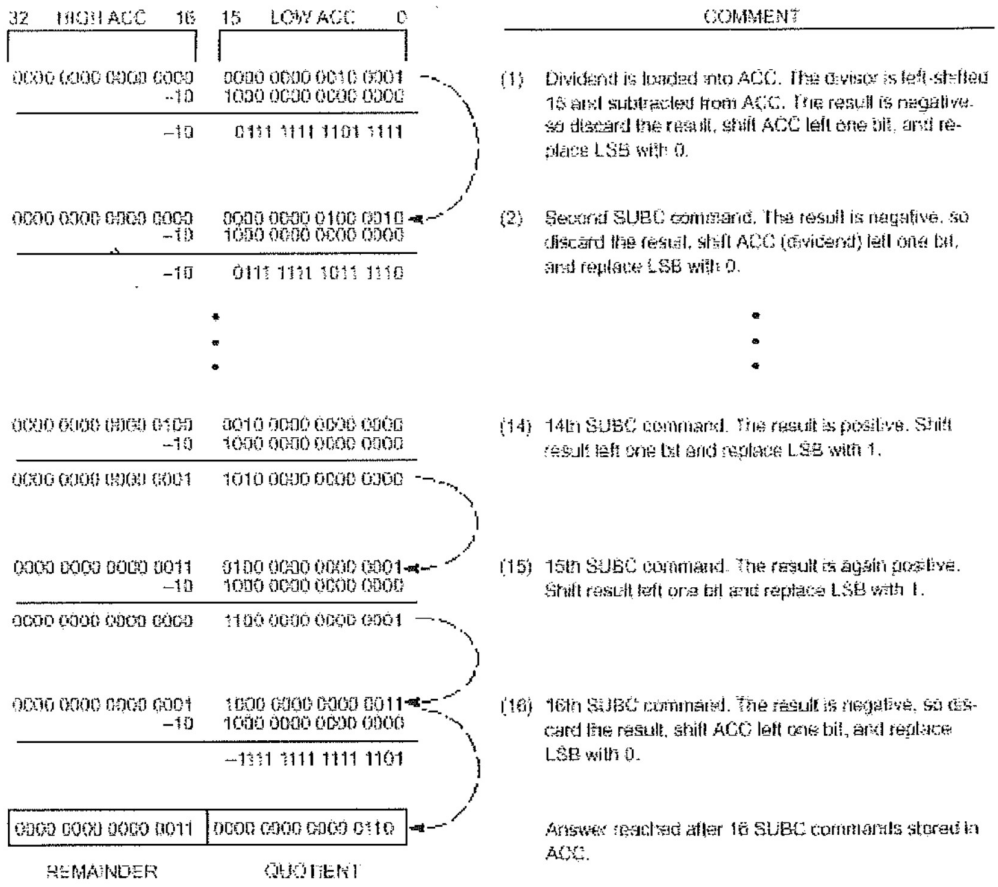
Para la división fraccionada, el valor absoluto del numerador debe ser menor que el valor absoluto del denominador. El dividendo se carga en el acumulador alto y sólo son requeridas N — 1 iteraciones para una fracción de N-bits.

Figura 2-5. División entera de 16 bits

LONG DIVISION:

```
                    000 0000 0000 0110   QUOTIENT
0000 0000 0000 0101 ) 000 0000 0010 0001
                      -101
                       110
                      -101
                         11   REMAINDER
```

SUBC METHOD:



COMENTARIO

(1) El dividendo está cargado en ACC. El divisor se desplaza 15 a la izquierda y restado del ACC. El resultado es negativo, así que se descarta el resultado, se desplaza el ACC un bit a la izquierda, y se reemplaza el LSB por O.

(2) Segundo comando SUBC. El resultado es negativo, así que se descarta el resultado, se desplaza el ACC (dividendo) un bit a la izquierda, y se reemplaza el LSB por O.

(14) Decimocuarto comando SUBC. El resultado es positivo. El resultado se desplaza un bit y se reemplaza el LSB por 1.

(15) Decimoquinto comando SUBC. El resultado es otra vez positivo. El resultado se desplaza un bit y se reemplaza el LSB por 1.

(16) Decimosexto comando SUBC. El resultado es negativo, así que se descarta el resultado, se desplaza el ACC un bit a la izquierda, y se reemplaza el LSB por 0.

La respuesta obtenida después de 16 comandos SUBC se almacena en el ACC.

Ejemplo 23. División de enteros de 16 bits usando la instrucción SUBC

Ejemplo 24. División fraccionada de 16 bits usando la instrucción SUBC

2.9 Aritmética de coma flotante.

Para implementar aritmética de coma flotante en el 'C5x, los operandos deben ser convertidos a punto fijo para operaciones aritméticas y entonces convertido de nuevo a coma flotante. La conversión a la notación de coma flotante se realiza normalizando los datos de entrada.

Para multiplicar dos números de coma flotante, las mantisas se multiplican y los exponentes se suman. La mantisa resultante debe ser renormalizada La suma o resta en coma flotante requiere intercambiar la mantisa a fin de que los exponentes de los dos operandos coincidan. La diferencia entre los exponentes se usa para desplazar a la izquierda el operando de la potencia menor antes de sumar. Entonces, el resultado de la suma debe ser renonnalizado.

Las instrucciones 'C5x usadas en operaciones de coma flotante son NORM, SATL, SATH, y XC. NORM puede usarse para convertir números de punto fijo a números de coma flotante. SATL en combinación con SATH proporciona un desplazamiento a la derecha de 0 a 31 bits de 2 ciclos. XC ayuda a evitar ciclos adicionales provocados por las instrucciones de la rama.

El Ejemplo 25 y el Ejemplo 26 muestran cómo implementar la aritmética de coma flotante en el 'C5x.

Los números de coma flotante están generalmente representados por valores de mantisa y exponente. Los números de coma flotante IEEE de precisión simple se representan por una mantisa de 24 bits, un exponente de 8 bits, y un bit de signo. Para simplificar las rutinas, se utiliza un formato ligeramente diferente del formato IEEE. Se ocupan cuatro palabras por cada número de coma flotante. Una palabra de signo, una palabra para el exponente, y dos palabras para la mantisa se reservan en la memoria como se muestra en los ejemplos.

Ejemplo 25. Suma de coma flotante usando las instrucciones SATL y SATH

Ejemplo 26. Multiplicación en coma flotante usando la instrucción BSAR

2.10 Operaciones orientadas a aplicaciones.

Las siguientes subsecciones proporcionan operaciones orientadas en aplicaciones para:

- Aplicaciones de módems.
- Filtrado adaptable.
- Filtros de respuesta de impulso infinita (IIR).
- Programación dinámica.

2.10.1. Aplicación en módems.

Los procesadores de señales digitales son especialmente apropiados para aplicaciones de modems.

Los dispositivos 'C5x con su set de instrucciones y tiempo de ciclo de instrucciones reducido son particularmente efectivos en implementar codificar y descodificar algoritmos. Las características como el direccionamiento circular, bloque de repetición, y almacenamiento de un solo ciclo reducen el tiempo de ejecución de tales rutinas.

El Ejemplo 27 muestra un codificador diferencial y convolucional para un modem de 9600 bits/segundo V.32. La cadena de datos a transmitir está dividida en grupos de cuatro bits de datos consecutivos. Los primeros dos bits en el tiempo Q1 n y Q2 n de cada gn1po están codificados diferencialmente en Y 1 n y Y2 n.

Los dos bits diferencialmente codificados Y1n y Y2n son utilizados como entradas para una subrutina de un codificador convolucional ENCODE, que genera un bit redundante Y0n. Estos cinco bits son empaquetados en una sola palabra por la subrutina PACK.

Ejemplo 27. Codificador V.32 usando el buffer del acumulador.

2.10.2 Filtrado adaptable

Hay muchas aplicaciones prácticas de filtrado adaptable; un ejemplo está en la adaptación o la actualización de coeficientes. Esto puede ser caro y puede consumir mucho tiempo. Las instrucciones MPYA, ZALR, y RPTB en el 'C5x pueden reducir el tiempo de ejecución.

Una manera de adaptar los coeficientes en el 'C5x es el algoritmo del cuadrado de término mínimo dado por la siguiente ecuación:

$$bk(i\_l) \_ bk(i) \_ 2Be(i)x(i-k)$$

donde $e(i) = x(i) - y(i)$

y

$$y(i) \_ \_ \quad \begin{matrix} N-1 \\ k\_o, \\ bk\ x(i-k) \end{matrix}$$

Los errores de cuantificación en los coeficientes actualizados pueden ser minimizados si el resultado es obtenido por redondeo en vez de truncando.

Para cada coeficiente en el filtro en un punto dado en el tiempo, el factor 2Be(i) es una constante. Este factor puede entonces ser computado una vez y almacenado en el TREGO para cada una de las actualizaciones.

Las instrucciones MPYA y ZALR ayudan a reducir el número de instrucciones en el lazo de adaptación principal. Además, la instrucción RPTB (repite bloque) permite al bloque de instrucciones ser repetido sin problemas para la realización de los lazos.

El Ejemplo 28 muestra una rutina que implementa un filtro de respuesta de impulso finita (FIR) de 128 tap y una adaptación LMS de sus coeficientes.

La SARAM del 'C5x puede ser mapeada en el programa y espacios de datos al mismo tiempo colocando las señales del OVLY y control de la RAM a 1. Esta característica puede usarse para localizar la tabla de coeficientes en la SARAM a fin de que se pueda tener acceso a la tabla por las instrucciones MACD y MPY sin modificar la configuración de la RAM. La instrucción MACD requiere que uno de sus operandos esté en el espacio de programa.

Si la dirección de la tabla de coeficientes se determina en el tiempo de marcha, carga el BMAR (registro de la dirección de movimiento del bloque) con la dirección calculada dinámicamente y reemplaza la instrucción MACD

COEFFP, *-con MADD *-.

Ejemplo 28. Filtro FIR adaptable usando instrucciones RPT y RPTB

2.10.3 Filtros de respuesta de impulso infinita (IIR)

Los filtros de respuesta de impulso infinita (IIR) son usados en aplicaciones de procesado digital de la señal. La función de transferencia de un filtro IIR es dada por:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_N z^{-N}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}} = \frac{Y(z)}{X(z)}$$

La figura 2-6 muestra un diagrama de bloques de un filtro IIR de orden N, forma directa, tipo II.

En el dominio de tiempo, un filtro IIR de orden N es representado por las dos ecuaciones diferenciales siguientes:

En el intervalo de tiempo n:

x(n) es la entrada
y(n) es la salida del filtro IIR

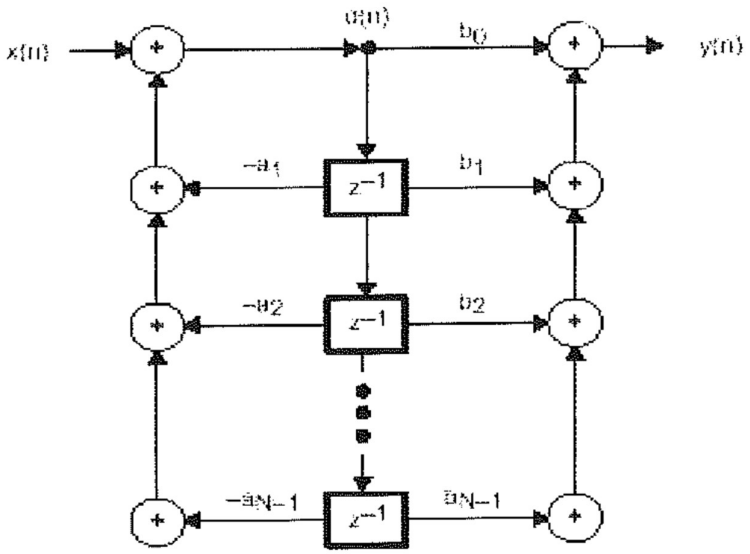
$$d(n) = x(n) - d(n-1)a1 - \ldots - d(n-N+1)aN-1$$

$$y(n) = d(n)bo + d(n-1)b1 + \ldots + d(n-N+1)hN-1$$

Las dos ecuaciones de arriba pueden ser fácilmente implementadas en el 'C5x usando las instrucciones de multiplicar y acumular (MAC, MACD, MADS, MADD). La segunda ecuación también requiere una operación de movimiento de datos para actualizar la secuencia de variables de estado d(n).

El Ejemplo 29 implementa un filtro IIR de orden N usando las instrucciones de repetición de instrucción simple (RPT) y de multiplicar acumular (MAC, MACD).

Figura 2-6. Filtro IIR de orden N, forma directa, tipo II.



Ejemplo 29. Filtro IIR de orden N usando instrucciones RPT y MACD

Debido a la naturaleza recursiva de un filtro IIR, la cuantificación de los coeficientes del filtro puede causar una variación significativa de la respuesta en frecuencia deseada. Para evitar este problema, la función de transferencia deseada del filtro puede ser partida en partes de orden menor que van en cascada u11a con otra. El Ejemplo 30 muestra una implementación de las N partes en cascada IIR de segundo orden. Los coeficientes del filtro y las variables de estado se almacenan en la memoria de datos. Se usan las instrucciones LTD y MPYA para realizar operaciones de multiplicar acumular y movimiento de datos.

Ejemplo 30. Filtro IIR de N cascadas bicuadrado usando las instrucciones LTD y MPYA

2.10.4Programado Dinámico

Las técnicas de programación dinámica son ampliamente utilizadas en algoritmos de búsqueda óptima.

Las aplicaciones como el reconocimiento del habla, las telecomunicaciones, y la robótica usan algoritmos de programación dinámicos. Los dispositivos 'C5x tienen un set de instrucciones para la implementación eficiente de métodos dinámicos de programación.

La mayoría de los algoritmos de búsqueda de tiempo real usan el principio programador dinámico básico que el camino óptimo final del estado de inicio al estado final siempre pasa a través de un camino óptimo del estado de inicio a un estado intermedio.

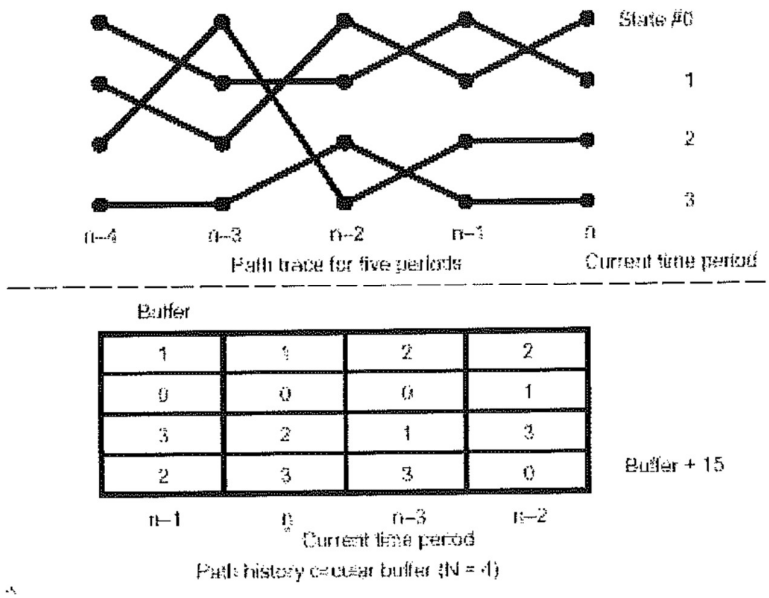
Identificar caminos intermedios reduce una búsqueda larga, de mucho tiempo para el cometido final. Una parte integral de cualquier búsqueda óptima basada en el principio dinámico de programación es la operación de backtracking. El backtracking es necesario para trazar el camino óptimo cuando se cumple el estado final.

El Ejemplo 31 muestra una implementación del algoritmo de backtracking en el cual el camino consiste en cuatro caminos independientes para N periodos de tiempo. Este camino se guarda en un buffer circular. Después de cada operación de backtracking, el camino se actualiza por un algoritmo de búsqueda para el siguiente período de tiempo. El buffer del camino es mostrado en la figura 2-7 para N igual a 4. Cada grupo de cuatro posiciones consecutivas de memoria en el buffer corresponde a la expansión de los cuatro caminos por un nodo (o por un periodo de tiempo). Cada elemento de un grupo corresponde a uno de los cuatro estados en ese periodo de tiempo_ Además, cada elemento de un grupo apunta hacia un elemento en el periodo de tiempo previo que forma parte de ese camino.

Usando el buffer del camino mostrado en la figura 2-7, el elemento correspondiente al estado # 0 en el actual periodo de tiempo contiene un 1.
Esto apunta al segundo elemento del periodo de tiempo previo que contiene un 0. De este modo, comenzando por el periodo de tiempo actual y usando punteros para retroceder en el tiempo, este camino es trazado como 1-0-2-1. Este simplificado backtracking se torna aquí para ilustrar las técnicas de programado del 'C5x. La mayoría de aplicaciones reales requerirían más algoritmos complicados de backtracking.

Ejemplo 31. Algoritmos backtracking usando direccionamiento circular

Figura 2-7. Backtracking Con Historia del Camino



2.11 Transformadas de Fourier.

Las transformadas de Fourier son una herramienta importante a menudo usada en sistemas de procesamiento de señales digitales. El propósito de la transformada es convertir la información del dominio del tiempo al dominio de la frecuencia. La transformada inversa de Fourier convierte la información al dominio del tiempo del dominio de la frecuencia. Las implementaciones computacionalmente eficientes de la transformada de Fourier son conocidas como transformadas rápidas de Fourier (FFT).

El 'C5x reduce el tiempo de ejecución de todas las FFTs en virtud de su tiempo de ciclo de instrucción de 50ns. También, el modo de direccionamiento de bit invertido ayuda a reducir el tiempo de ejecución para FFTs de radio 2. Como se demuestra en la figura 2-8 y figura 2-9, las entradas o las salidas de una FFT no están en orden secuencial. Este revoltijo de las posiciones de datos es un resultado directo de la derivación de la FFT de radio 2. La observación de las figuras y la relación del direccionamiento de la entrada y de la salida revelan que el índice de la dirección está en orden de bit invertido, como se muestra en la tabla 2-3.

Como consecuencia, la secuencia de datos de entrada o la secuencia de datos de salida deben estar mezcladas en asociación con la ejecución de la FFT. En el ejemplo 2-32, los datos de entrada se confunden antes de la ejecución del algoritmo de la FFT a fin de que la salida está en orden.

Figura 2-8. Una FFT DIT en lugar con salidas en orden y entradas de bit invertido

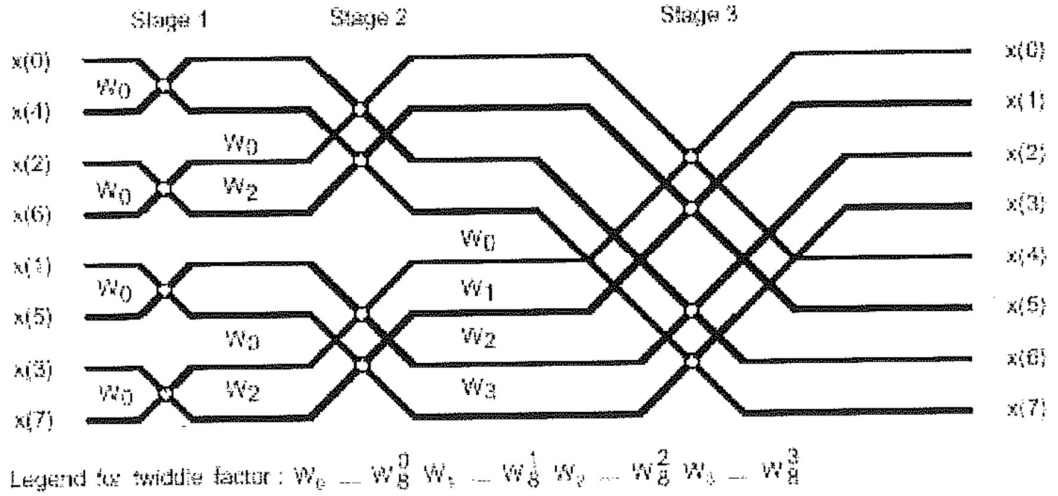
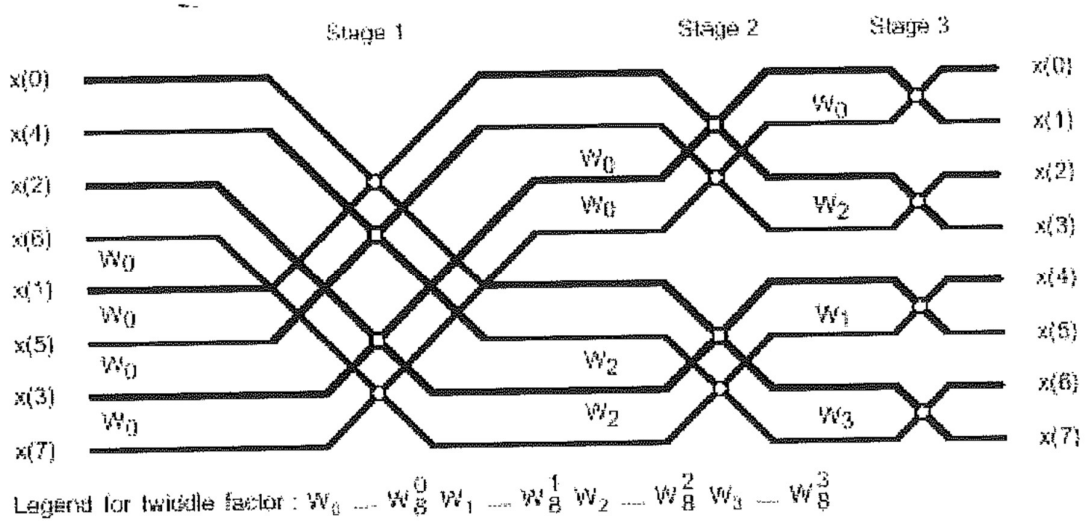


Figura 2-9. Una FFT DIT en lugar con entradas en orden y salidas de bit invertido



lugar de eso, es propagado en la dirección inversa. El resultado es una confusión en el acceso a la dirección.

El procedimiento para generar la secuencia de dirección de bit invertido es cargar INDX con un valor correspondiente a la mitad de la longitud de la FFT y cargar otro registro auxiliar -por ejemplo, AR1 -con la dirección base de la tabla de datos. Sin embargo, las implementaciones de las FFTs implican aritmética complicada; como consecuencia, dos posiciones de memoria de datos (una real y una imaginaria) son asociadas con cada muestra de datos. Para la facilidad de direccionamiento, las muestras son almacenadas en la memoria del espacio de trabajo a pares con la parte real en las posiciones de direcciones pares y la parte imaginaria en las posiciones de dirección impares. Esto quiere decir que el offset de la dirección base para cualquier muestra dada es dos veces el índice de la muestra. Si los datos entrantes están en la siguiente forma:

XR (0), XR (1), ..., XR (7), XI (0), XI (1), ..., XI (7)
donde:

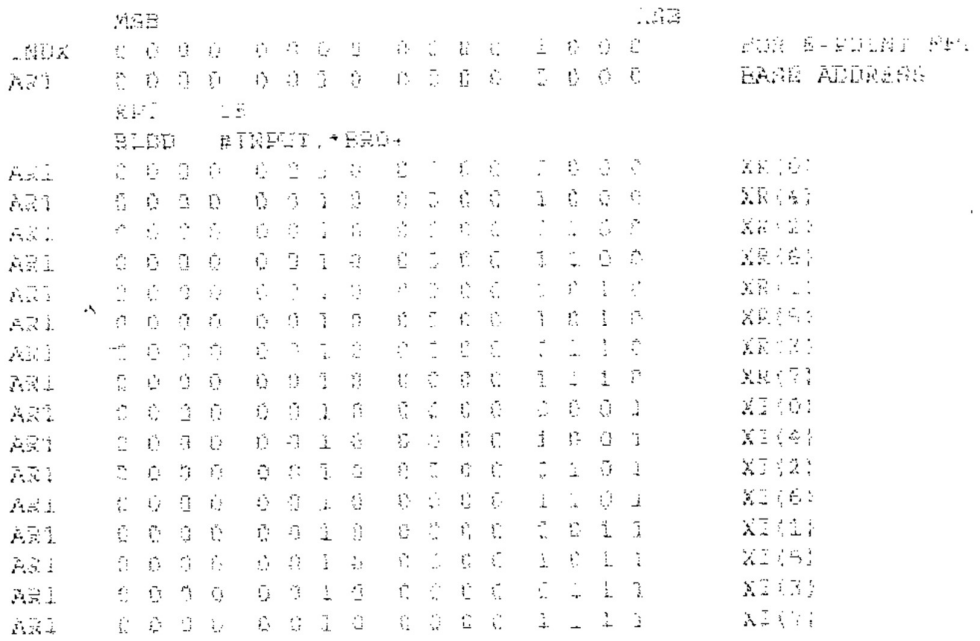
XR -componente real de la muestra de entrada
XI - componente imaginaria de la muestra de entrada entonces es fácilmente transferida en la memoria de datos y almacenada en el orden confuso:

XR {D), XI (O), XR (4), XI (4), XR (2), XI (2), ..., XR (7), XI (7)

cargando el registro INDX con el tamaño de la FFT y usando direccionamiento de bit invertido para guardar cada palabra de entrada.

El Ejemplo 33 muestra los contenidos del registro auxiliar AR I cuando INDX es inicializado con un valor de 8.

Ejemplo 33. Direccionamiento de bit invertido para una FFT



El Ejemplo 34 provee macros para una FFT de 16 puntos.

El Ejemplo 35 provee una rutina de inicialización.

Ejemplo 34. Macros para una FFT DIT de 16 puntos

C:TARJETA DE ADQUISICIÓN DE DATOS.

I. INTRODUCCIÓN.

A continuación se describen los dispositivos 6024E, lo que se necesita para comenzar, desempacar instrucciones, y describir el software opcional y el equipo.

- Características del 6024E.

El 6024E caracteriza 16 canales de entrada analógicos, dos canales de salida analógicos, un conector de 68 pines y ocho líneas de TI O digitales.

Estos dispositivos usan el controlador de tiempo del sistema DAQ-STC de National Instruments para funciones relacionadas con el tiempo. El DAQ-STC consta de tres gn1pos cronometradores que controlan la entrada analógica, la salida analógica y funciones de contador/ temporizador de usos generales. Estos grupos incluyen un total de siete contadores de 24 bits y tres de 16 bits y una resolución máxima de temporización de 50 ns. El DAQ-STC hace posible aplicaciones tales como la generación moderada de pulso, el tiempo de muestreo equivalente, y libre de irregularidades cambiantes de la tasa de muestreo.

Con muchos dispositivos DAQ, se puede sincronizar fácilmente varias funciones de medida para un disparo común o un evento cronometrado. Estos dispositivos tienen el bus de integración de tiempo real (RTSI) para solucionar este problema. En un sistema PCI, el bus RTSI consta de la interfaz del bus de National Instruments RTSI y un cable en cinta para encaminar el cronometrado y las señales de disparo entre diferentes funciones hasta cinco dispositivos DAQ en el ordenador. En un sistema PXI, el bus RTSI consta de la interfaz del bus RTSI de National Instruments y de las señales disparadas PXI en el PXI para encaminar el cronómetro y disparar señales entre diferentes funciones hasta siete dispositivos DAQ en el sistema.

Estos dispositivos pueden interactuar para un sistema a fin de que se puedan adquirir señales analógicas de termopares, RTDs, fuentes de tensión y de corriente. También se pueden adquirir o generar señales digitales para la comunicación y el control.

- Lo que se necesita para comenzar:

Los siguientes dispositivos:
- PCI-6024E
- DAQCard-6024E

Uno de los siguientes paquetes informáticos y su documentación:
- LabVIEW para Windows
- VirtualBench
- Measurement Studio
- NI-DAQ para PC Compatibles

Que el ordenador este equipado con uno de lo siguiente:
- Bus PCI para dispositivo PCI
- PXI o chasis CompactPCI y controlador para un dispositivo PXI
- Slot PCMCIA tipo II para dispositivo de tarjeta DAQ

* Siempre hay que instalar el software antes de instalar el dispositivo.

- Elección del software programable.

Al programar el hardware de la DAQ y SCXI, se puede usar una aplicación software de National Instruments u otro ambiente de desarrollo de aplicación (ADE). En uno u otro caso, se usa a NI-DAQ.

- Software de aplicación de National Instruments.

LabVIEW presenta gráficos interactivos, una interfaz del usuario del estado actual de la tecnología, y un poderoso lenguaje de programación gráfico. La librería de adquisición de datos LabVIEW VI, una serie de instrumentos virtuales para usar a LabVIEW con hardware DAQ de National Instruments, es incluido en LabVIEW. La librería VI de adquisición de datos de LabVIEW es funcionalmente equivalente al software NI-DAQ.

El estudio de la medida, el cual incluye LabWindows/ CVI, herramientas para Visual C++, y herramientas para Visual Basic, es un entorno de desarrollo que permite usar ANSI C, C Visual ++, y Basic Visual para diseñar el test y el software de medida. Para desarrolladores de C,

Measurement Studio incluye a LabWindows/ CVI, un ambiente de desarrollo de aplicación ANSI C totalmente integrado que presenta gráficos interactivos y el LabWindows /CVI Data Acquisition y librerías Fáciles I/ O. Para desarrolladores de Visual Basic, Measurement Studio presenta un set de controles ActiveX para usar el hardware de National Instruments DAQ. Estos controles ActiveX proveen una interface de programación de alto nivel para construir instrumentos virtuales. Para desarrolladores Visual C++, Measurement Studio ofrece un set de clases de Visual C++ y las herramientas para integrar esas clases en aplicaciones de Visual C++. Las librerías, los controles ActiveX, y las clases están disponibles con Measurement Studio y el software NI-DAQ.

VirtualBench presenta instrumentos virtuales que combinan productos DAQ, programas, y el ordenador para crear un instrumento autónomo con el beneficio añadido del procesamiento, representación, y capacidades de almacenamiento del ordenador. Los instrumentos VirtualBench cargan y salvan los datos de la forma de onda en la misma forma para que puedan ser usadas en hojas de cálculos y procesadores de texto. Usando LabVIEW, Measurement Studio, o el software de VirtualBench disminuye en gran medida el tiempo de desarrollo para la adquisición de datos y la aplicación de control.

- Propulsor Software NI-DAQ.

El propulsor software NI-DAQ que viene con el 6024E es compatible el dispositivo. Tiene una librería extensa de funciones que se pueden llamar desde el ambiente de programación de la aplicación. Estas funciones dan permiso para usar todas las características del 6024E. NI-DAQ se ocupa de muchos de los asuntos complicados entre el ordenador y el hardware DAQ tales como programar intem1pciones. NI-DAQ mantiene una interfaz informática consistente entre sus diferentes versiones a fin de que se pueda cambiar plataformas con modificaciones mínimas del código. Mientras se usa LabVIEW, Measurement Studio, u otros lenguajes de programación, los usos de aplicación del software del conductor NI-DAQ, están ilustrados en la siguiente figura.

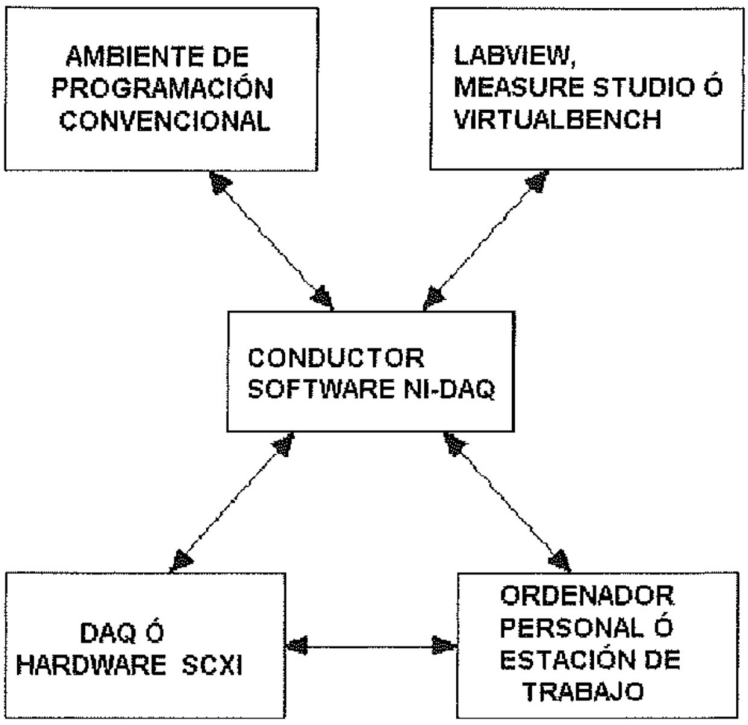


Figura 1. Relación entre el entorno programador, NI-DAQ, y Hardware

-Equipamiento optativo.

National Instruments ofrece una variedad de productos para usar con el dispositivo, incluyendo cables, bloques del conector, y otros accesorios, como los siguientes:

* Cables y conjuntos de cables, blindado y en cinta.
* Bloques de conectores, blindados y terminales atornillados sin blindar.
* Cables de bus RTSI.
* Módulos SCXI y accesorios para aislamiento, amplificando, excitando, y multiplexado de señales para relés y salida analógica. Con SCXI se puede acondicionar y adquirir hasta 3.072 canales.
* Módulos de acondicionamiento de la señal de cuenta de canal bajo, dispositivos, y accesorios, incluyendo el acondicionamiento para los calibres de tensión y RTDs, muestra y retención simultánea, y relés.

2. Instalación y Configuración.

Instalación del Software

Hay que instalar el software antes de instalar el dispositivo. Si se usa el LabVIEW, LabWindows/ CVI, ComponentWorks, o VirtualBench, hay que instalar este software antes de instalar el software del conductor NI-DAQ. Hay que referirse a las notas informáticas de liberación del software para las instrucciones de instalación.

Si se usa NI-DAQ, entonces hay que referirse a las notas de liberación de NI- DAQ. Se busca el capítulo de instalación del sistema operativo y se siguen las instrucciones.

Instalación del hardware.

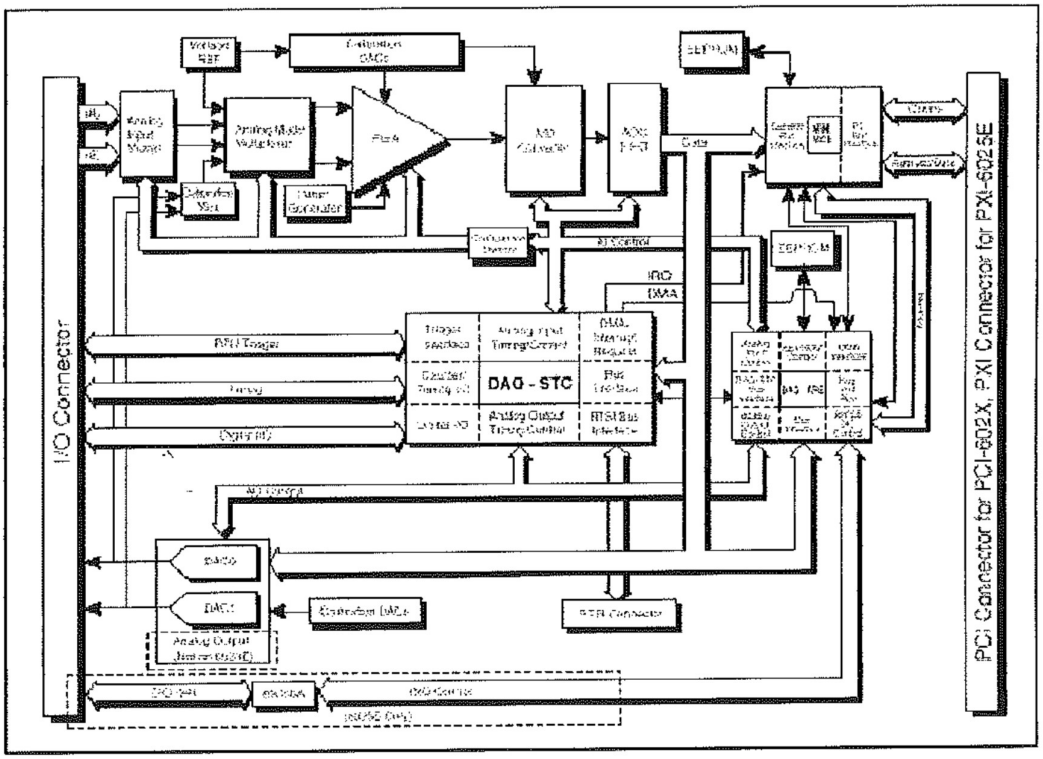
Este se instala después de instalar el software. El dispositivo encajará en cualquier ranura disponible en el ordenador. Sin embargo, para lograr el mejor desempeño del ruido, dejando tanto espacio como sea posible entre el dispositivo y otros dispositivos. Lo que viene a continuación son las instrucciones generales de instalación. Se ha de consultar el manual del usuario del ordenador o el manual de referencia técnico para las instrucciones específicas y las advertencias.

Instalación del dispositivo PCI

1. Apagar y desenchufar el ordenador.
2. Quitar la cubierta superior del ordenador.
3. Quitar la cubierta de la ranura de expansión en el panel posterior del ordenador.
4. Tocar cualquier parte de metal del chasis del ordenador para descargar cualquier electricidad estática que podríamos tener en las ropas ó el cuerpo.
5. Introducir el dispositivo en una ranura *5* V PCI. Balancear suavemente el dispositivo para facilitar su introducción. Puede ser un ajuste forzado, pero no hay que meter el dispositivo a la fuerza en el lugar.
6. Atornillar el corchete de la montura del dispositivo al riel del panel posterior del ordenador.
7. Verificar visualmente la instalación.
8. Colocar la tapa superior del ordenador.
9. Enchufar y encender el ordenador.

3. Visión general del Hardware.

Esta figura muestra un diagrama de bloques para el PCI-6024E.



ENTRADA ANALÓGICA

Modo de entrada

Los dispositivos tienen tres modos diferentes de entrada -entrada simple no referenciada (NRSE), entrada simple referenciada (RSE), y entrada diferencial (DIFF). Las configuraciones de entrada simple proveen solo hasta 16 canales. La configuración de entrada DIFF provee hasta ocho canales. Los modos de entrada están programados para cada para el escaneo multimodo. Por ejemplo, se puede configurar el sistema de circuitos para escanear 12 canales -cuatro canales DIFF y ocho RSE. La siguiente tabla muestra las tres configuraciones de entrada.

Configuración	Descripción
DIFF	Un canal configurado en modo DIFF usa dos líneas de entrada analógicos. Una línea conecta la entrada positiva del amplificador de instrumentación de ganancia programable (PGIA) del dispositivo, y el otro conecta la entrada negativa del PGIA
RSE	Un canal configurado en modo RSE usa una línea de entrada analógica, la cual conecta con la entrada positiva del PGIA. La entrada negativa del PGIA está conectada internamente a la tierra de la entrada analógica (AIGND).
NRSE	Un canal configurado en modo NRSE usa una línea de entrada analógica, la cual conecta con la entrada positiva de la PGIA La entrada negativa de la PGIA está conectada al sensor de la entrada analógica (A.ISENSE).

Rango de entrada

Los dispositivos tienen un rango de entrada bipolar que cambia con la ganancia programable. Se puede programar cada canal con una única ganancia de 0.5, 1.0, 10, o 100 para maximizar la resolución del convertidor de analógico-digital de 12 bits (ADC). Con la ganancia correcta fijada, se puede usar la resolución completa ADC para medir la señal de entrada. La siguiente tabla muestra el rango de entrada y la precisión según la ganancia usada.

Ganancia	Rango de entrada	Precisión
0.5	-10 a +10V	4.88mV
1	-5 a +5 V	2.44mV
10.0	-500 a +500mV	244.14uV
100 0	-50 a +50mV	244.14uV

SALIDA ANALÓGICA

Estos dispositivos proporcionan dos canales de tensión de salida analógica en el conector I/ O. El rango bipolar es fijado en ± 10 V. Los datos escritos para el convertidor digital-analógico (DAC) son interpretados en formato de complemento a dos.

Ráfaga de Salida Analógica

En operación normal, unas ráfagas de salida DAC siempre que se produzcan son actualizadas con un valor nuevo. La energía de la ráfaga difiere de código a código y aparece como la distorsión en el espectro de frecuencia.

Entradas/ Salidas digitales

Los dispositivos contienen ocho líneas de entradas/ salidas digitales (DIO < 0 ..7 >) para usos generales. Se puede configurar individualmente el software tanto para cada línea de entrada o de salida. En el arranque de sistema y la reanudación, los puertos de entradas/ salidas digitales son todos de alta impedancia.

El hardware del control alto/ bajo para los contadores de usos generales O y 1 están conectados a DI06 y DI07, respectivamente. Así, se puede usar DI06 y DI07 para controlar los contadores de usos generales. Las señales de control del alto/ bajo son únicamente de entrada y no alteran la operación de las líneas DIO.

El Dispositivo y los relojes RTSI

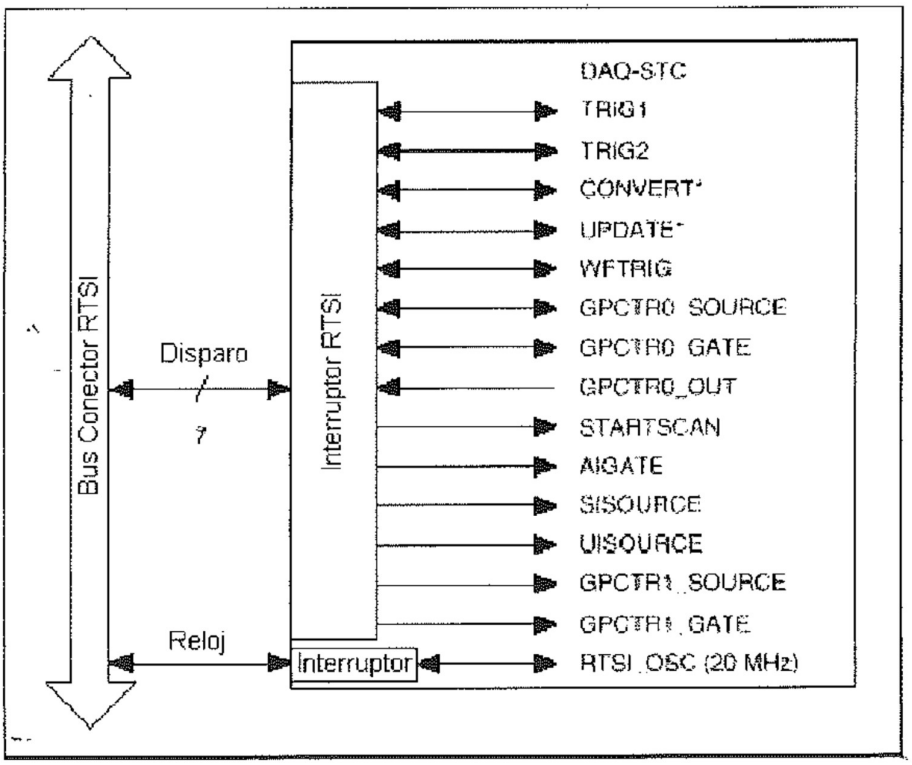
- Bus PCI -.
Muchos funciones del dispositivo requieren una frecuencia de base de tiempos para generar las señales de tiempo necesarias para controlar las conversiones *A*/ D, actualizaciones del DAC, o señales de uso general en el conector I/ O.

Estos dispositivos pueden usar tanto su base de tiempos interna de 20 MHz como una base de tiempos recibida por el bus RTSI. Además, si se configura el dispositivo para usar la base de tiempos interna, entonces también se puede programar el dispositivo para conducir a su base de tiempos interno por el bus RTSI hacia otro dispositivo que está programado para recibir esta señal de base de tiempos. Esta fuente del reloj, ya sea local o del bus RTSI, esta usada directamente por el dispositivo como la fuente primaria de frecuencia. La configuración predeterminada en el arranque es usar la base de tiempos interna sin conducir la señal de la base de tiempos del bus RTSI. Esta base de tiempos es software seleccionable.

Disparo del RTSI

Bus PCI

Las siete líneas de disparo RTSI del bus RTSI proveen un esquema de la interconexión muy flexible para cualquier dispositivo que compartiendo el bus RTSI. Estas líneas bidireccionales pueden conducir y recibir cualquiera de las ocho señales de tiempo por el bus RTSI. El esquema de conexión de la señal se muestra en la siguiente figura.



4. CONEXIÓN DE LAS SEÑALES.

En la siguiente tabla se muestran los cables que se han de usar con los conectores de Entrada/ Salida para conectar diferentes accesorios.

Dispositivo Conector de Entradas/ Salidas	Número de pines	Cable para conectar a Accesorios de 100 pines	Cable para conectar a Accesorios de 68 pines	Cable para conectar a Accesorios de señal de 50 pines
PCI-60241'	68	*N/A*	Cable Apantallado SH6868, Cable en cinta R6868	Cable Apantallado SH6850, Cable en cinta R6B50

Conector de Entradas/ Salidas

Asignación- de pines del conector entrada/ salida del 6024E.

ACH8	34	68	ACH0
ACH1	33	67	AIGND
AIGND	32	66	ACH9
ACH10	31	65	ACH2
ACH3	30	64	AIGND
AIGND	29	63	ACH11
ACH4	28	62	AISENSE
AIGND	27	61	ACH12
ACH13	26	60	ACH5
ACH6	25	59	AIGND
AIGND	24	58	ACH14
ACH15	23	57	ACH7
DAC0OUT	22	56	AIGND
DAC1OUT	21	55	AOGND
RESERVED	20	54	AOGND
DIO4	19	53	DGND
DGND	18	52	DIO0
DIO1	17	51	DIO5
DIO6	16	50	DGND
DGND	15	49	DIO2
+5 V	14	48	DIO7
DGND	13	47	DIO3
DGND	12	46	SCANCLK
PFI0/TRIG1	11	45	EXTSTROBE*
PFI1/TRIG2	10	44	DGND
DGND	9	43	PFI2/CONVERT*
+5 V	8	42	PFI3/GPCTR1_SOURCE
DGND	7	41	PFI4/GPCTR1_GATE
PFI5/UPDATE*	6	40	GPCTR1_OUT
PFI6/WFTRIG	5	39	DGND
DGND	4	38	PFI7/STARTSCAN
PFI9/GPCTR0_GATE	3	37	PFI8/GPCTR0_SOURCE
GPCTR0_OUT	2	36	DGND
FREQ_OUT	1	35	DGND

* Indica que la señal se activa baja.

En las siguientes tablas se muestra la descripción de los conectores de las señales de Entrada/ Salida.

Nombre de la Señal	Referencia	Dirección	Descripción
AIGND	---	---	Tierra de la entrada Analógica-estos pines son el punto de referencia de las medidas en configuración RSE y el punto de retorno de la corriente inversa de las medidas en DIFF.
ACH<0..15>	AIGND	Entrada	Canales de entrada Analógica 0-15. Se puede configurar cada canal, así como una entrada DIFF ó dos de entrada simple.
AISENSE	AIGND	Entrada	Sensor de la entrada Anológica-este pin sirve como nodo de referencia para cualquier canal en NRSE.
DAC0OUT	AOGND	Salida	Salida Analógica del canal 0-este pin proporciona la tensión de salida de la salida analógica del canal 0.
DAC1OUT	AOGND	Salida	Salida Analógica del canal 1.
AOGND	---	---	Tierra de la salida Analógica. Las tensiones de salida analógica estan referenciadas a este nodo.
DGND	---	---	Tierra digital-este pin proporciona la referencia para las señales digitales del conector I/ O así como la fuente de +5V de continua.
DIO<0..7>	DGND	Entrada ó Salida	Señales digitales I/ O.
+5V	DGND	Salida	Fuente de +5V de continua.
SCANCLK	DGND	Salida	Reloj de escaneo- cuando este pin esta habilitado da un pulso en modo escaneo por cada conversión A/ D.
EXTSTROBE	DGND	Salida	Strobe externo.
PFI0/TRIG1	DGND	Entrada Salida	PF10/TRIG1 como entrada es una de las funciones de entrada programable PFIs. Estas se explican en la página 4-30 del manual del PCI-60244E. Como salida, este es la señal TRIG1 (disparo de la entrada analógica.
PFI1/TRIG2	DGND	Entrada Salida	Es uno de los PFIs. Es la señal TRIG2, (disparo de parada de la entrada Analógica).
PFI2/CONVERT	DGND	Entrada Salida	Es uno de los PFIs. Es la señal CONVERSORA de la entrada Analógica.
PFI3/GPCTR1_SOURCE	DGND	Entrada Salida	Es uno de los PFIs. Es la señal GPCTR1_SOURCE, la cual refleja la fuente actual conectada con el contador de usos generales 1.
PFI4/GPCTR1_GATE	DGND	Entrada Salida	Es uno de los PFIs. Es la señal GRCTR1_GATE, la cual refleja la puerta actual conectada al contador de usos generales 1.

Nombre de la señal	Referencia	Dirección	Descripción
GPCTR1_OUT	DGND	Salida	Salida del contador (de usos generales) 1.
PFI5/UPDATE*	DGND	Entrada	Es uno de los PFIs.
		Salida	Señal de UPDATE de la salida Analógica.
PFI6/WFTRIG	DGND	Entrada	Es uno de los PFIs.
		Salida	Es la señal de salida de disparo analógico.
PFI7/STARTSCAN	DGND	Entrada	Es uno de los PFIs.
		Salida	Este pin genera un pulso en el comienzo de cada escaneo de entrada analógica.
PFI8/GPCTR0_SOURCE	DGND	Entrada	Es uno de los PFIs.
		Salida	Esta señal refleja la actual señal defuente conectada al contador de usos generales 0.
PFI9/GPCTR0_GATE	DGND	Entrada	Es uno de los PFIs.
		Salida	Esta señal refleja la actual señal de puerta conectada al contador de usos generales 0.
GPCTR0_OUT	DGND	Salida	Salida del contador (de usos generales) 0.
FREQ_OUT	DGND	Salida	Frecuencia de salida (del generador de frecuencia).

* Indica que la señal se activa baja.

3. PROGRAMAS

Ejemplo 1. Inicialización del TMS320C5x

```
.mmregs
      .ref ISR0,ISR1,ISR2,ISR3,ISR4,TIME
      .ref RCV,XMT,TRX,TXMT,TRP,NMISR
*;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
* Inicialización del procesador para el TMS320C50.
*
* El mapa de memoria de la RAM de simple acceso en el espacio de programa
* y en el espacio de datos es diferente para los dispositivos 'C5x. Por tanto,
* la localización de memoria señalada por la dirección 0800h en el espacio de
* datos es mapeada a una dirección diferente en el espacio de programa para
* los distintos dispositivos 'C5x. Así, el IPTR se debería cargar con el valor
* correspondiente para colocar la tabla de vectores en el espacio de programa
* correcto.
*
*       | C50  | C51  | C53  | C56  | C57
*-------+---------+---------+---------+---------+--------
* PMST | 0081E | 0201E | 0401E | 0801E | 0801E
*
*;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
V_TBL   .sect  "vectors"
RESET   B      INIT          ;Esta parte se cargara en la direccion de
                             ;memoria del programa 0h.
INT0    B      ISR0          ;INT0 - empieza a procesar aqui
INT1    B      ISR1          ;INT1 - empieza a procesar aqui
INT2    B      ISR2          ;INT2 - empieza a procesar aqui
INT3    B      ISR3          ;INT3 - empieza a procesar aqui
TINT    B      TIME          ;procesado de interrupcion del procesador
RINT    B      RCV           ;Puerto serie recibe interrupcion
XINT    B      XMT           ;Puerto serie transmite interrupcion
TRNT    B      TRX           ;Puerto TDM recibe interrupcion
TXNT    B      TXMT          ;Puerto TDM transmite interrupcion
INT4    B      ISR4          ;INT4 - empieza a procesar aqui
        .space  14*16        ;14 palabras
TRAP    B      TRP
NMI     B      NMISR
        .text
INIT    LDP #0               ;Inicializa puntero de datos
        OPL #20h,PMST        ;Configura S/A RAM en memoria de datos
        LAR AR7,#0800h       ;Direccion espacio de datos para tabla vectores
        MAR *,AR7
        RPT #39
```

```
BLPD #V_TBL,*+          ;Carga tabla de vectores en 0800h
SPLK #0081Eh,PMST       ;Configura S/A RAM en espacio de programa
                        ;e inicializa puntero de tabla de vectores
SPLK #01FFh,IMR         ;Borra registro enmascaramiento interrupcion
CLRC OVM                ;Deshabilita modo saturacion overflow
LAR AR7,#60h            ;Inicializa bloque B2
RPTZ #31
SACL *+
LAR AR7,#100h           ;Inicializa bloque B0
RPTZ #511
SACL *+
LAR AR7,#300h           ;Inicializa bloque B1
RPTZ #511
SACL *+
CLRC INTM               ;Habilita interrupciones globalmente
B MAIN_PRG              ;Retorno
```

Ejemplo 2. El uso de la instrucción INTR

```
      .mmregs
TEMP   .set 63h       ;Almacenamiento temporal
X      .set 64h
Y      .set 65h
COEFF  .set 66h
V_TBL  .sect "vectors"
RESET  B INIT         ;Esta parte se cargara en la direccion de memoria
                      ;del programa 0h.
       .space 38*16   ;Skip siguientes 38 localizaciones a interrupcion #20
IN20   B ISR20        :Interrupcion #20 - empieza a procesar aqui
       .text
INIT   LDP #0         ;Inicializa puntero de datos
       LAR AR1,#X     ;AR1 apunta a X los valores
       LAR AR2,#COEFF ;AR2 apunta a los coeficientes b,a,c en ese orden
       LAR AR3,#Y     ;AR3 apunta a Y los resultados
       INTR 20        ;Llama interrupcion software #20
       B $            ;Finaliza el programa
*.....
*
* Esta rutina usa la repeticion de bloque del 'C5x para encontrar el maximo
* valor de las 16 soluciones de la ecuacion Y=aX^2+bX+c. Los valores X los
* apunta AR1. Los resultados Y los apunta AR3. Los coeficientes los apunta
* AR2. Al final de la rutina, ACC contiene el valor maximo.
* AR1, AR2, y AR3 se modifican. Todos los demas registros no se ven afectados.
*
*.....
ISR20  LDP #0         ;Usa pagina 0 de la memoria de datos.
       LACC #08000h
       SACB           ;Inicializa AccB con el valor minimo posible
       MAR *,AR1      ;ARP <- AR1
*
* Carga registro de cuenta de repeticion de bloque con 15.
       SPLK #0Fh,BRCR
*
* Repite Bloque.
       RPTB END_LOOP-1 ;For i=0; i<=15; i++.
       ZAP             ;ACC = PREG = 0
       SQRA *+,AR2     ;TREG0 = X PREG = X^2
       SPL TEMP        ;Guarda X^2.
       MPY *+          ;PREG = b*X
       LTA TEMP        ;TREG = X^2 ACC = b*X
```

```
MPY *+          ;PREG = a*X^2
APAC            ;ACC = a*X^2 + b*X
ADD *,0,AR3     ;ACC = A*X^2 + b*X + c
SACL *+,0,AR1   ;Guarda Y.
CRGT            ;Guarda Y maxima.
END_LOOP
SACL TEMP       ;Guarda el resultado temporalmente
LACC #RE_ENTER
PUSH            ;Hace push de re-entrada de direcciones sobre la pila
RETI            ;Hace un pop a todos los registros
RE_ENTER
LACC TEMP       ;Carga ACC con el valor maximo
RET             ;Regresa al codigo interrumpido
```

Ejemplo 3. Operación de pila software.

```
*
* Esta rutina expande la pila mientras deja al programa principal
* determinando donde almacenar los contenidos de la pila, o de donde los
  tiene
* que reemplazar. La entrada Conditions:
  * ACC = O (reemplaza pila); 1 (guarda pila)
  * AR2 - > Tope de pila en la memoria de datos
  *
  STACK: BCNDD POP,NEQ  ;Rama con retardo si se
  requiere POPD MAR *,AR2  ;Usa AR2 como puntero de
  pila
  POP                          ;Devuelve la direccion
  RPT #6                        ;Repite 7 veces
  PSHD *+                       ;Pone la
  memoria en la pila BACC   ;Regresa al
  programa principal
  POP:              MAR *-          ;Alinea AR2
  RPT #6                       ;Repite 7 veces
  POPD *-                      ;Pone la pila
  en la memoria MAR *+     ;Realinea
  puntero de pila BACC       ;Regresa al
  programa principal
```

Ejemplo 4. Utilizando la PLU para desempaquetar.

```
* PCKD
* ---------------
* |Bn ------ B0|
* ---------------
*
* UNPCKD
* ---------------
* |0 -- 0 |Bn|
* ---------------
* |0 -- 0|Bn-1|
* ---------------
* . . .
* ---------------
* |0 -- 0|B0|
* ---------------
      .mmregs
NO_BITS .set 16              ;Numero de bits empaquetados en la palabra
PCKD    .set 60h             ;Entrada de palabra
UNPCKD  .set 61h             ;Buffer de salida. Cada palabra tendra
                        ;un bit en localizacion LSB.
      .text
UNPACK  LDP #0               ;DP=0
      MAR *,AR0
      LAR AR0,#UNPCKD+NO_BITS-1 ;Fin de tabla de direcciones
      SPLK #NO_BITS-1,BRCR     ;Inicializa el registro de cuenta
      SPLK #1,DBMR             ;Carga mascara en registro DBMR
      LACC PCKD                ;Bits empaquetados -> Acc
      RPTB LOOP-1              ;Empieza looping
      SACL *                   ;Guarda restantes bits empaquetados
      APL *-                   ;Guarda solo el LSB
      SFR                      ;Desplaza a la derecha para eliminar bit
                          ;desempacado
      LOOP RET                 ;Retorno
```

Ejemplo 5.Utilizando la PLU para empaquetar.

```
*
* PCKD
* ---------------
* |Bn ------ B0|
* ---------------
*
* UNPCKD
* ---------------
* |0 -- 0 |Bn|
* ---------------
* |0 -- 0|Bn-1|
* ---------------
* . . .
* ---------------
* |0 -- 0|B0|
* ---------------
      .data
NO_BITS .set 16            ;Numero de bits a empaquetar
PCKD    .set 60h           ;Palabra empaquetada
UNPCKD  .set 61h           ;Array de bits desempacados
      .text
PACK   LAR AR0,#UNPCKD     ;AR0 apunta al inicio de array desempacado
      MAR *,AR0
      LDP #0               ;DP=0
      SPLK #NO_BITS-2,BRCR ;Hace loop NO_BITS-1 veces
      LACC *+              ;Obtiene el MSB
      RPTB LOOP-1          ;Empieza el looping
      SFL                  ;Hace espacio para el siguiente bit
      ADD *+               ;Pone el siguiente bit
      NOP
      LOOP
      SACL PCKD            ;Almacena el resultado
      RET                  ;Retorno
```

Ejemplo 6. Utilizando condiciones múltiples con la instrucción BCND.

```
 LDP #O
 SPLK #63,BRCR                ;No. de iteraciones - 1
 ;Codigo para obtener palabra de entrada de 64-bits y
 ;la carga en ACC y ACCB
 LAR AR0,#0                   ;Inicializa el contador
 de bits RPTB ENDLOOP-1       ;For I=0,1<=63,I++
 SFLB                         ;Desplaza a la izquierda ACC+ACCB, MSB
es
 ;desplazado en la señalizacion del carry
 MAR *+                       ;Incrementa el contador de bits
 BCND ENDLOOP,NC,L T          ;Sale si carry=O y el actual MSB=
1 ENDLOOP:  ;ACC+ACCB contiene ahora datos alineados
 APL #OFFFEh,PMST             ;Borra señalizacion BRAF
```

Ejemplo 7. Utilizando las instrucciones CRGT y CRLT.

```
* Esta rutina busca entre un bloque de datos en la memoria de datos para
* almacenar el valor máximo y la dirección de ese valor en las posiciones
* de memoria MAXVAL y MAXADR, respectivamente. El bloque de datos
    puede
ser
* de cualquier tamaño definido por el registro contador de repetición de
* bloque (BRCR).
    *
    * Instrucciones llave 'C5X:
    *
    * RPTB Repite un bloque de código como fue definido por el contador
      de
    * repetición BRCR.
    * CRGT Compara el ACC con el ACCB. Almacena el valor mayor en los
    ACC y
ACCB,
    * pone a set el bit de CARRY si el valor es mayor que el valor mayor
    * previamente encontrado.
    * XC Ejecuta condicionalmente (1 o 2 palabras) si el bit de CARRY est
      a set.
    *
    MAXADR  .set 60h
MAXVAL   .set 6 lh
       .1mmegs
       .text
       LDP #O           ;Apunta a página de datos O
       LAR AR0,#0300h    ;AR= dirección memoria datos
           SETC SXM      ;Pone a set el modo de
           extensión de signo LACC #08000h       ;Carga
           el valor mínimo
    * Usa #07FFFh (el mayor posible) para corregir el
           valor mínimo SACB     ;Almacena en ACCB
           SPLK #9,BRCR  ;Rpt cont = 9 para 1O
           valores de datos RPTB endb -1    ;Repite
           bloque desde aqui a endb-1
startb:
       LACC *            ;Carga datos de <(ARO)> en ACC
       CRGT              ;Pone a set carry si ACC > valor mayor previo
* Usa CRLT para encontrar el valor mínimo
           SACL MAXVAL     ;Guarda el nuevo mayor que esta en
           ACC & ACCB XC #1,C ;Guarda dirección si valor actual >
```

```
        mayor previo
        SAR
        ARO,MAXADR
        MAR *+
endb:   RET
* Al final de la rutina, los siguientes registros contienen:
* ACC = 32050
```

* ACCB = 32050
  * (MAXVAL) = 32050
  * (MAXADR) =
    0307h

```
.data                   ;Datos esperados que esten en la RAM de datos
.word 5000                  ;Direccion inicio = 0300h
.word 10000
.word 320
.word 3200
.word -5600
.word -2105
.word 2100
.word 32050
.word 1000
.word -1
.end
```

Ejemplo 8. Utilizando lazos anidados.

```
.mmregs
*.....
*
* Esta rutina realiza la multiplicacion de dos matrices NxN.
* A x B = C donde A,B, y C son de tamaño NxN.
* Condiciones de entrada:
* AR1 -> elemento (0,0) de A (en espacio de programa)
* AR2 -> elemento (0,0) de B (en espacio de datos)
* AR3 -> elemento (0,0) de C (en espacio de datos)
* DP = 0, NDX = 1
* ARP = 2
* Almacenamiento de elementos de matriz en memoria (comienzo por memoria
baja):
* M(0,0),...,M(0,N-1),M(1,0),...,M(N-1,N-1)
*
*.....
      LDP #0
      SPLK #3Eh,PMST
      SPLK #2000h,AR1
      SPLK #0810h,AR2
      SPLK #0820h,AR3
      MAR *,AR2
MTRX_MPY:
      LAR AR0,#(N-1)          ;Establece cuenta de loop
      SPLK #N,INDX            ;Tamaño de fila
      SAR AR2,AR4             ;Guarda direccion de B
* ;For i=0,i<N,++i
LOOP1:  SMMR AR1,BMAR           ;BMAR -> A(i,0)
      SPLK #(N-1),BRCR        ;Establece cuenta de loop2
      SAR AR4,AR5             ;AR5 -> B(0,0)
LOOP2:  RPTB ELOOP2             ;For j=0,j<N,++j
      SAR AR5,AR2             ;AR2 -> B(0,j)
LOOP3:  RPTZ #(N-1)             ;For k=0,k<N,++k
ELOOP3: MADS *0+                ;Acc=A(i,k)xB(k,j)
      APAC                  ;Acumulacion final
      MAR *,AR5               ;ARp = AR5
      MAR *+,AR3              ;AR5 -> B(0,j+1)
ELOOP2: SACL *+,0,AR2           ;Guarda C(i,j)
      MAR *,AR0
      BANZD LOOP1,*-,AR1      ;Cuenta != N
```

```
      ADRKN                ;AR1 -> A(i+1,0)
ELOOP1: MAR  *,AR2         ;ARp = AR2
```

Ejemplo 9. El uso del direccionamiento circular

```
* 'Generador de onda senoidal digital'
.nnnregs
 *.....................----------....... .....---...
 ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,, ,,,,,,,,,,,,
 *
 * Esta rutina ilustra la capacidad de direccionamiento circular de los
 * dispositivos C5x. Un generador de onda senoidal digital se implementa
   como
 * buffer circular #1 con AR7 como su puntero. XSINTBL es la
   localización en
 * la memoria de programa externa donde se almacena esta tabla. Se
   mueve al
 * bloque de memoria de datos interna BI donde se establece como un
   buffer
*circular.

 *-------------------
XSINTBL .set 03000h          ;Dirección de espacio de programa de
   tabla de senos
       .
   tex
   t
   SI
   NT
   BL
   LD
   P
   #O
   LAR AR0,#0300h
                              ;Dir
   eccion del bloque BI MAR
   *,ARO
LACC #XSINTBL                 ;Coloca la direccion de la tabla de senos
   en
* ;la memoria de programa externa
SAMM BMAR                     ;Carga registro fuente
   *
RPT #255                      ;Mueve 256 palabras
BLPD BMAR,*+                  ;Carga tabla de la memoria de programa
   externa
* ;a la memoria de datos inten1a
   SAMM CBSRI                 ;Inicio
   direccion del buffer=300h SAMM AR7
```

```
                                    ;AR7 apunta
    al inicio del buffer ADD #255
    SAMM CBERI                      ;Fin de
    direccion del buffer=3FFh  SPLK
    #OFh,CBCR                       ;Habilita
    CB#I, deshabilita CB#2
    ;puntero para CB#I
    es AR7 NXTSMP
    MAR *,AR7
LACC *+                             ;Toma siguiente muestra de la tabla
    ;AR7 se actualiza para la siguiente muestra valida

DISBLE              APL
    #OFFF7h,CBCR ;Deshabilita
    CB#1
   RET
```

Ejemplo 10. El direccionamiento de módulo 256.

```
START   .set 04000h  ;Inicio direccion
      del buffer LDP #O
  LACL #OFFh
  SAMM DBMR          ;Maximo valor = 255

      MAR *O+       ;Incrementa AR7
      alguna cantidad APL AR7
                    ;Extrae los 8 bits mas
      bajos
  OPL #START,AR7   ;Suma la dirección de inicio
```

Ejemplo 11. Movimiento de bloques de memoria a memoria utilizando RPT con BLDD.

```
*
* Esta rutina usa la instrucción BLDD para mover la memoria de datos externa
* a la memoria de datos interna.
*
MOVEDD:
     LACC #4000h
   SAMM BMAR                ;BMAR -> fuente en
   memoria de datos. LAR AR7,#100h          ;AR7 ->
   destino en memoria de datos MAR *,AR7    ;LARP =
   AR7.
   RPT #1023                ;Mueve 1024 valores a
  los bloques BO y Bl BLDD BMAR, *+
  RET
```

Ejemplo 12. Movimiento de bloques de memoria a memoria utilizando RPT con BLDP.

```
*
* Esta 1utina usa la instrucción BLDP para mover la memoria de datos
  externa
* a la memoria de programa interna. Esta instrucción se podría usar para
  cargar
  * un programa en la memoria de programa de 8K on-chip desde la
    memoria de
  * datos extenm.

MOVEDP:
    LACC #800h
        SAMM BMAR      ;BMAR -> destino en memoria de
        programa ('C50) LAR AR7,#0EOOOh ;AR7 -> fuente
        en la memoria de datos.
        RPT #8191    ;Mueve 8K al espacio de la memoria de
        programa. BLDP *+
        RET
```

Ejemplo 13. Movimiento de bloques de memoria a memoria utilizando RPT con BLPD.

```
*
* Esta última usa la instrucción BLPD para mover la memoria de programa externa
* a la memoria de datos interna. Esta rutina sirve para cargar una tabla de
* coeficientes almacenada en la memoria de programa externa a la memoria de
* datos cuando no hay disponible memoria de datos externa.
*
MOVEPD:
      LAR AR7,#100h ;AR7 -> destino en memoria de datos.
      RPT #127      ;Mueve 128 valores desde la memoria de
      programa externa BLPD #3800h, *+  ;a la memoria de
      datos interna BO.

     RET
```

Ejemplo 14. Movimiento de bloques de memoria a memoria utilizando RPT con TBLR.

```
*
* Esta rutina usa la instrucción TBLR para mover la memoria de programa externa
* a la memoria de datos interna. Esto difiere de la instrucción BLPD en que el
* acumulador contiene la dirección de memoria de programa fuente de la cual
* se transfiere. Esto permite para una calculada, más que predeterminada,
* localización en la memoria de programa a especificar. La rutina de llamada
* debe contener la dirección de memoria de programa fuente en el acumulador.
*
TABLER:
        MAR *,AR3          ;AR3 -> destino en
        memoria de datos. LAR AR3,#300h
        RPT #127           ;Mueve 128 items al bloque de
        memoria de datos Bl TBLR *+
        RET
```

Ejemplo 15. Movimiento de bloques de memoria a memoria utilizando RPT con TBLW.

```
*
        * Esta rutina usa la instrucción TBLW para mover la memoria de
          datos a la
        * memoria de programa. Esto difiere de la instrucción BLDP en
          que el acumulador
        * contiene la dirección de memoria de programa destino a la que
          hay que
* transferir. Esto permite para una calculada, más que predeterminada,
* localización en la memoria de programa a especificar. La rutina de
  llamada
            * debe contenerla dirección de memoria de programa destino
            en el acumulador.

        TABLEW:
   MAR *,AR4                ;ARP = AR4.
      LAR AR4,#380h       ;AR4 -> dirección fuente en memoria
      de datos. RPT #127  ;Mueve 128 items desde la memoria de
      datos a la TBLW *+  ;memoria de programa.
      RET
```

Ejemplo 16. Movimiento de bloques de memoria a memoria utilizando RPT con SMMR.

```
*
* Esta rutina usa la instrucción SMMR para mover datos desde un puerto I/O
* mapeado en memoria a la memoria de datos local. Los 16 puertos I/O están
* mapeados en la página de datos O del mapa de memoria del 'C5x.
"
INPUT:
      LDP #O
      RPT #511 ;Introduce 512 valores del puerto
      51h al comienzo
     SMMR 5lh,800h  ;de tabla en 800h en
      memoria de datos.
      RET
```

Ejemplo 17. Movimiento de bloques de memoria a memoria utilizando RPT con LMMR.

```
    *
* Esta rutina usa la instrucción LMMR para mover datos desde la memoria
    de
* datos local a un puerto I/O mapeado en memoria. Los 16 puertos I/O
    están
* mapeados en la página de datos O del mapa de memoria 'C5x.
    *
OUTPUT:
LDP #O              ;página de datos O
RPT #63             ; Salida de 64 valores desde una
    tabla comenzando LMMR 50h,800h      ;en 800h en
    la memoria de datos al puerto 50h.
RET
```

Ejemplo 18. Cálculo de la raíz cuadrada utilizando la instrucción XC.

```
* Autocorrelacion
* Esta rutina realiza una correlacion de dos vectores y llama a una subrutina
* de raiz cuadrada que determinara la amplitud RMS de la forma de onda.
*
AUTOC
      CALLD SQRT       ;Llama a la subrutina de raiz cuadrada despues de

      SST #0,ST0       ;ejecutar las siguientes dos instrucciones
      SST #1,ST1       ;Obtiene el valor que pasa a la subrutina SQRT
*
* Calculo de la raiz cuadrada
*
* Esta rutina calculala raiz cuadrada de un numero que esta localizado
* en la mitad alta del acumulador. El numero esta en formato Q15.
BRCR     .set 09h        ;DP=0
ST0      .set 60h        ;Bloque de RAM interna B2
ST1      .set 61h
NUMBER   .set 62h
TEMPR    .set 63h
GUESS    .set 64h
         .text
SQRT     MAR *,AR0
         LACC *
         LDP #0
         SETC SXM        ;Pone SXM=1
         SPM 1           ;Pone el modo PM para aritmetica fraccional
         SACL NUMBER     ;Guarda el numero
         LACL #0
         SACB            ;Borra el buffer del acumulador
         SPLK #11,BRCR   ;Inicializa para 12 iteraciones
         SPLK #800h,GUESS
         LACC NUMBER
         SUB  #200h
         BCNDD LOOP,LT   ;Si NUMBER<200h empieza el looping
         SPLK #800h,TEMPR
         LACC #4000h
         SACL GUESS      ;Raiz temporal a 4000h
         SACL TEMPR
         SPLK #14,BRCR   ;e incrementa las iteraciones a 15
LOOP     RPTB ENDLP-1    ;Repite bloque
```

```
      SQRA TEMPR      ;Raiz cuadrada temporal
      LACC NUMBER,16
      SPAC            ;Acc=NUMBER-TEMPR**2
      NOP             ;Ciclo muerto para XC
      XC 2,GT         ;Si NUMBER>TEMPR**2 salta las siguientes 2 instr.
      LACC TEMPR,16
      SACB            ;ROOT <- TEMPR
      LACC GUESS,15
      SACH GUESS      ;GUESS <- GUESS/2
      ADDB
      SACH TEMPR      ;TEMPR <- GUESS+ROOT
ENDLP LACB            ;El Acc alto contiene la raiz cuadrada de NUMBER
      RETD
      LST #1,ST1
      LST #0,ST0      ;Restablece contexto
```

Ejemplo 19. Suma de 64 bits.

```
*
*Dos números de 64-bits se suman produciendo un resultado de 64-bits.
* Los números X (X3, X2, X1 , XO) e Y (Y3, Y2, Y1 , YO) se suman y el
  resultado
* va a W (W3, W2, W1, WO). Si el resultado se requiere en pares de 64-
  bits
* ACC/ACCB, se sustituyen las instrucciones como se indica más abajo.
  *
* X3 X2 X1 XO
  *+Y3 Y2 Y1 YO
  *----------
  * W3 W2 W1 WO -OR- ACC ACCB*
  ADD64      LACC   X1,16      ;ACC = X1 00
             AD     X          ;ACC = XI XO
             DS     0          ;ACC = X1 XO + 00 YO
             ADD    Y1,16      ;ACC = X1 XO + Y1 YO
             SAC    WO         ;Estas 2 instrucciones se sustituyen por
             SAC    W1         ;"SACB" si el resultado se desea en (ACC
             LAC    X3,16      ;ACC = X3 00
             AD     X          ;ACC = X3 X2 + C
             DC     2          ;ACC = X3 X2 + 00 Y2 + C
             ADD    Y3,16      ;ACC = X3 X2 + Y3 Y2 + C
             SAC    W2         ;Estas 2 instrucciones no se requieren si
             SAC    W3         ;el resultado se desea en (ACC ACCB)
             H
             RET
```

Ejemplo 20. Resta de 64 bits.

```
*
* Dos números de 64-bits se restan, produciendo un resultado de 64-
  bits.
* El numero Y (Y3, Y2, Y1, YO) se resta de X (X3, X2, XI, XO) y el
  resultado
* en W (W3, W2, W1, WO). Si el resultado se requiere en pares de 64-
  bits
* ACC/ACCB, se sustituyen las instrucciones como se indica mas
  abajo.
*
* X3 X2 X1 XO
* - Y3 Y2 Y 1 YO
* ----------
W3 W2 W1 WO -OR- ACC ACCB
*
SUB64 LACC   X1 ,16          ; ACC = XI 00
      ADDS   X0              :ACC = X1 X0
      SUBS   Y0              :ACC = X1 XO - 00 Y0
      SUB    Y1,16           ACC = XI XO - YI Y0
      SACL   W0              ; Estas 2 instrucciones se sustituyen
      por
      SACH   W 1             ; "SACB" si el resultado se desea en
      (ACC ACCB)
      LACL   X2              ;ACC = 00 X2
      SUBB   Y2              ; ACC = 00 X2 - 00 Y2 - C
      ADD    X3,16           ; ACC = X3 X2 - 00 Y2 - C
      SUB    Y3,16           ; ACC = X3 X2 - Y3 Y2 - C
      SACL W2                ; Estas 2 instrucciones no se
      requieren si
      SACH W3                ; el resultado se desea en (ACC
      ACCB)
      RET
```

Ejemplo 21. Multiplicación de enteros de 32 bits.

```
        .def MPY32
*..........................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
*
* Esta rutina multiplica dos enteros con signo de 32-bits resultando un producto
* de 64-bits. Los operandos se toman de la memoria de datos y el resultado se
* vuelve a escribir en la memoria de datos.
* Almacenamiento de datos:
* X1,X0 operandos de 32-bits
* Y1,Y0 operandos de 32-bits
* W3,W2,W1,W0 producto de 64-bits
*
* Condiciones de entrada:
* DP = 6, SXM = 1
* OVM = 0
*
*..........................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
X1     .set   300h   ;DP=6
X0     .set   301h   ;DP=6
Y1     .set   302h   ;DP=6
Y0     .set   303h   ;DP=6
W3     .set   304h   ;DP=6
W2     .set   305h   ;DP=6
W1     .set   306h   ;DP=6
W0     .set   307h   ;DP=6
       .text
MPY32:
       BIT    X0,0   ;TC = X0 bit#15
       LT     X0     ;T = X0
       MPYU   Y0     ;P = X0Y0
       SPL    W0     ;Guarda W0
       SPH    W1     ;Guarda W1 parcial
       MPY    Y1     ;P = X0Y1
       LTP    X1     ;ACC = X0Y1, T = X1
       MPY    Y0     ;P = X1Y0
       MPYA   Y1     ;ACC = X0Y1+X1Y0, P=X1Y1
       ADDS   W1     ;ACC = X0Y1+X1Y0+X0Y02^-16
       SACL   W1     ;Guarda W1 final
       BSAR   16     ;Desplaza ACC a la derecha 16
       XC     1,TC   ;Si el MSB ed X0 es 1
       ADD    Y1     ;Suma Y1
       BIT    Y0,0   ;TC = Y0 bit#15
```

```
APAC          ;ACC = X1Y1 + (X0Y1+X1Y0)2^-16
XC     1,TC   ;Si el MSB de Y0 es 1
ADD    X1     ;Suma X1
SACL   W2     ;Guarda W2
SACH   W3     ;Guarda W3
```

Ejemplo 22. Multiplicación fraccional de 32 bits

```
*.....
*
* Esta rutina multiplica dos enteros con signo Q31 resultando un producto Q30.
* Los operandos se toman de la memoria de datos y el resultado se vuelve a
* escribir en la memoria de datos.
* Almacenamiento de datos:
* X1,X0  operandos Q31
* Y1,Y0  operandos Q31
* W1,W0  producto Q30
* Condiciones de entrada:
* DP = 6, SXM = 1
* OVM = 0
*
*.....
X1      .set   300h    ;DP=6
X0      .set   301h    ;DP=6
Y1      .set   302h    ;DP=6
Y0      .set   303h    ;DP=6
W1      .set   304h    ;DP=6
W0      .set   305h    ;DP=6
        .text
        BIT    X0,0    ;TC = X0 bit#15
        LT     X0      ;TREG0 = X0
        MPY    Y1      ;P = X0*Y0
        LTP    X1      ;ACC = X0*Y0
        MPY    Y0      ;P = X1*Y0
        MPYA   Y1      ;ACC = X0*Y0 + X1*Y0
        BSAR   16      ;16 bits bajos
        XC     1,TC    ;Si el MSB de X0 es 1
        ADD    Y1      ;entonces suma Y1
        BIT    Y0,0    ;TC = Y0 bit#15
        APAC           ;ACC = ACC + X1*Y1
        XC     1,TC    ;Si el MSB de Y0 es 1
        ADD    X1      ;entonces suma X1
        SACL   W0      ;Guarda el producto mas bajo
        SACH   W1      ;Guarda el producto mas alto
```

Ejemplo 23. División de enteros de 16 bits usando la instrucción SUBC

```
*
* Esta rutina implementa la división de enteros con la instrucción SUBC.
  Para
* esta rutina de división de enteros, el valor absoluto del numerador debe
  ser
* mayor que el valor absoluto del denominador. Además, la rutina de
  llamada
* debe comprobar que el divisor no sea O.
'"
* El dividendo de 16 bits se coloca en el acumulador bajo, y el
  acumulador
*  alto es puesto a cero. El divisor está en la memoria de datos. En la
  última
*  SUBC, el cociente de la división está en los 16 bits de orden m s bajo
  del
* acumulador.
* El resto está  en los 16 bits de orden más alto.
*
* RETCD devuelve si las condiciones son ciertas - después de ejecutar la
*siguiente instrucción de 2-palabras o dos instrucciones de palabras
simples
*
DENOM   .set   60h
NUMERA  .set 6Ih QUOT   .set          62h REM    .set    63h TEMSGN
.set
          64h
*
INTDIV LDP     #0
       LT      NUMERA ;Determina signo del
       cociente.
       MPY     DENOM
*
       SPH    TEMSGN ;Guarda
       el signo LACL
                DENOM
       ABS           ; Hace al denominador y numerador
       positivos.
       SACL   DENOM   ;Guarda el valor absoluto del
       denominador LACL       NUMERA
   ABS
```

```
*
* Si el divisor y el dividendo están alineados, la división puede empezar
aqui.
*
      RPT    #15    ;División de 16 ciclos. El acumulador bajo contiene
      SUBC   DENOM   ;el cociente y el acumulador alto contiene
* ;el resto al final del lazo.
      BIT    TEMSGN,0 ;Testea el signo del
      cociente.
      RETCD  NTC    ;Return si signo positivo, sino
      continua.
      SACL QUOT      ;Almacena cociente y recuerda durante el retraso
      SACH  REM      ;return.
   LACL  #0        ;Si signo negativo, niega cociente y return
   RETD
    SUB  QUOT
   SACL  QUOT
```

Ejemplo 24. División fraccionada de 16 bits usando la instrucción SUBC

```
*
* Esta rutina implementa la división fraccionada con la instrucción SUBC.
* Para esta rutina de división, el valor absoluto del denominador debe ser
* mayor que el valor absoluto del numerador. Adem s, la rutina de llamada debe
* comprobar que el divisor no sea 0.
*
* El dividendo de 16 bits se coloca en el acumulador alto, y el acumulador bajo
* es puesto a cero. El divisor est  en la memoria de datos.
*
DENOM  .set  60h
NUMERA .set  61h
QUOT   .set  62h
REM    .set  63h
TEMSGN .set  64h
*
FRACDIV LDP   #0
     LT    NUMERA        ;Determina el signo del cociente.
*
     MPY   DENOM
     SPH   TEMSGN
     LACL  DENOM
     ABS                 ;Hace al denominador y al numerador positivos.
     SACL  DENOM
     LACC  NUMERA,16     ;Carga el acumulador alto, hace cero el
                         ;acumulador bajo.
     ABS
*
* Si el divisor y el dividendo estan alineados, la division puede empezar aqui.
*
     RPT   #14           ;Division de 15 ciclos. El acumulador bajo
     SUBC  DENOM         ;contiene el cociente y el acumulador alto
                         ;contiene el resto al final del lazo.
*
     BIT   TEMSGN,0      ;Testea el signo del cociente.
     RETCD NTC           ;Retorna si signo es positivo, si no continua.
     SACL  QUOT          ;Almacena el cociente y el resto durante el
     SACH  REM           ;retorno con retardo.*
     LACL  #0            ;Si el signo es negativo, niega el cociente y
                         ;retorna
     RETD
     SUB   QUOT

     SACL QUOT
```

Ejemplo 25. Suma de coma flotante usando las instrucciones SATL y SATH

```
*.....
*
* ESTA SUBRUTINA SUMA DOS NUMEROS COMA FLOTANTE
PRODUCIENDO UN PRODUCTO
* NORMALIZADO COMA FLOTANTE. EL FORMATO DE LOS NUMEROS
COMA FLOTANTE ESTA
* ESPECIFICADO DEBAJO.

*
* FORMATO ENTRADA / SALIDA
* ========================
* ----------------
* | TODO 0 O 1 | PALABRA CON SIGNO
* ----------------
*
* ----------------
* | 16 BITS | EXPONENTE
* ----------------
*
* ----------------
* |0| 15 BITS | PARTE ALTA DE LA MANTISA
* ----------------
*
* ----------------
* | 16 BITS | PARTE BAJA DE LA MANTISA
* ----------------
*
* Instrucciones C5x:
*
* SAMM guarda el contenido del acumulador en un registro mapeado en memoria
* LACB el acumulador se carga con el contenido del buffer del acumulador
* SACB contenido del acumulador se copia en el buffer del acumulador
* SATL el acumulador se desplaza a la derecha por el valor especificado
* en los 4 LSBs del TREG1
* SATH el acumulador se desplaza a la derecha 16 bits si el bit 4 del TREG1
* es un uno.
* SPLK almacena la constante inmediata larga en la memoria de datos
* CPL compara el valor del inmediato largo (o DBMR) con la memoria de datos
* TC=1 si los dos valores son identicos
* TC=0 en otro caso
*
```

```
*.......................................................................
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
TREG1  .set  0dh
ASIGN  .set  60h   ;Signo, exponente, parte alta y baja de la mantisa
AEXP   .set  61h   ;del numero de entrada A
AHI    .set  62h
ALO    .set  63h
BSIGN  .set  64h   ;Signo, exponente, parte alta y baja de la mantisa
BEXP   .set  65h   ;del numero de entrada B
BHI    .set  66h
BLO    .set  67h
CSIGN  .set  68h   ;Signo, exponente, parte alta y baja de la mantisa
CEXP   .set  69h   ;del numero resultante coma flotante C
CHI    .set  6Ah
CLO    .set  6Bh
DIFFEXP .set  6Ch
       .text
FL_ADD LDP    #0     ;Inicializacion
       SETC   SXM    ;Set modo de extension de signo
       MAR    *,AR0  ;ARP <- AR0
       LAR    AR0,#0 ;AR0 se usa por la instruccion NORM
CMPEXP LACL   BLO    ;Carga el Acc bajo con BLO
       ADD    BHI,16 ;Suma BHI al Acc alto
       SACB          ;AccB = BHIBLO
       LACC   AEXP
       SUB    BEXP   ;Acc = AEXP=BEXP
       SACL   DIFFEXP ;Guarda las diferencias
       BCND   AEQB,EQ ;Si |A| == |B|
       BCND   ALTB,LT ;Si |A| < |B|
AGTB   LACC   DIFFEXP ;Si |A| > |B|
       SAMM   TREG1  ;Carga TREG1 con # de desplazamientos requeridos
       SUB #32
       BCND   AGRT32,GEQ ;Si diferencia > 32
       LACB          ;Acc = BHIBLO
       SATL
       SATH
       SACB          ;Almacena el resultado de vuelta en AccB
AEQB   LACC   ASIGN  ;Copia los valores del signo y exponente de
       SACL   CSIGN  ;A en C (el resultado)
       LACC   AEXP
       SACL   CEXP
CHKSGN LACC   ASIGN  ;Acc=ASIGN-BSIGN
       SUB    BSIGN
       CLRC   TC     ;Borra señal TC
```

```
    XC    1,LT   ;Si A<0 y B>0
    SETC  TC     ;Set señal TC
    BCNDD ADNOW,EQ ;Si A y B tienen mismo signo
    LACL  ALO
    ADD   AHI,16 ;Acc = AHIALO
    SBB          ;Acc=A-B
    XC    1,TC   ;Si A<0 y B>0
    NEG          ;entonces Acc=B-A
    BCND  CZERO,EQ ;Si A-B == 0
    XC    2,LT   ;Si A-B < 0
    SPLK  #0FFFFh,CSIGN ;entonces CSIGN=-1
    XC    2,GT   ;Si A-B > 0
    SPLK  #0,CSIGN ;entonces CSIGN=0
    XC    1,LT   ;Si A-B<0
    ABS          ;entonces Acc=|A-B|
    BD    NORMAL ;rama retardo
    SACH  CHI    ;Guarda el resultado
    SACL  CLO
CZERO LACL  #0     ;Si A-B == 0
    SACL  CEXP   ;el resultado es cero
    SACL  CSIGN  ;Hace el signo positivo
    RETD         ;Retorno con retardo
    SACL  CHI
    SACL  CLO    ;Borra CHICLO
ADNOW ADDB           ;Si signos son iguales
    BCNDD OVFLOW,OV ;suma los dos numeros
    SACH  CHI
    SACL  CLO    ;Lo guarda en CHICLO
    BCND  CZERO,EQ ;Si CHICLO es cero, va a CZERO
NORMAL CPL   #0,CHI ;Compara CHI con 0
    NOP          ;Ciclo muerto para XC
    XC    2,TC   ;Si CHI es 0
    LACC  CLO,16 ;normaliza solo la parte CLO
    LAR   AR0,#16 ;AR0 tiene valor exponente
    XC    2,NTC  ;Si CHI != 0
    LACC  CHI,16 ;Acc=CHICLO
    ADDS  CLO
    CLRC  SXM    ;Deshabilita modo extension signo
    XC    2,LT   ;Si el MSB del CLO es 1
    SBRK  1      ;desplaza a la derecha una vez
    SFR          ;y decrementa el exponente.
    SETC  SXM    ;Habilita modo extension signo
    RPT   #13    ;Repite 14 veces
```

```
     NORM   *+     ;Normaliza
OUTPUT SACH   CHI    ;Almacena parte alta
     SACL   CLO    ;Almacena parte baja del resultado
     LACC   CEXP
     SAR    AR0,CEXP ;Guarda el exponente
     RETD          ;Retorno con retardo
     SUB    CEXP
     SACL   CEXP   ;CEXP=CEXP-AR0
OVFLOW CLRC   SXM    ;Deshabilita modo extension signo
     SFR           ;Desplaza Acc a la derecha
     SACH   CHI
     SACL   CLO    ;Guarda el resultado
     LACC   CEXP
     ADD    #1     ;Incrementa el exponente en uno
     SACL   CEXP   ;Lo guarda
ALTB   LACC   BSIGN  ;Copia signo de B en C
     SACL   CSIGN
     LACC   BEXP   ;Copia exponente de B en C
     SACL   CEXP
     LACC   DIFFEXP
     NEG           ;mientras A-B < 0
     SAMM   TREG1  ;Numero de desplz. requeridos para justificacion
     SUB    #32
     BCND   BGRT32,GEQ ;diferencia en exponente >= 32
     LACL   ALO
     ADD    AHI,16 ;Acc=AHIALO
     SATL
     SATH
     BD     CHKSGN ;Salta despues de las dos instrucciones siguientes
     SACL   ALO    ;Guarda valor normalizado
     SACH   AHI    ;en ALO y AHI
BGRT32 LACC   BHI    ;Si exponente de B > 32
     SACL   CHI    ;entonces C <- B.
     RETD          ;Retorna despues
     LACC   BLO    ;guardando CHI y CLO
     SACL   CLO
AGRT32 LACC   AHI    ;Si exponente de A > 32
     SACL   CHI    ;entonces C <- A.
     LACC   ALO
     SACL   CLO    ;Copia ALO a CLO
     LACC   ASIGN
     SACL   CSIGN  ;Copia ASIGN a CSIGN
     RETD          ;Retorna despues
```

```
LACC   AEXP   ;copiando AEXP a
SACL   CEXP
```

Ejemplo 26. Multiplicación en coma flotante usando la instrucción BSAR

```
*.....
*
* ESTA SUBRUTINA MULTIPLICA DOS NUMEROS COMA FLOTANTE
PRODUCIENDO
* UN PRODUCTO NORMALIZADO EN COMA FLOTANTE. EL FORMATO DE
LOS NUMEROS EN
* COMA FLOTANTE ESTA ESPECIFICADO ABAJO.
*
* FORMATO ENTRADA / SALIDA
* ========================
* ---------------
* | TODO 0 O 1 | PALABRA CON SIGNO
* ---------------
*
* ---------------
* | 16 BITS | EXPONENTE
* ---------------
*
* ---------------
* |0| 15 BITS | PARTE ALTA DE LA MANTISA
* ---------------
*
* ---------------
* | 16 BITS | PARTE BAJA DE LA MANTISA
* ---------------
*
* NOTAR QUE SI EL PRODUCTO ES CERO, EL SIGNO DEL PRODUCTO
PUEDE SER
* POSITIVO O NEGATIVO DEPENDIENDO DE LAS ENTRADAS.
*
* Instrucciones C5x:
* BSAR desplazamiento aritmetico a la derecha de 1-16 bits en un ciclo
* CLRC resetea el bit de control
* SETC pone a set el bit de control
* BD rama despues de ejecutar las dos siguientes instrucciones de una palabra
* o una instruccion de dos palabras
*
*.....
ASIGN   .set   60h    ;Signo, exponente, partes alta y baja de la mantisa
AEXP    .set   61h    ;del numero de entrada A
AHI     .set   62h
```

```
ALO    .set   63h
BSIGN  .set   64h    ;Signo, exponente, partes alta y baja de la mantisa
BEXP   .set   65h    ;del numero de entrada B
BHI    .set   66h
BLO    .set   67h
CSIGN  .set   68h    ;Signo, exponente, partes alta y baja de la mantisa
CEXP   .set   69h    ;del numero resultante en coma flotante C
CHI    .set   6ah
CLO    .set   6bh
       .text
MULT   LDP    #0
       MAR    *,AR0  ;ARP <- AR0
       LAR    AR0,#0 ;Resetea el contador de exponente
       SPM    0      ;No desplazamiento a la izquierda del registro P
       LACC   AEXP
       ADD    BEXP
       SACL   CEXP   ;CEXP = AEXP + BEXP
       CLRC   SXM    ;para desplazamiento, deshabilita extension de signo
       LT     ALO    ;T = ALO
       MPYU   BHI    ;P = ALO*BHI
       LTP    AHI    ;Acc=ALO*BHI, T=AHI
       MPYU   BLO    ;P=AHI*BLO
       MPYA   BHI    ;Acc=ALO*BHI + AHI*BLO, P=AHI*BHI
       BSAR   16     ;Retiene 16 bits de mas arriba mas 1 bit adicional
       APAC          ;debido a los MSBs cero de BLO & ALO
       BCND   NZERO,NEQ ;Si el producto no es cero
       SACH   CHI    ;Si el producto es cero
       BD     SIGN   ;entonces borra CHI,CLO y CEXP
       SACL   CLO    ;y salta a SIGN
       SACL   CEXP
NZERO  SFL           ;Descarta el bit de signo adicional (Q63)
       NORM   *+
       SACH   CHI    ;Guarda el producto
       SACL   CLO
       SETC   SXM    ;Habilita modo extension de signo
       LACC   CEXP
       SAR    AR0,CEXP ;CEXP<-AR0
       SUB    CEXP
       SACL   CEXP   ;CEXP=CEXP-AR0
SIGN   LACL   ASIGN  ;Si los signos son iguales el producto es +ve
       RETD          ;Retorna despues de las dos siguientes instrucciones
       XOR    BSIGN  ;en otro caso es -ve.
       SACL   CSIGN
```

Ejemplo 27. Codificador V.32 usando el buffer del acumulador

```
      .mmregs
STATMEM .set   60h          ;(60h - 62h) Estados de retardo S1,S2,S3
INPUT   .set   64h          ;(64h - 67h) Cuatro bits de entrada
YPAST   .set   68h          ;(68h - 69h) Valores pasados de Y1 y Y2
OUTPUT  .set   63h          ;Y0, el bit redundante
LOCATE  .set   6ah          ;Almacenamiento temporal para palabra entrada
PCKD_IP .set   1000h        ;Buffer entrada (paquetes 4 bits por palabra)
PCKD_OP .set   2000h        ;Buffer salida (paquetes 5 bits por palabra)
COUNT   .set   50           ;# de palabras de datos de entrada
      .text
INIT   LAR    AR1,#PCKD_IP
       LAR    AR2,#PCKD_OP
       LAR    AR3,#COUNT-1   ;COUNT contiene # de palabras de entrada
       LDP    #0
START  MAR    *,AR1
       LACC   *+,0,AR0
       SACL   LOCATE         ;Almacenamiento temporal para palabra entrada
       LAR    AR0,#INPUT+3
       LACL   #3             ;Lazo 4 veces
       SAMM   BRCR
       LACL   #1
       SAMM   DBMR           ;Carga DBMR con mascara para el LSB
UNPACK LACC   LOCATE         ;Acc = bits entrada empaquetados
       RPTB   LOOP1-1        ;for I=0,I<=3,I++
       SACL   *              ;Lo guarda
       APL    *-             ;Desenmascara todos los bits excepto LSB
       SFR                   ;Desplaza derecha para obtener siguiente bit
LOOP1
       CALL   DIFF           ;Llama al codificador diferencial
       CALL   ENCODE         ;Llama al codificador convolucional
PACK   LAR    AR0,#INPUT
       LACL   #3             ;Lazo 4 veces
       SAMM   BRCR
       LACC   *+             ;Obtiene el primer bit (MSB)
       RPTB   LOOP2-1        ;for I=0,I<=2,I++
       SFL                   ;hace espacio por desplaz izquierda una vez
       ADD    *+             ;Empaqueta siguiente bit desplazando izq otro
       NOP
LOOP2
       MAR    *,AR2          ;ARP <- AR2
```

```
      SACL   *+,0,AR3       ;Lo guarda en forma de paquete
      BANZ   START          ;Lazo si COUNT no es cero
      RET                   ;Retorna

; Esta subrutina codifica diferencialmente Q1n y Q2n (buffer INPUT)
; de acuerdo con los valores de salida anteriores Y1n-1 y Y2n-1 (buffer YPAST).
; Los valores resultantes Y1n y Y2n sobreescriben a los anteriores Q1n y Q2n.
DIFF   LACC   YPAST           ;Acc=Y1n-1
       AND    INPUT          ;Q1n & Y1n-1
       XOR    INPUT+1        ;(Q1n & Y1n-1) xor Q2n
       XOR    YPAST+1        ;(Q1n & Y1n-1) xor Q2n xor Y2n-1
       SACL   INPUT+1
       SACL   YPAST+1        ;Guarda Y2n
       LACC   YPAST
       XOR    INPUT          ;Q1n xor Y1n-1
       RETD                  ;Retorno con retardo
       SACL   INPUT          ;Guarda Y1n
       SACL   YPAST          ;guarda Y1n-1
; Esta subrutina genera un bit redundante Y0n por codificacion convolucional,
; tomando Y1n y Y2n como entrada. Tres estados de retardo S1, S2 y S3 se
; localizan en el buffer STATMEM.
ENCODE LACC   STATMEM
       SACL   OUTPUT          ;Y0 <- S1
       LACC   INPUT+1
       XOR    STATMEM+1       ;Y2 xor S2
       SACB                   ;Guarda en AccB
       LACC   OUTPUT
       AND    INPUT           ;Y0 & Y1
       XORB                   ;(Y0 & Y1) xor (Y2 xor S2)
       SACL   STATMEM         ;Lo guarda en S1
       LACC   OUTPUT
       ANDB                   ;Y0 & (Y2 xor S2)
       SACB
       LACC   INPUT
       XOR    INPUT+1         ;Y1 xor Y2
       XOR    STATMEM+2       ;(Y1 xor Y2) xor S3
       XORB                   ;((Y1 xor Y2) xor S3) xor (Y0 & (Y2 xor S2))
       SACL   STATMEM+1       ;Actualiza S2
       RETD                   ;Retorno con retardo
       LACC   OUTPUT
       SACL   STATMEM+2       ;Actualiza S3
```

Ejemplo 28. Filtro FIR adaptable usando instrucciones RPT y RPTB

```
.def ADPFIR
.def X,Y
.mmregs
*
* Este filtro FIR adaptable de 128-tap usa el bloque de memoria on-chip B0
* para coeficientes y el bloque B1 para muestras de datoss. La entrada mas
* nueva deberia estar en el lugar de memoria X cuando sea llamada. La salida
* estara en el lugar de memoria Y cuando retorne.
*
* OVLY =1 , RAM =1 cuando esta rutina se llama.
*
COEFFP  .set   02000h  ;Direccion de memoria de programa de coef. en S/A RAM
COEFFD  .set   02000h  ;Direccion de memoria de datos de coef. en S/A RAM
      * Para 'C51,'C53,'C56,'C57, COEFFD es 0800h en vez de 02000h
ONE     .set   7Ah     ;Constante uno. (DP=0).
BETA    .set   7Bh     ;Constante de adaptacion. (DP=0).
ERR     .set   7Ch     ;Señal de error. (DP=0).
ERRF    .set   7Dh     ;Funcion de error. (DP=0).
Y       .set   7Eh     ;Filtro salida. (DP=0).
X       .set   037Fh   ;Muestra de datos mas nueva.
FRSTAP  .set   0380h   ;Siguiente muestra de datos mas nueva.
LASTAP  .set   03FFh   ;Muestra de datos mas vieja.
*
* Filtro de respuesta de impulso finita (FIR).
*
ADPFIR  ZPR                ;Borra registro P.
        LACC   #1,14       ;Carga bit de redondeo de salida.
        MAR    *,AR3
        LAR    AR3,#LASTAP     ;Apunta a muestra mas vieja.
FIR     RPT    #127
        MACD   COEFFP,*-       ;Filtro FIR de 128-tap .
        APAC
        SACH   Y,1             ;Alamacena la salida filtro.
        NEG                ;Acc = -y(n)
        LAR    AR3,#X
        ADD    *,15            ;Suma la muestra de entrada mas nueva.
        SACH   ERR,1           ;err(n) = x(n) - y(n)
        DMOV   *               ;Incluye la muestra mas nueva
*
* Adaptacion LMS de los Coeficientes de Filtro.
```

```
*
      LT    ERR          ;T = err
      MPY   BETA         ;P = beta*err(i)
      PAC                ;errf(i) = beta * err(i)
      ADD   ONE,14       ;Redondea los resultados.
      SACH  ERRF,1       ;Guarda errf(i)
      LACC  #126
      SAMM  BRCR         ;127 coeficientes para actualizar lazo.
      LAR   AR2,#COEFFD  ;Apunta a los coeficientes.
      LAR   AR3,#LASTAP+1 ;Apunta a las muestras de datos.
      LT    ERRF
      MPY   *-,AR2       ;P = 2*beta*err(i)*x(i-255)
*
      RPTB  LOOP-1       ;For I=0,I<=126,I++
ADAPT ZALR  *,AR3        ;Carga ACCH con ak(i).
      MPYA  *-,AR2       ;P = 2*beta*err(i)*x(i-k-1)
* Acc = ak(i) + 2*beta*err(i)*x(i-k)
      SACH  *+           ;Almacena ak(i+1)
*
LOOP  ZALR  *,AR3        ;Finalmente actualiza ultimos coef. a0(i)
      RETD               ;Retorno con retardo
      APAC               ;Acc = a0(i) + 2*beta*err(i)*x(i)
      SACH  *+           ;Guarda a0(i+1)
```

Ejemplo 29. Filtro IIR de orden N usando instrucciones RPT y MACD.

```
      .mmregs
*.....................................................................
*
* Esta rutina implementa un filtro IIR de orden N tipo II.
* d(n) = x(n) - d(n-1)a1 - d(n-2)a2 -...- d(n-N+1)aN-1
* y(n) = d(n)b0 + (dn-1)b1 +...+ d(n-N+1)bN-1
*
* Requerimientos de Memoria:
* Variables de estado (memoria de datos baja a alta):
* d(n) d(n-1) ... d(n-N+1)
*
* Coeficiente (memoria de programa baja a alta):
* -a(N-1) -a(N-2) ... -a(1) b(N-1) b(N-2) ... b(1) b(0)
*
* Condiciones de Entrada:
* AR0 -> Entrada
* AR1 -> d(n-N+1)
* AR2 -> Salida
* COEFFA -> -a(N-1)
* COEFFB -> b(N-1)
* ARP = AR0
*
*.....................................................................
IIR_N:  ZPR             ;Borra registro P
        LACC    *,15,AR1        ;Obtiene entrada Q15
        RPT     #(N-2)          ;For i=1,i<=N-1,++i
        MAC     COEFFA,*-       ;Acc+=-a(N-i))*d(n-N+i)
        APAC                    ;Acumulacion final
        SACH    *,1             ;Guarda d(n)
        ADRK    N-1             ;AR1 -> d(n-N+1)
        LAMM    BMAR            ;Acc -> a(N-1)
        ADD     #N-1            ;Acc -> b(N-1)
        SAMM    BMAR            ;BMAR -> b(N-1)
        RPTZ    #(N-1)          ;For i=1,i<=N,++i
        MACD    COEFFB,*-       ;Acc+=b(N-i)*d(n-N+i)
        LTA     *,AR2           ;Acumulacion final
        SACH    *,1             ;Guarda Yn
```

Ejemplo 30. Filtro IIR de N cascadas bicuadrado usando las instrucciones LTD y MPYA

```
      .mmregs
*.....
*
* Esta rutina implementa N bloques en cascada de filtros bicuadrados IIR
* canonicos de tipo II. Cada bicuadrados requiere 3 lugares de memoria de datos

* d(n),d(n-1),d(n-2), y 5 coeficientes -a1,-a2,b0,b1,b2.
* Para cada bloque: d(n) = x(n)-d(n-1)a1-d(n-2)a2
* y(n) = d(n)b0+d(n-1)b1+d(n-2)b2
*
* Almacenamiento de coeficientes (memoria de datos baja a alta):
* -a2,-a1,b2,b1,b0, ... ,-a2,-a1,b2,b1,b0
* Primer bicuadrado N-simo bicuadrado
*
* Variables de estado (memoria de datos baja a alta):
* d(n),d(n-1),d(n-2), ... ,d(n),d(n-1),d(n-2)
* N-simo bicuadrado primer bicuadrado
*
* Condiciones de entrada:
* AR1 -> d(n-2) del primer bicuadrado
* AR2 -> -a2 del primer bicuadrado
* AR3 -> muestra de entrada (numero Q15)
* AR4 -> muestra de salida (numero Q15)
* DP = 0, PM = 0, ARP = 3
*
*.....
BIQUAD:                 ;Establece variables
      ZPR               ;Borra registro P
      LACC   *,15,AR1   ;Obtiene entrada Q15
      SPLK   #2,INDX    ;Establece registro indice
      SPLK   #N-1,BRCR  ;Establece cuenta
                        ;Empieza calculo;
      RPTB   ELOOP-1    ;Repeticion para N bicuadrados
LOOP:
      LT     *-,AR2     ;T = d(n-2)
      MPYA   *+,AR1     ;Acc = x(n), P = -d(n-2)a2
      LTA    *-,AR2     ;Acc += -d(n-2)a2, T = d(n-1)
      MPY    *+         ;P = -d(n-1)a1
      LTA    *+,AR1     ;Acc += -d(n-1)a1, T = b2
```

```
        SACH    *0+,1           ;Guarda d(n)
        MPY     *-              ;P = d(n-2)b2
        LACL    #0              ;Acc = 0
        LTD     *-,AR2          ;T = d(n-1), d(n-2) = d(n-1)
        MPY     *+,AR1          ;Acc += d(n-2)b2, P = d(n-1)b1
        LTD     *-,AR2          ;T = d(n), d(n-1) = d(n)
        MPY     *+,AR1          ;Acc += d(n-1)b1, P = d(n)b0
ELOOP:
        LTA     *,AR4           ;Acumulacion final
        SACH    *,1             ;Guarda la salida en formato Q15
```

Ejemplo 31. Algoritmos backtracking usando direccionamiento circular

```
*
* Ejemplo de Backtracking
* Este programa traza el camino optimo expandido por un algoritmo de
* programacion dinamico. La historia del camino consta de cuatro caminos
* expandidos N veces.
* Esta establecido como un buffer circular de longitud N*4. Se usa buffer
* circular de tipo decremental. La direccion de inicio y fin del buffer
* circular se inicializan de esta manera por dos razones:
* 1- para evitar skip la direccion fin del buffer circular
* 2- para asegurar que el wrap-around se completa antes de la proxima iteracion
*
      LAR    AR0,#BUFFER    ;Toma la direccion del buffer
      LMMR   INDX,PATH      ;Toma el camino seleccionado [0..3]
      SPLK   #N-1,BRCR
* init. AR0 como puntero a buffer circular#1; longitud=N*4 palabras
      SPLK   #BUFFER+(N-1)*4,CBSR1
      SPLK   #BUFFER-3,CBER1
      SPLK   #08h,CBCR
*
      RPTB   TLOOP-1        ;For i=0,i<N,i++
      MAR    *0+            ;Offset por estado#
      LACC   *0-            ;Toma siguiente puntero & resetea el estado#0
      SAMM   INDX           ;Guarda siguiente estado#
      SBRK   3              ;Decrementa AR0 para evitar skip CBER1
      SBRK   1              ;Ahora AR0 se posiciona correctamente 1 vez
TLOOP:                      ;periodo de vuelta (direccionamiento circular)
```

Ejemplo 32. FFT compleja 16 puntos y radio 2.

```
      .file  "c5cx0016.asm"
      .width  120
N     .set   16          ;NUMERO DE PUNTOS PARA FFT
      .mmregs
pmstmask .set  0110b       ;ndx=trm=1
*
*************************************************************************
**********

* *
* 16 - PUNTOS, RADIO-2 DIF FFT COMPLEJA CON EL TMS320C5x / CODIGO
LAZO *
* ------------------------------------------------------------- *
*************************************************************************
**********
* EL PROGRAMA ESTA BASADO EN EL LIBRO 'DIGITAL SIGNAL
PROCESSING APPLICATIONS' *
* DE TEXAS INSTRUMENTS P. 69.
* *
*************************************************************************
**********
* *
* REGISTROS USADOS: INDX,AR1,AR2,AR3,AR4,AR5,ACCU,PREG,TREG0,
PMST, BRCR *
* *
* MEMORIA DE PROGRAMA: 164 PALABRAS ('END' - 'FFT') SIN
INICIALIZACION *
* *
* COEFICIENTES : 16 BITS (Formato Q15) ESCALA: 1/2^4 *
* *
* SECUENCIA DE PROGRAMA:0. INICIALIZACION PARA FFT/COEFF ADD:
240H - 20BH *
* 1. ENTRADA NUEVOS DATOS EN 'INPUT': 220H - 23FH *
* 2. LLAMADA SUBRUTINA SUMA FFT: 600H - 6A3H *
* 2.1.INVERSION BIT DE ENTRADA A SUMA DATOS: 200H - 21FH *
* 2.2. FFT CON SUMA DATOS DE ESPACIO DE TRABAJO: 200H - 21FH *
* 3. SALEN LOS RESULTADOS DE LA SUMA DE DATOS: 200H - 21FH *
* *
* DATOS DE ENTRADA EN LA DIRECCION 0220h-023fh: *
* ------------------------------- *
```

```
* LOS DATOS SE ALMACENAN EN 'INPUT' COMO LA SECUENCIA:
X(0),X(1),...,X(15) *
* Y(0),Y(1),...,Y(15) *
* *
* DATOS DE SALIDA EN LA DIRECCION 0200h-021fh: *
* ---------------------------------- *
* LOS DATOS SE ALMACENAN EN 'DATA' COMO LA SECUENCIA: *
* X(0),Y(0),X(1),Y(1),... ... ,X(15),Y(15) *
*****************************************************************
**********
* *
* ESTE PROGRAMA INCLUYE EL SIGUIENTE ARCHIVO: *
* ---------------------------------- *
* EL ARCHIVO 'TWIDDLES.Q15' CONSISTE DE FACTORES TWIDDLE EN
FORMATO Q15 *
* EL ARCHIVO 'C5CXRAD2.MAC' archivos macro *
* EL ARCHIVO 'INIT-FFT.ASM' para inicializacion *
*****************************************************************
**********
*
      .include      C5CXRAD2.MAC
      .def          TWIDLEN,FFTLEN,TEMP,WAIT,cos45
      .def          INIT,FFT,TWIDSTRT,TWIDEND
      .def          STAGE1,STAGE3,STAGE4,INPUT,DATA,TWID
;
      .sect   "twiddles"
                  ; tabla de factores twiddle para la FFT
TWIDSTRT .set   $
      .include twiddles.q15
TWIDEND .set    $
TWIDLEN .set    TWIDEND-TWIDSTRT
*
INPUT   .usect  "input",N*2 ;array de datos de entrada
DATA    .usect  "data",N*2 ;trabajando array de datos
TWID    .usect  "twid",N*2 ;reserva espacio para twiddles
*
*       .include init-fft.asm
*
      .sect   "fftprogram"
*
* FFT CODIGO CON MUESTRAS DE ENTRADA DE BIT INVERTIDO /
ARP=AR3
*
```

```
FFT:   LAR    AR3,DATAADD    ;TRANSFIERE 32 PALABRAS DE 'input' a
'data'
       LACC   NN
       SAMM   INDX          ;registro indice=7
       RPT    NN2           ;N VECES
       BLDD   #INPUT,*BR0+
*
* FFT CODIGO para STAGES 1 y 2
*
STAGE1: SPLK   #7,INDX       ;registro indice = 7
       LAR    AR1,DATAADD    ;puntero a DATA r1,i1
       LAR    AR2,#DATA+2    ;puntero a DATA + 2 r2,i2
       LAR    AR3,#DATA+4    ;puntero a DATA + 4 r3,i3
       LAR    AR4,#DATA+6    ;puntero a DATA + 6 r4,i4
       COMBO5X 4             ;repite 4 veces
*
* FFT CODIGO PARA STAGE 3 / ARP=AR2
*
STAGE3: SPLK   #9,INDX       ;registro indice = 9
       LAR    AR1,DATAADD    ;ar1 -> DATA
       LAR    AR2,#DATA+8    ;ar2 -> DATA+8
       stage3  2             ;repite 2 veces
*
* FFT CODIGO PARA STAGE 4 / ARP=ARP
*
STAGE4: SPLK   #1,INDX       ;registro indice = 1
       LAR    AR1,DATAADD
       LAR    AR2,#DATA+16
       LAR    AR3,cos4       ;inicio de coseno en stage 4
       LAR    AR4,sin4       ;inicio de seno en stage 4
       SPLK   #6,BRCR
       ZEROI                 ;ejecuta ZEROI
       BUTTFLYI              ;ejecuta 7 veces BUTTFLYI
       RET
END:   .set   $
FFTLEN .set   END-FFT+1
       .end
```

Ejemplo 34. Macros para una FFT DIT de 16 puntos

```
*****************************************************************
***********
* FILE: c5cxrad2.mac --> archivo macro para fft's radio 2 basadas en 320c5x *
* *
* COPYRIGHT TEXAS INSTRUMENTS INC. 1990 *
*****************************************************************
***********
* *
* MACRO 'COMBO2X' PARA COMPLEJAS, RADIO-2 DIT FFT *
* *
* ORGANIZACION DE LA MEMORIA DE DATOS DE ENTRADA:
R1,I1,R2,I2,R3,I3,R4,I4 *
* *
*****************************************************************
***********
* *
* LA MACRO 'COMBO2x' REALIZA LOS SIGUIENTES CALCULOS: *
* *
* R1 := [(R1+R2)+(R3+R4)]/4 ENTRADA SALIDA *
* R2 := [(R1-R2)+(I3-I4)]/4 ---------- ---------- *
* R3 := [(R1+R2)-(R3+R4)]/4 AR0 = 7 *
* R4 := [(R1-R2)-(I3-I4)]/4 AR1 -> R1,I1 AR1 - > R5,I5 *
* I1 := [(I1+I2)+(I3+I4)]/4 AR2 -> R2,I2 AR2 - > R6,I6 *
* I2 := [(I1-I2)-(R3-R4)]/4 ARP-> AR3 -> R3,I3 ARP - > AR3 - > R7,I7 *
* I3 := [(I1+I2)-(I3+I4)]/4 AR4 -> R4,I4 AR4 - > R8,I8 *
* I4 := [(I1-I2)+(R3-R4)]/4 *
* *
* Para una FFT de 16-puntos Radio 2 compleja la Macro 'COMBO2x' debe ser *
* repetida N/4 veces (4 veces para una FFT 16 puntos). *
* *
*****************************************************************
***********
COMBO5x $MACRO  num          ; REPITE MACRO 'COMBO5x': N/4 veces
     SPLK   #:num:-1,BRCR   ; ejecuta 'num' veces 'COMBO5x'
*
     RPTB   comboend       ; ARP AR1 AR2 AR3 AR4 AR5
* --- --- --- --- --- ---
     LACC   *,14,AR4       ;ACC :=(R3)/4 4 R1 R2 R3 R4 T1
     SUB    *,14,AR5       ;ACC :=(R3-R4)/4 5 R1 R2 R3 R4 T1
     SACH   *+,1,AR4       ;T1 =(R3-R4)/2 4 R1 R2 I3 R4 T2
```

```
*
     ADD    *+,15,AR5      ;ACC :=R3+R4)/4 5 R1 R2 R3 I4 T2
     SACH   *,1,AR2        ;T2 =(R3+R4)/2 2 R1 R2 R3 I4 T2
*
     ADD    *,14,AR1       ;ACC :=(R2+R3+R4)/4 1 R1 R2 R3 I4 T2
     ADD    *,14          ;ACC :=(R1+R2+R3+R4)/4 1 R1 R2 R3 I4 T2
     SACH   *+,0,AR5       ;R1 :=(R1+R2+R3+R4)/4 5 I1 R2 R3 I4 T2
     SUB    *,16,AR3       ;ACC :=(R1+R2-(R3+R4))/4 3 I1 R2 R3 I4 T2
     SACH   *+,0,AR5       ;R3 :=(R1+R2-(R3+R4))/4 5 I1 R2 I3 I4 T2
*
     ADD    *,15,AR2       ;ACC :=(R1+R2)/4 2 I1 R2 I3 I4 T2
     SUB    *,15,AR3       ;ACC :=(R1-R2)/4 3 I1 R2 I3 I4 T2
     ADD    *,14,AR4       ;ACC :=((R1-R2)+(I3))/4 4 I1 R2 I3 I4 T2
     SUB    *,14,AR2       ;ACC :=((R1-R2)+(I3-I4))/4 2 I1 R2 I3 I4 T2
     SACH   *+,0,AR4       ;R2 :=((R1-R2)+(I3-I4))/4 4 I1 I2 I3 I4 T2
     ADD    *-,15,AR3      ;ACC :=((R1-R2)+I3+I4)/4 3 I1 I2 I3 R4 T2
     SUB    *,15,AR4       ;ACC :=((R1-R2)-(I3-I4))/4 4 I1 I2 I3 R4 T2
     SACH   *+,0,AR1       ;R4 :=((R1-R2)-(I3-I4))/4 1 I1 I2 I3 I4 T2
*
     LACC   *,14,AR2       ;ACC :=(I1)/4 2 I1 I2 I3 I4 T2
     SUB    *,14,AR5       ;ACC :=(I1-I2)/4 5 I1 I2 I3 I4 T2
     SACH   *,1,AR2        ;T2 :=(I1-I2)/2 2 I1 I2 I3 I4 T2
     ADD    *,15,AR3       ;ACC :=((I1+I2))/4 4 I1 I2 I3 I4 T2
     ADD    *,14,AR4       ;ACC :=((I1+I2)+(I3))/4 4 I1 I2 I3 I4 T2
     ADD    *,14,AR1       ;ACC :=((I1+I2)+(I3+I4))/4 1 I1 I2 I3 I4 T2
     SACH   *0+,0,AR3      ;I1 :=((I1+I2)+(I3+I4))/4 3 R5 I2 I3 I4 T2
     SUB    *,15,AR4       ;ACC :=((I1+I2)-(I3+I4))/4 4 R5 I2 I3 I4 T2
     SUB    *,15,AR3       ;ACC :=((I1+I2)-(I3+I4))/4 3 R5 I2 I3 I4 T2
     SACH   *0+,0,AR5      ;I3 :=((I1+I2)-(I3+I4))/4 5 R5 I2 R7 I4 T2
*
     LACC   *-,15          ;ACC :=(I1-I2)/4 5 R5 I2 R7 I4 T1
     SUB    *,15,AR2       ;ACC :=((I1-I2)-(R3-R4))/4 2 R5 I2 R7 I4 T1
     SACH   *0+,0,AR5      ;I2 :=((I1-I2)-(R3-R4))/4 5 R5 R6 R7 I4 T1
     ADD    *,16,AR4       ;ACC :=((I1-I2)+(R3-R4))/4 4 R5 R6 R7 I4 T1
comboend:
     SACH   *0+,0,AR3 ;I4  :=((I1-I2)+(R3-R4))/4 3 R5 R6 R7 R8 T1
*
     MAR    *,AR2          ;ARP=AR2
     $ENDM
*****************************************************************
************
* *
* MACRO 'ZEROI' numero de palabras : 10 *
```

```
* *
* ARP=2 PARA ENTRADA Y SALIDA *
* AR2 -> QR,QI,QR+1,... *
* AR3 -> PR,PI,PR+1,... *
* *
* CALCULA Re[P+Q] AND Re[P-Q] *
* QR'=(PR-QR)/2 *
* PR'=(PR+QR)/2 *
* PI'=(PI+QI)/2 *
* PI'=(PI-QI)/2 *
* *
*****************************************************************
*************
ZEROI  $MACRO              ; AR1 AR2 ARP
* --- --- ---
     LACC   *,15,AR1      ;ACC :=(1/2)(QR) PR QR 1
     ADD    *,15          ;ACC :=(1/2)(PR+QR) PR QR 1
     SACH   *+,0,AR2      ;PR :=(1/2)(PR+QR) PI QR 2
     SUB    *,16          ;ACC :=(1/2)(PR+QR)-(QR) PI QR 2
     SACH   *+            ;QR :=(1/2)(PR-QR) PI QI 2
*
     LACC   *,15,AR1      ;ACC :=(1/2)(QI) PI QI 1
     ADD    *,15          ;ACC :=(1/2)(PI+QI) PI QI 1
     SACH   *+,0,AR2      ;PI :=(1/2)(PI+QI) PR+1 QI 2
     SUB    *,16          ;ACC :=(1/2)(PI+QI)-(QI) PR+1 QI 2
     SACH   *+            ;QI :=(1/2)(PI-QI) PR+1 QR+1 2
     $ENDM
*****************************************************************
*************
* *
* MACRO 'PBY2I' numero de palabras: 12 *
* *
* PR'=(PR+QI)/2 PI'=(PI-QR)/2 *
* QR'=(PR-QI)/2 QI'=(PI+QR)/2 *
* *
* *
*****************************************************************
*************
PBY2I  $MACRO              ; AR1 AR2 ARP
* --- --- ---
     LACC   *+,15,AR5     ; PR QI 5
     SACH   *,1,AR2       ;TMP=QR PR QI 2
*
```

```
     LACC   *,15,AR1       ;ACC :=QI/2 PR QI 1
     ADD    *,15           ;ACC :=(PR+QI)/2 PR QI 1
     SACH   *+,0,AR2       ;PR :=(PR+QI)/2 PI QI 2
     SUB    *-,16          ;ACC :=(PR-QI)/2 PI QR 2
     SACH   *+,0,AR1       ;QR :=(PR-QI)/2 PI QI 1
*
     LACC   *,15,AR5       ;ACC :=(PI)/2 PI QI 5
     SUB    *,15,AR1       ;ACC :=(PI-QR)/2 PI QI 1
     SACH   *+,0,AR5       ;PI :=(PI-QR)/2 PR+1 QI 5
     ADD    *,16,AR2       ;ACC :=(PI+QR)/2 PR+1 QI 2
     SACH   *+             ;QI :=(PI+QR)/2 PR+1 QI+1 2
     $ENDM
*****************************************************************
************
* *
* MACRO 'PBY4J' numero de palabras: 16 *
* *
* T=SIN(45)=COS(45)=W45 *
* *
* PR'= PR + (W*QI + W*QR) = PR + W * QI + W * QR (<- AR1) *
* QR'= PR - (W*QI + W*QR) = PR - W * QI - W * QR (<- AR2) *
* PI'= PI + (W*QI - W*QR) = PI + W * QI - W * QR (<- AR1+1) *
* QI'= PI - (W*QI - W*QR) = PI - W * QI + W * QR (<- AR1+2) *
* *
*****************************************************************
************
PBY4J  $MACRO               ;TREG =W AR5 PREG AR1 AR2 ARP
* --- ---- --- --- ---
     MPY    *+,AR5         ;PREG =W*QR/2 - W*QR/2 PR QI 5
     SPH    *,AR1          ;TMP =W*QR/2 W*QR/2 W*QR/2 PR QI 1
     LACC   *,15,AR2       ;ACC =PR/2 W*QR/2 W*QR/2 PR QI 2
     MPYS   *-             ;ACC =(PR-W*QR)/2 W*QR/2 W*QI/2 PR QR 2
     SPAC                  ;ACC =(PR-W*QI-W*QR)/2 W*QR/2 W*QI/2 PR QR 2
     SACH   *+,0,AR1       ;QR =(PR-W*QI-W*QR)/2 W*QR/2 W*QI/2 PR QI 1
     SUB    *,16           ;ACC =(-PR-W*QI-W*QR)/2 W*QR/2 W*QI/2 PR QI 1
     NEG                   ;ACC =(PR+W*QI+W*QR)/2 W*QR/2 W*QI/2 PR QI 1
     SACH   *+             ;QR =(PR+W*QI+W*QR)/2 W*QR/2 W*QI/2 PI QI 1
*
     LACC   *,15,AR5       ;ACC =(PI)/2 W*QR/2 W*QI/2 PI QI 5
     SPAC                  ;ACC =(PI-W*QI)/2 W*QR/2 - PI QI 5
     ADD    *,16,AR2       ;ACC =(PI-W*QI+W*QR)/2 - - PI QI 2
     SACH   *+,0,AR1       ;QI =(PI-W*QI+W*QR)/2 - - PI QR1 1
     SUB    *,16           ;ACCU =(-PI-W*QI+W*QR)/2 - - PI QR1 1
```

```
     NEG              ;ACCU =(PI+W*QI-W*QR)/2 - - PI QR1 1
     SACH   *+,0,AR2      ;PI =(PI+W*QI-W*QR)/2 - - PR1 QR1 2
     $ENDM
*****************************************************************
*************
* *
* MACRO 'P3BY4J' numero de palabras: 16 *
* *
* ENTRADA EN LA MACRO: ARP=AR2 *
* AR1->PR,PI *
* AR2->QR,QI *
* TREG=W=COS(45)=SIN(45) *
* *
* PR'= PR + (W*QI - W*QR) = PR + W * QI - W * QR (<- AR1) *
* QR'= PR - (W*QI - W*QR) = PR - W * QI + W * QR (<- AR2) *
* PI'= PI - (W*QI + W*QR) = PI - W * QI - W * QR (<- AR1+1) *
* QI'= PI + (W*QI + W*QR) = PI + W * QI + W * QR (<- AR1+2) *
* *
* SALIDA DE LA MACRO: ARP=AR2 *
* AR1->PR+1,PI+1 *
* AR2->QR+1,QI+1 *
* *
*****************************************************************
*************
P3BY4J  $MACRO            ;TREG =W AR5 PREG AR1 AR2 ARP
* --- ---- --- --- ---
     MPY    *+,AR5       ;PREG =W*QR/2 - W*QR/2 PR QI 5
     SPH    *,AR1        ;TMP =W*QR/2 W*QR/2 W*QR/2 PR QI 1
     LACC   *,15,AR2     ;ACC =PR/2 W*QR/2 W*QR/2 PR QI 2
     MPYA   *-           ;ACC =(PR+W*QR)/2 W*QR/2 W*QI/2 PR QR 2
     SPAC                ;ACC =(PR-W*QI+W*QR)/2 W*QR/2 W*QI/2 PR QR 2
     SACH   *+,0,AR1     ;QR' =(PR-W*QI+W*QR)/2 W*QR/2 W*QI/2 PR QI 1
     SUB    *,16         ;ACC =(-PR-W*QI+W*QR)/2 W*QR/2 W*QI/2 PR QI 1
     NEG                 ;ACC =(PR+W*QI-W*QR)/2 W*QR/2 W*QI/2 PR QI 1
     SACH   *+           ;PR' =(PR+W*QI-W*QR)/2 W*QR/2 W*QI/2 PI QI 1
*
     LACC   *,15,AR5     ;ACC =(PI)/2 W*QR/2 W*QI/2 PI QI 5
     APAC                ;ACC =(PI+W*QI)/2 W*QR/2 - PI QI 5
     ADD    *,16,AR2     ;ACC =(PI+W*QI+W*QR)/2 - - PI QI 2
     SACH   *0+,0,AR1    ;QI' =(PI+W*QI+W*QR)/2 - - PI QR5 1
     SUB    *,16         ;ACCU =(-PI+W*QI+W*QR)/2 - - PI QR5 1
     NEG                 ;ACCU =(PI-W*QI-W*QR)/2 - - PI QR5 1
     SACH   *0+,0,AR2    ;PI' =(PI-W*QI-W*QR)/2 - - PR5 QR5 2
```

```
      $ENDM
*****************************************************************
*************
* *
* MACRO 'stage3' numero de palabras: 54 *
* *
*****************************************************************
*************
stage3  $macro  num
      SPLK    #:num:-1,BRCR   ;ejecuta 'num'-1 veces 'stage3'
      LT      cos45
      RPTB    stage3e
      ZEROI
      PBY4J
      PBY2I
      P3BY4j
stage3e: .set   $-1
      $ENDM
*****************************************************************
*************
* *
* MACRO: 'BUTTFLYI' general butterfly radio 2 para 320C5x *
* *
* LA MACRO 'BUTTFLYI' REQUIERE 18 PALABRAS *
* *
* Definicion: ARP -> AR2 (entrada) ARP -> AR2 (salida) *
* *
* Definicion: AR1 -> QR (entrada) AR1 -> QR+1 (salida) *
* Definicion: AR2 -> PR (entrada) AR2 -> PR+1 (salida) *
* Definicion: AR3 -> Cxxx (entrada) AR3 -> Cxxx+1 (salida) --> WR=coseno *
* Definicion: AR4 -> Sxxx (entrada) AR4 -> Sxxx+1 (salida) --> WI=seno *
* Definicion: AR5 -> variable temporal (sin cambio) *
* *
* usa registro indice *
* *
* PR' = (PR+(QR*WR+QI*WI))/2 WR=COS(W) WI=SIN(W) *
* PI' = (PI+(QI*WR-QR*WI))/2 *
* QR' = (PR-(QR*WR+QI*WI))/2 *
* QI' = (PI-(QI*WR-QR*WI))/2 *
* *
*****************************************************************
*************
BUTTFLYI $MACRO
```

```
* (contenido del registro despues de ejecucion)
* TREG AR1 AR2 AR3 AR4 ARP
      RPTB   btflyend       ; ---- --- --- --- --- ---
      LT     *+,AR3         ;TREG :=QR QR PR QI C S 3
      MPY    *,AR2          ;PREG :=QR*WR/2 QR PR QI C S 2
      LTP    *-,AR4         ;ACC :=QR*WR/2 QI PR QR C S 4
      MPY    *,AR3          ;PREG :=QI*WI/2 QI PR QR C S 3
      MPYA   *+,AR2         ;ACC :=(QR*WR+QI*WI)/2 QR PR QR C+1 S 2
                            ;PREG :=QI*WR
      LT     *,AR5          ;TREG =QR QR PR QR C+1 S 5
      SACH   *,1,AR1        ;H0 :=(QR*WR+QI*WI) QR PR QR C+1 S 1
*
      ADD    *,15           ;ACC :=(PR+(QR*WR+QI*WI))/2 QR PR QR C+1 S 1
      SACH   *+,0,AR5       ;PR :=(PR+(QR*WR+QI*WI))/2 QR PI QR C+1 S 5
      SUB    *,16,AR2       ;ACC :=(PR-(QR*WR+QI*WI))/2 QR PI QR C+1 S 2
      SACH   *+,0,AR1       ;QR :=(PR-(QR*WR+QI*WI))/2 QR PI QI C+1 S 1
*
      LACC   *,15,AR4       ;ACC :=PI/PREG=QI*WR QI PI QI C+1 S 4
      MPYS   *+,AR2         ;PREG :=QR*WI/2 QI PI QI C+1 S+1 2
                            ;ACC :=(PI-QI*WR)/2
      APAC                  ;ACC :=(PI-(QI*WR-QR*WI))/2 QI PI QI C+1 S+1 2
      SACH   *+,0,AR1       ;QI :=(PI-(QI*WR-QR*WI))/2 QI PI QR+1 C+1 S+1 1
      NEG                   ;ACC :=(-PI+(QI*WR-QR*WI))/2 QI PI QR+1 C+1 S+1 1
      ADD    *,16           ;ACC :=(PI+(QI*WR-QR*WI))/2 QI PI QR+1 C+1 S+1 1
btflyend:
      SACH   *+,0,AR2       ;PI :=(PI+(QI*WR-QR*WI))/2 QI PR+1 QR+1 C+1
S+1 2
      $ENDM
; fin de archivo
```

Ejemplo 35. Genera un Diente de Sierra de amplitud 8V y frecuencia 20 Hz

```
;                       Diente.asm
;
; Genera un diente de sierra mediante los siguientes pasos:
; Envía por el puerto serie el contenido de la memoria de datos,
; y lo carga en los bits inferiores del acumulador, ACCL, mientras que
; pone a cer0 los superiores, ACCH. Realiza OR a ACCL con 10 y el resultado
; lo almacena en IMR (registro de enmascaramiento de la interrupción)
; En un lazo suma 10 lo almacena en el ACCL y lo desplaza a la izquierda y
; TRANSMISION por el puerto serie y espera a interrupción y entonces se repite.
*
* Declaración de los registros del mapa de memoria y las direcciones del
* bloque de programa
*
        .mmregs
;
AIC_CTR  .set   9h
;
*
* Programa principal
*
        .ps     0080ah
        B       RINT        ;llamada a la habilitación de interrupciones
        .ps     00a00h      ;el programa comienza en 0xa00
        .entry              ;activación del punto de entrada
        CALL    AICINIT
        LDP     DXR
        LAMM    IMR
        OR      #10h
        SAMM    IMR
;
*
* Lazo generador de la forma de onda
*
LOOP:   ADD     #10
        SACL    DXR,3       ;SACL   DXR, 0h
        IDLE
        B       LOOP
RINT:   LAMM    DRR
        RETE                ;vuelve y habilita las interrupciones INTM=0
;
*
```

```
* Rutina de inicialización del puerto serie del C50 y los
* TA, RA, TB, RB del TLC320C40 del (AIC) y los registros de control
*
AICINIT:
      SPLK   #012h, IMR  ;0001,0010 activa registros de enmascaramiento de
                         ;la interrupción (RINT, INT2)
      SPLK   #20h,TCR    ; Para generar 10 Mhz de Tout
      SPLK   #01h,PRD    ; para el reloj del AIC
      MAR    *,AR0       ;carga los registros auxiliares con 0
      LACC   #0008h      ;Activa el registro de control del puerto serie
      SACL   SPC         ;para modo no continuo y datos de 16 bits.
      LACC   #00c8h      ;carga ACC para escribir RRST=1 y XRST=1
      SACL   SPC
      LACC   #080h       ;Activa el reset del AIC manteniéndolo bajo
      SACH   DXR
      SACL   GREG
      LAR    AR0,#0FFFFh ; y lo toma alto después de 10.000 ciclos
      RPT    #10000      ; (.5ms a 50 ns)
      LACC   *,0,AR0     ;carga datos a FFFF en ACC (0000h).
      SACH   GREG        ;Reseteo de GREG para memoria local de 65K.
      LDP    #AIC_CTR
      LACC   AIC_CTR,2   ; Inicialización del registro de control.
      ADD    #13h        ;
      CALL   AIC_2ND     ; llamada a AIC_2ND
      CLRC   INTM        ; habilita interrupciones
      RET                ;

AIC_2ND:
      LDP    #0          ; puntero de la pagina de datos es 0
      SACH   DXR         ; envía ACChi 00
      CLRC   INTM        ; habilita interrupciones
      IDLE               ; y espera a ellas
      ADD    #6h,15      ; 0000 0000 0000 0011 XXXX XXXX XXXX XXXX b
      SACH   DXR         ; envía ACChi para iniciar el protocolo secundario
      IDLE               ; y espera la interrupción
      SACL   DXR         ; la envía otra vez para asegurarse que la primera
      IDLE               ; palabra ha sido enviada y espera interrupción
      LACL   #0
      SACL   DXR
      IDLE
      SETC   INTM
      RET                ;
       .end
```

Ejemplo 36. Genera una onda senoidal con frecuencia de 200hz y amplitud de 2V.

```
;                       seno.asm
;
;
;                  Generador de Onda Senoidal
;
;
*
* Declaración de los registros del mapa de memoria y las direcciones del
* bloque de programa
*
        .mmregs
        .ds     0f00h
TA      .word   16      ; 16
RA      .word   16      ;  Activa los registros del AIC dando
                        ;  una frecuencia de muestreo de 10.081 Hz
                        ;  mientras que la F(nyquist) = 5040hz
TB      .word   31      ; 31
RB      .word   31      ;
AIC_CTR .word   08h     ; In+ y modo sync
coeff   .word   32514   ; 32514 -> 400Hz
sinx    .word   3834    ;
Ymenos1 .word   400h    ;       1   COSX(Wc)
*
* Programa principal
*
        .ps     0080ah
rint:   B       RECEIVE
xint:   B       TRANSMISION

        .ps     0a00h
        .entry          ;
        ;-------------------------------------------------------

        SETC    INTM
        LDP     #0
        OPL     #0834h,PMST
        LACC    #0
        SAMM    CWSR
        SAMM    PDWSR
        SETC    SXM         ;  SXM debe ser activada
        SPLK    #022h,IMR   ;  Activa el interruptor de recepción
```

```
      CALL   AICINIT      ;
      LDP    #0
      SPLK   #12h,IMR
      CLRC   OVM
      SPM    0            ;
      CLRC   INTM         ;
      LDP    #coeff       ;
      LACC   DURA         ; Carga duración en AR0
      SAMM   AR0
      MAR    *,0
ESPERA:                   ; aquí viene el programa principal
      B      ESPERA       ;

RECEIVE:
      CLRC   INTM         ; Habilita las interrupciones para usos del
                          ; DEBUGGER del DSK
      BANZ   GENR8,0      ; Si duración no ha finalizado genera salida


;-------- genera valor de salida ----------------
GENR8:  ZAP
      LACC   Ymenos1,15     ; y1 ==> ACC desplaza izquierda 15 bit
      NEG                   ; -ACC ==> ACC
      MACD   coeff,sinx     ; coeff * y
      APAC                  ;
      APAC                  ; 2*coeff*y - y1
      SACH   sinx,1
      LACC   sinx,15


      RPT    #15
      SFR
      AND    #0FFFCh        ; bit 0 y 1 debe ser 0 para AIC
      SAMM   DXR            ; para informarle de sus datos ,
      LAMM   DRR            ; para la siguiente interrupción
      RETE                  ;


TRANSMISION:
      RETE
```

```
*************************************************************
; Esta rutina inicializa el puerto serie del C50 y los        *
; TA, RA, TB, RB del TLC320C40 del (AIC) y los registros de control*
;*************************************************************
;
;
AICINIT: SPLK   #20h,TCR        ; Para generar 10 MHz from Tout
      SPLK   #01h,PRD          ; para el reloj del AIC
      MAR    *,AR0
      LACC   #0008h           ; Modo no continuo
      SACL   SPC              ; FSX actua como entrada
      LACC   #00c8h           ; de palabras de 16 bit
      SACL   SPC
      LACC   #080h            ; Activa el reset del AIC manteniéndolo bajo
      SACH   DXR
      SACL   GREG
      LAR    AR0,#0FFFFh
      RPT    #10000           ; y lo toma alto  después de 10.000 ciclos
      LACC   *,0,AR0          ; (.5ms a 50ns)
      SACH   GREG
      ;-----------------------
      LDP    #TA              ;
      SETC   SXM              ;
      LACC   TA,9             ; Inicializa los registros TA y RA
      ADD    RA,2             ;
      CALL   AIC_2ND          ;
      ;-----------------------
      LDP    #TB
      LACC   TB,9             ; Inicializa los registros TB y RB
      ADD    RB,2             ;
      ADD    #02h             ;
      CALL   AIC_2ND          ;
      ;-----------------------
      LDP    #AIC_CTR
      LACC   AIC_CTR,2        ; Inicializa el control de los registros
      ADD    #03h             ;
      CALL   AIC_2ND          ;
      RET                     ;

AIC_2ND:
      LDP    #0               ;El puntero dela pagina de datos es 0(MMregs)
      SACH   DXR              ; envía ACChi 00
      CLRC   INTM             ; habilita interrupciones
      IDLE                    ; espera a interrupción
```

```
ADD   #6h,15    ; 0000 0000 0000 0011 XXXX XXXX XXXX XXXX b
SACH  DXR           ; envía ACChi para iniciar el protocolo secundario
IDLE              ; espera a interrupción
SACL  DXR           ; envía los datos del registro T
IDLE              ; espera a interrupción
LACL  #0            ; pone a 0 ACClo
SACL  DXR           ; envía otra vez para asegurarse que la primera
                  ; palabra ha sido enviada
IDLE              ; espera a interrupción
SETC  INTM
RET               ;
.END
```

Ejemplo 37. Generador de Barrido Senoidal.

```
;                    Barrido.asm
;
;
; El barrido senoidal genera una onda senoidal cuya frecuencia varia,
; comenzando en 200Hz e incrementándose hasta 4800Hz donde rebota al
; máximo valor y comienza a decrecer hasta alcanzar el mínimo valor donde
; rebota y comienza de nuevo el ciclo.
;
; PASO es el cambio en frecuencia cuando aumenta/ disminuye la frecuencia
; DURACION es el tiempo (# de interrupciones) antes que se tome la nueva
; etapa. El step size no debe exceder de 150 ya que los pasos deben ser inferiores
; al coefficiente de Nyquist. Si son inferiores al coeficiente de Nyquist
; (5KHz max) se pueden producir efectos de aliasing.
;
;
*
* Declaración de los registros del mapa de memoria y las direcciones del
* bloque de programa
*
      .mmregs
      .ds    0f00h
TA    .word  16      ; 16
RA    .word  16      ;  Activa los registros del AIC dando
                     ;  una frecuencia de muestreo de 10.081 Hz
                     ;  mientras que la F(nyquist) = 5040hz
TB    .word  31      ; 31
RB    .word  31      ;
AIC_CTR .word  08h     ; In+ y modo sync
coeff .word  32514   ; 32514 -> 400Hz
sinx  .word  3834    ;
Ymenos1 .word  400h    ;
PASO  .word  150     ; cambio en frecuencia (tamaño de PASO)
DURA  .word  010h    ; # de interrupciones antes de PASO (duración)
MAX   .word  -32514  ; Max = 4800Hz
MIN   .word  32514   ; Min = 200Hz

      .ps    0080ah
rint: B     RECIBIR
xint: B     TRANSMISIONIR
*
* Programa principal
```

```
    .ps     0a00h
    .entry          ;
    ;-------------------------------------------------------

    SETC   INTM
    LDP    #0
    OPL    #0834h,PMST
    LACC   #0
    SAMM   CWSR
    SAMM   PDWSR
    SETC   SXM        ; SXM debe ser activada
    SPLK   #022h,IMR   ; Activa el interruptor de recepción

    CALL   AICINIT     ;
    LDP    #0
    SPLK   #12h,IMR
    CLRC   OVM
    SPM    0          ;
    CLRC   INTM       ;
    LDP    #coeff
    LACC   DURA       ; Carga duración en AR0
    SAMM   AR0
    MAR    *,0

ESPERA:               ; aquí viene el programa principal
    B     ESPERA      ;

RECIBIR:
    CLRC   INTM       ; Habilita las interrupciones para usos del
                      ; DEBUGGER del DSK
    BANZ   GENR8,0    ; Si duración no ha finalizado genera salida

;--- mira si F >= Fmax
    LACC   MAX
    SUB    coeff
    BCND   UP,LT      ; Si ACC < 0 entonces F < Fmax y desplaza arriba
    LACC   PASO
    NEG               ; sino F >= Fmax bounce back y desplaza abajo
    SACL   PASO


;--- mira si F <= Fmin
```

```
UP    LACC    MIN       ; si ACC > 0 entonces F < Fmin y desplaza abajo
      SUB     coeff
      BCND    CONT,GT
      LACC    PASO      ; sino F <= Fmin bounce back y desplaza arriba
      ABS               ; toma el valor absoluto (siempre positivo)
      SACL    PASO

;--- Continua
CONT  LACC    coeff     ; Cambia frecuencia por tamaño de PASO
      SUB     PASO
      SACL    coeff
      LACC    DURA      ; resetea duración
      SAMM    AR0

;-------- genera valor de salida ----------------
GENR8:  ZAP
      LACC    Ymenos1,15      ; y1 ==> ACC shift left 15 bit
      NEG                     ; -ACC ==> ACC
      MACD    coeff,sinx      ; coeff * y
      APAC                    ;
      APAC                    ; 2*coeff*y - y1
      SACH    sinx,1
      LACC    sinx,15

      RPT     #15
      SFR
      AND     #0FFFCh         ; bit 0 y 1 debe ser 0 para AIC
      SAMM    DXR             ; para informarle de sus datos
      LAMM    DRR             ; para la siguiente interrupción
      RETE                    ;


 TRANSMISIONIR:
      RETE


 ***************************************************************
 ; Esta rutina inicializa el puerto serie del C50 y los        *
 ; TA, RA, TB, RB del TLC320C40 del (AIC) y los registros de control*
 ***************************************************************
 ;
 ;
 AICINIT: SPLK   #20h,TCR        ; Para generar 10 MHz from Tout
       SPLK    #01h,PRD          ; para el reloj del AIC
```

```
      MAR    *,AR0
      LACC   #0008h          ; Modo no continuo
      SACL   SPC             ; FSX actúa como entrada
      LACC   #00c8h          ; de palabras de 16 bit
      SACL   SPC
      LACC   #080h           ; Activa el reset del AIC manteniéndolo bajo
      SACH   DXR
      SACL   GREG
      LAR    AR0,#0FFFFh
      RPT    #10000          ; y lo toma alto  después de 10.000 ciclos
      LACC   *,0,AR0         ; (.5ms a 50ns)
      SACH   GREG
      ;-----------------------
      LDP    #TA             ;
      SETC   SXM             ;
      LACC   TA,9            ; Inicializa los registros TA y RA
      ADD    RA,2            ;
      CALL   AIC_2ND         ;
      ;-----------------------
      LDP    #TB
      LACC   TB,9            ; Inicializa los registros TB y RB
      ADD    RB,2            ;
      ADD    #02h            ;
      CALL   AIC_2ND         ;
      ;-----------------------
      LDP    #AIC_CTR
      LACC   AIC_CTR,2       ; Inicializa el control de los registros
      ADD    #03h            ;
      CALL   AIC_2ND         ;
      RET                    ;

AIC_2ND:
      LDP    #0              ; El puntero de la pag de datos es 0(mmregs)
      SACH   DXR             ; envía ACChi 00
      CLRC   INTM            ; habilita interrupciones
      IDLE                   ; espera a interrupción
      ADD    #6h,15          ; 0000 0000 0000 0011 XXXX XXXX XXXX XXXX b
      SACH   DXR             ; envía ACChi para iniciar el protocolo 2io
      IDLE                   ; espera a interrupción
      SACL   DXR             ; envía los datos del registro T
      IDLE                   ; espera a interrupción
      LACL   #0              ; pone a 0 ACClo
      SACL   DXR             ; envía otra vez para asegurarse que la 1a
```

```
                ; palabra ha sido enviada
IDLE                ; espera a interrupción
SETC    INTM
RET                 ;
.END
```

Ejemplo 38. Toma una señal de entrada analógica y la envía a la salida analógica con una ganancia global de 1

```
*                          IN-OUT.ASM
*
*
*
* PASO-BANDA: ACTIVADO
*         Frecuencia de corte baja: 200 Hz
*         Frecuencia corte alta: 2.5 kHz
*         Frecuencia de muestreo: 8 kHz
*
* CONDICIONES DE MANEJO:
*
*       ****************************************
*       * CUIDADO:  VOLTAJE MAXIMO de ENTRADA: 3 V
*       ****************************************
*
*          La ENTRADA ANALOG < - la Función Generada
*
*          La SALIDA ANALOG - > Osciloscopio
*
*************************************************************
;
      .mmregs             ; Declaración de registros del mapa de memoria
;
*************************************************************
* Declaración e inicialización de los registros TA, RA, TB, RB y AIC_CTR en la
*  memoria de datos
*************************************************************
      .ds    0f00h        ; Memoria de datos del espacio del usuario
TA      .word   25         ; Fcorte = 2.5 KHz
RA      .word   25
TAp     .word   31
RAp     .word   31
TB      .word   25         ; Fs = 8 kHz
RB      .word   25         ;
AIC_CTR .word   19h        ; ganancia total=1,sync,loopback deshabilitados
                           ; filtro de paso-banda habilitado
;
      .ps    080ah      ; Memoria de los vectores de interrupción
rint:   B    RECIBIR    ;080Ah: Interrupción de recepción del puerto serie
xint:   B    TRANSMISION IR  ;080Ch: Interrupción de transmisión del puerto serie
```

```
;
*
* PRINCIPIO DEL PROGRAMA PRINCIPAL
*
;
        .ps 0a00h           ; Memoria del programa de usuario
       .entry               ; punto de entrada del programa
*
* Inicialización del DSP TMS320C50
*
INICIO:  SETC  INTM         ; Deshabilita interrupciones
       LDP   #0             ; Activa el puntero de la pagina de datos
       OPL   #0834h, PMST   ;comienzo de la tabla de interrupción en 0800h
                            ; Programa y datos ON-CHIP en la memoria SARAM
                            ; (RAM = 1 y OVLY = 1)
                     ; Modo de la microcomputadora MP/ MC = 0
                     ; NDX = 1, TRM = 0 => C5X modo no realzado
       LACC  #0
       SAMM  CWSR     ; Pone cero en el acceso de estado espera a memoria
        SAMM  PDWSR
;
* Resetea el AIC escribiendo PA2 (dirección > 52) para DSK
       SPLK  #022h,IMR  ; Usando XINT para syn ch(e INT2 para debugger)
       CALL  AICINIT    ; Llamada a subrutina de inicialización del AIC
;
       CLRC   OVM         ; OVM = 0 - modo de Excedente
       SPM    0           ; PM = 0 ningún desplazamiento después de reg
                     ; el producto.
       SPLK   #012h,IMR   ; RINT solo (e INT2 para debugger)
       CLRC   INTM        ; habilita interrupción
;
*
* Lazo infinito - en espera de recepción de datos
*
* NB: La interrupción RINT es habilitada
*
ESPERA  IDLE               ; espera a interrupción
       B      ESPERA       ; Espera a entrada de datos
;
*
* Rutina del servicio de interrupción de la recepción
*
RECIBIR:
```

```
      LAMM  DRR       ; Lee los datos de DRR
      AND   #0FFFCh   ; resolución de 14 bits del AIC
      SAMM  DXR       ; Envía los datos a DXR (el comunicador primario)
      RETE
;
TRANSMISIONIR:
      RETE
;
;
;
*
* La rutina AICINIT inicializa el Puerto Serie y el TLC320C40 AIC,
*  mediante los siguientes pasos:
* Paso 1: Ajusta la frecuencia Timer (10.000 MHz para AIC MCLK).
* Paso 2: Configura señales de sincronización del marco.
* Paso 3: Inicialización del puerto serie
* Paso 4: Inicialización del AIC
*
* NB: La línea AIC Reset esta conectada con el pin BR (Bus Request) del 'C50.
* Este pin es puesto a bajo cuando la memoria global externa se accede a la
* memoria global externa. Entonces para resetear el AIC, la memoria global
* debe estar definida y accesible para llevar al pin del /BR bajo en
* al menos 800ns.
*
AICINIT: SPLK   #20h,TCR     ; Para generar  10.000 MHz de Tout
      SPLK   #01h,PRD     ; Para el reloj del  AIC
      MAR    *,AR0
      LACC   #0008h      ; Modo discontinuo, FSM = 1,FSX como la entrada
      SACL   SPC         ; resetea el puerto serie
      LACC   #00c8h      ; palabras de 16 bits
      SACL   SPC         ; envía el puerto serie para reanulación
      LACC   #080h       ; Inicializa 8000h, FFFFh como memoria global
      SACH   DXR         ; La palabra falsa envía a aclarar registro SPC
      SACL   GREG        ; Almacena el registro de direccionamiento
                         ; de la memoria global
      LAR    AR0,#0FFFFh ; Usa AR0 para apuntar la posición FFFFh
      RPT    #10000      ; Accede a la memoria global 10000 veces
      LACC   *,0,AR0     ; Para propulsar pin bajo para la duración
      SACH   GREG        ; Restaura  GREG a 0000
;
*
* NB: Enviado como parámetro requiere una "comunicación secundaria",
*       por ello la llamada a la subrutina AIC_2ND.
*
```

```
        ;
       LDP    #TA        ; Inicialización del puntero de la pagina de datos
       SETC   SXM        ; Activa el modo de la extensión del signo
       LACC   TA, 9      ; TA es desplazada a la izquierda 9 bits
       ADD    RA, 2      ; RA es desplazada a la izquierda 2 bits
       CALL   AIC_2ND    ; Llama a la subrutina AIC _ 2ND
        ;-------------------------
       LDP    #TB        ; Inicialización del puntero de la pagina de datos
       LACC    TB, 9     ; TB es desplazada a la izquierda 9 bits
       ADD     RB, 2     ; RB es desplazada a la izquierda 2 bits
       ADD     # 02h
       CALL    AIC_2ND
        ;----------------------
       LDP    #AIC_CTR   ; Inicialización del puntero de la pagina de datos
       LACC   AIC_CTR,2  ; AIC_CTR es desplazada a la izquierda 2 bits
       ADD    # 03h
       CALL   AIC_2ND
        RET;
;
*
* La rutina AIC_2ND inicializa una transmisión secundaria,
* con los siguientes pasos:
* Paso 1: Habilita la interrupción de la transmisión del puerto serie
* Paso 2: Envía 03h para iniciar una transmisión secundaria
* Paso 3: Espera hasta que este listo para los nuevos datos (idle)
* Paso 4: Envía parámetro AIC
* Paso 5: Desactiva la interrupción de la transmisión del puerto serie
*
*
AIC_2ND:
      LDP   #0
      SACH  DXR        ; La palabra falsa envía a aclarar registro SPC
      CLRC  INTM       ; Habilita interrupción
      IDLE             ; Esperar a interrupción de la transmisión
      ADD   #6h,15     ; 0000 0000 0000 0011 | Palabras de datos del
                       ; acumulador  DX
      SACH  DXR        ; Envía  0003h para iniciar transmisión 2ia
       IDLE
      SACL  DXR        ; Manda palabras de datos del AIC DX
       IDLE
      LACL  #0
      SACL  DXR        ; Se asegura que la palabra sea enviada
       IDLE
```

```
SETC   INTM        ; deshabilita interrupciones
  RET
.END
```

Ejemplo 39. MODULADOR DE AMPLITUD

```
;                          modamp.asm
; -> FILTRO FIR
;   Este programa usa un filtro 80 tap FIR para implementar un filtro
; paso banda. EL filtro deja pasar frecuencias entre 200Hz-1.8KHz.
;
; -> AIC
;   Para que el DSK funcione a 40 Mhz la frecuencia de muestreo es
;   10.000.000/2/TA/ TB en la transmisión y 10.000.000/2/RA/ RB en la
;   recepción.  El valor máximo para TA, TB, RA, RB es 63 mientras que
;   los registros sean de 6 bits.
;
; -> MODULACION
;   La señal entrante se captura y se carga en el ACC. A la muestra se le
;   aplica un filtrado paso banda (BW = 200Hz a 1.8Kz). Se genera una señal
;   portadora de 2.0Khz usando las variables coeff, senox y Ymenos1.
;   En este punto a la señal filtrada se le multiplica por la portadora.
;   La salida es una señal de frecuencia modulada a 2KHz con dos lóbulos
;   laterales (HS, LS).
;                            |
;                          |   |   |
;  Lado bajo = Fc-BW   ->   ||||| | ||||| Lado alto = Fc+BW
;                         ||||||| | |||||||
;
;                         |-LS--| Fc|-HS--|
;
;   Conectando a la salida  un analizador de frecuencias se puede ver este
;   grafico. La salida puede ser oída si se conecta un altavoz
;
;
;
;
; ---------------------------------------------------------------
;
        .MMREGS
        .ds     0f00h
TA      .word   16      ;
RA      .word   16      ; Activación de los registros del AIC
                        ; dando una frecuencia de muestreo de 10.081 Hz
                        ;
TB      .word   31      ;
RB      .word   31      ;
```

```
AIC_CTR .word   0ch
coef    .word   5219      ; 10438
senox   .word   15531     ; 31061
Ymenos1 .word   02bfh     ;


;Coeficiente Generador del filtro  paso banda @ flcorte=200hz  f2corte= 1800Hz

h0      .word       0   ; 40      0.0000
h1      .word       1   ; 39      0.0001
h2      .word       0   ; 38      0.0000
h3      .word       1   ; 37      0.0001
h4      .word      11   ; 36      0.0013
h5      .word      31   ; 35      0.0037
h6      .word      39   ; 34      0.0047
h7      .word      17   ; 33      0.0020
h8      .word     -11   ; 32     -0.0013
h9      .word      13   ; 31      0.0016
h10     .word     101   ; 30      0.0123
h11     .word     161   ; 29      0.0197
h12     .word      86   ; 28      0.0105
h13     .word     -91   ; 27     -0.0111
h14     .word    -169   ; 26     -0.0206
h15     .word      -0   ; 25     -0.0000
h16     .word     252   ; 24      0.0307
h17     .word     203   ; 23      0.0248
h18     .word    -288   ; 22     -0.0351
h19     .word    -812   ; 21     -0.0991
h20     .word    -772   ; 20     -0.0942
h21     .word    -157   ; 19     -0.0191
h22     .word     206   ; 18      0.0251
h23     .word    -503   ; 17     -0.0614
h24     .word   -1955   ; 16     -0.2386
h25     .word   -2716   ; 15     -0.3315
h26     .word   -1832   ; 14     -0.2237
h27     .word    -294   ; 13     -0.0359
h28     .word    -341   ; 12     -0.0416
h29     .word   -2846   ; 11     -0.3474
h30     .word   -5680   ; 10     -0.6934
h31     .word   -5541   ;  9     -0.6763
h32     .word   -1952   ;  8     -0.2382
h33     .word    1107   ;  7      0.1352
h34     .word   -1191   ;  6     -0.1453
```

```
h35    .word   -8518   ;  5     -1.0398
h36    .word   -13568  ;  4     -1.6562
h37    .word   -7758   ;  3     -0.9471
h38    .word    9971   ;  2      1.2171
h39    .word   30016   ;  1      3.6640
h40    .word   30016   ;  1      3.6640
h41    .word    9971   ;  2      1.2171
h42    .word   -7758   ;  3     -0.9471
h43    .word   -13568  ;  4     -1.6562
h44    .word   -8518   ;  5     -1.0398
h45    .word   -1191   ;  6     -0.1453
h46    .word    1107   ;  7      0.1352
h47    .word   -1952   ;  8     -0.2382
h48    .word   -5541   ;  9     -0.6763
h49    .word   -5680   ; 10     -0.6934
h50    .word   -2846   ; 11     -0.3474
h51    .word    -341   ; 12     -0.0416
h52    .word    -294   ; 13     -0.0359
h53    .word   -1832   ; 14     -0.2237
h54    .word   -2716   ; 15     -0.3315
h55    .word   -1955   ; 16     -0.2386
h56    .word    -503   ; 17     -0.0614
h57    .word     206   ; 18      0.0251
h58    .word    -157   ; 19     -0.0191
h59    .word    -772   ; 20     -0.0942
h60    .word    -812   ; 21     -0.0991
h61    .word    -288   ; 22     -0.0351
h62    .word     203   ; 23      0.0248
h63    .word     252   ; 24      0.0307
h64    .word      -0   ; 25     -0.0000
h65    .word    -169   ; 26     -0.0206
h66    .word     -91   ; 27     -0.0111
h67    .word      86   ; 28      0.0105
h68    .word     161   ; 29      0.0197
h69    .word     101   ; 30      0.0123
h70    .word      13   ; 31      0.0016
h71    .word     -11   ; 32     -0.0013
h72    .word      17   ; 33      0.0020
h73    .word      39   ; 34      0.0047
h74    .word      31   ; 35      0.0037
h75    .word      11   ; 36      0.0013
h76    .word       1   ; 37      0.0001
h77    .word       0   ; 38      0.0000
```

```
h78   .word     1   ; 39      0.0001
h79   .word     0   ; 40      0.0000


XN     .word   0,0,0,0,0,0,0,0,0,0   ; 80 localizaciones de datos
XN1    .word   0,0,0,0,0,0,0,0,0,0   ; para 80 líneas de retraso
XN2    .word   0,0,0,0,0,0,0,0,0,0   ; de estado
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   0,0,0,0,0,0,0,0,0,0    ;
XNLAST  .word   0                    ; palabra extra para el bit bucket
SALIDA  .word   0

      .ps     0080ah
rint:  B      RECEPCION
xint:  B      TRANSMISION

      .ps     0a00h
      .entry
      ;-----------------------------------------------------------

      SETC   INTM
      LDP    #0
      OPL    #0834h,PMST
      LACC   #0
      SAMM   CWSR
      SAMM   PDWSR
      SETC   SXM          ;  SXM debe ser activado
      SPLK   #022h,IMR     ; Esto activa únicamente la interrupción
                           ; de recepción
      CALL   AICINIT       ;
      SPLK   #12h,IMR
      CLRC   OVM
      SPM    0          ;
      CLRC   INTM         ;
ESPERA:                  ; el programa principal puede venir aquí
      B      ESPERA      ;

RECEPCION:
      LDP    #XN
      CLRC   INTM         ; Habilitación de interrupciones para usar en el
```

```
                    ; DEBUGGER del DSK
      ZAP
;-------- generamos la portadora -----
      LACC   Ymenos1,15    ; y1 ==> ACC trasladar a la izquierda 15 bits
      NEG                  ; -ACC ==> ACC
      MACD   coef, senox   ; coef * y
      APAC                 ;
      APAC                 ; 2*coef*y - y1
      SACH   senox,1       ; Reduce la salida en al menos 1/8
      ZAP
;-------- toma una muestra y lo ejecuta a través del filtro paso banda ------

      LAMM   DRR           ; carga el ACC con palabra recibida desde AIC
      SACL   XN,1          ; almacena el valor de la palabra recibida
                           ; a una variable
      LAR    AR0,#XNLAST   ; carga AR0 con la dirección del ultimo elemento
      ZAP                  ; se pone a 0 ACC y registros de productos
      MAR    *,AR0         ; AR0 es el registro AR actual
      RPT    #79           ; Repite la siguiente instrucción 80 veces
      MACD   #h0,*-
      APAC                 ; ACC ultimo producto
      SACH   SALIDA,1
      ZAP
;----------modulación de paso banda AM en portador

      MAC    senox,SALIDA  ; Multiplica la portadora por la salida filtrada
      APAC                 ; Producto -> ACC
      SACH   SALIDA,1      ; guarda y rechaza el bit del signo extra.
      LACC   SALIDA,15     ;
      RPT    #13           ; Aclara el ACC y desplaza la palabra a bajo(x4)
      SFR                  ;
      AND    #0FFFCh       ; los 2 bits LSB's deben ser cero para el AIC
      SAMM   DXR           ; Envía a los registros de transmisión
      RETE


TRANSMISION:
      RETE
```

```
*****************************************************************
;  Esta rutina inicializa el puerto serie 'C50 y los TA, RA, TB, RB *
;   del TLC320C40's (AIC) y los registros de control          *
*****************************************************************
;
;
AICINIT: SPLK   #20h,TCR        ; Para generar 10 MHz desde Tout
      SPLK   #01h,PRD         ; para el reloj del AIC
      MAR    *,AR0
      LACC   #0008h          ; Modo discontinuo
      SACL   SPC             ; FSX como entrada de
      LACC   #00c8h          ; palabras de 16 bits
      SACL   SPC
      LACC   #080h           ; Activa el reset del AIC reset
                  ; manteniéndolo bajo
      SACH   DXR
      SACL   GREG
      LAR    AR0,#0FFFFh
      RPT    #10000          ; y tomándolo alto después de 10.000 ciclos
      LACC   *,0,AR0         ; (.5ms en 50ns)
      SACH   GREG
      ;-----------------------
      LDP    #TA             ;
      SETC   SXM             ;
      LACC   TA,9            ; Inicializa los registros TA y RA
      ADD    RA,2            ;
      CALL   AIC_2ND         ;
      ;-----------------------
      LDP    #TB
      LACC   TB,9            ; Inicializa los registros TB y RB
      ADD    RB,2            ;
      ADD    #02h            ;
      CALL   AIC_2ND         ;
      ;-----------------------
      LDP    #AIC_CTR
      LACC   AIC_CTR,2       ; Inicializa los registros de control
      ADD    #03h            ;
      CALL   AIC_2ND         ;
      RET                    ;

AIC_2ND:
      LDP    #0              ; Puntero de la pagina de datos es 0(MM regs)
      SACH   DXR             ; envía al ACChi 00
      CLRC   INTM            ; habilita interrupciones
```

```
IDLE                ; ESPERA a interrupción
ADD   #6h,15    ; 0000 0000 0000 0011 XXXX XXXX XXXX XXXX b
SACH  DXR           ; envía ACChi para iniciar el protocolo 2
                ; secundario
IDLE                ; ESPERA a interrupción
SACL  DXR           ; envía los datos del registro T
IDLE                ; ESPERA a interrupción
LACL  #0            ; limpia ACClo
SACL  DXR           ; envía otra vez para asegurarse que la
                ; primera palabra ha sido enviada
IDLE                ; ESPERA a interrupción
SETC  INTM          ;
RET                 ;
.END
```

Ejemplo 40. Implementación de un filtro IIR de orden N.

```
                    filtroiir.asm
; d(n) = x(n)   + d(n-1)a1 + d(n-2)a2 + ... + d(n-N+1)aN-1
; y(n) = d(n)b0 + d(n-1)b1 + d(n-2)b2 + ... + d(n-N+1)bN-1
; Requerimientos de memoria:
; *Variables de estado (memoria de datos bajo a alto) d(n) d(n-1) .. d(n-N+1)
; *coeficientes(memoria de programa bajo a alto) a(N-1) .. a(1) b(N-1) .. b(0)
; Condiciones de entrada:
; AR0 -> entrada
; AR1 ->  d(n-N+1)
; AR2 -> salida
; BMAR -> a(n-1)
; ARP = ARO
      .mmregs
;
IIR_N: ZPR
      LACC   *,15,AR1       ; Pone Q15 entrada
      RPT    #(N-2)         ; Lazo for i=1, i<=N-1, ++i
      MADS   *-             ; ACC+ = a(n-N+i)
      APAC                  ; Acumulación final
      SACH   *,1            ; Salva d(n)
      ADRK   N-1            ; AR1 -> d(n-N+1)
      LAMM   BMAR           ; ACC -> a(N-1)
      ADD    #N-1           ; ACC -> b(N-1)
      SAMM   BMAR           ; BMAR -> b(N-1)
      RPTZ   #(N-1)         ; Lazo for i=1, i<=N, ++i
      MADD   *-             ; ACC+ = b(N-i)*d(n-N+i)
      LTA    *,AR2          ; Acumulación final
      SACH   *,1            ; Salva Y(n)
```

Ejemplo 41. Un filtro Fir

```
;                    fir161.asm
;
; Las secciones de transmisión y recepción están en modo síncror
; El generador de tono  en la interrupción de transmisión
;
      .mmregs
      .ds    ofooh
;
TA     .word  18
RA     .word  18
TB     .word  18     ; frecuencia de conversión = 15.432Hz
RB     .word  18
aic_ctr .word  08h    ; xmit y Rcv son sincronos
TIME   .word  0d60h   ; duracion del pulso
TIMER  .word  0000h
DIGIT  .word  0000h
; Coeficientes requeridos paras la generación de seno
C1     .word  25736   ; 6488h       1.570796327
C3     .word -10583   ; d6a9h      -0.6459640968
C5     .word  1306    ; 051ah       0.07969260
C7     .word  -77     ; ffb4h      -0.004681666
C9     .word  3       ; 0003h       0.00016
sinput .word  0
SALIDA .word  0
temp   .word  0       ; espacio temp
x2     .word  0       ; guarda a x2 (x al cuadrado)
x      .word  0       ; guarda a x
```

Ejemplo 42. FILTRO PASOBAJO.

```
;
;  -> FILTRO FIR
;     Este programa usa un filtro 80 tap FIR para implementar un filtro
;     Paso bajo. El filtro deja pasar únicamente frecuencias menores 1.0KHz.
;
;  -> AIC
;     Parámetros de set-up del AIC set-up . Para que el DSK funcione a 40 Mhz,
;     la frecuencia de muestreo es 10.000.000/2/TA/ TB en la transmisión y
;     10.000.000/2/RA /RB en la recepción. El valor máximo para TA, ;TB, RA,
;     RB es 63 mientras que los registros sean de 6 bits.
;
;     GENERADOR DE RUIDO ALEATORIO
;  -> En este código se incluye un generador de ruido aleatorio para permitir
;     al usuario escuchar las diferencias de los filtrados a la salida o
;     verlos usando un analizador de frecuencias. El generador de ruido
;     aleatorio puede ser bipaseada colocando un ; en la primera columna de
;     cada estado asm del generador de ruido aleatorio
;
;
;
;
; ---------------------------------------------------------------------

        .MMREGS
        .ds     0f00h
TA      .word   16      ;
RA      .word   16      ;  Establece los registros del AIC
                        ;  dando una frecuencia de muestreo de 10.081 Hz
                        ;
TB      .word   31      ;
RB      .word   31      ;
AIC_CTR .word   08h
SALIDA  .word   0
TEMP    .word   0       ; localización del almacenamiento temporal
TEMP1   .word   0       ;
origen  .word   07e6dh  ; origen del generador de ruido aleatorio

;Coeficiente Generador de filtro   pasobajo @ fcorte=1K
h0      .word       0   ; 40      0.0000
h1      .word     -157  ; 39      -0.0048
h2      .word     -261  ; 38      -0.0080
```

```
h3      .word    -268   ; 37     -0.0082
h4      .word    -170   ; 36     -0.0052
h5      .word      0    ; 35     -0.0000
h6      .word     180   ; 34      0.0055
h7      .word     301   ; 33      0.0092
h8      .word     310   ; 32      0.0095
h9      .word     198   ; 31      0.0060
h10     .word      0    ; 30     0.0000
h11     .word    -211   ; 29     -0.0065
h12     .word    -354   ; 28     -0.0108
h13     .word    -367   ; 27     -0.0112
h14     .word    -236   ; 26     -0.0072
h15     .word      0    ; 25     -0.0000
h16     .word     255   ; 24      0.0078
h17     .word     431   ; 23      0.0132
h18     .word     451   ; 22      0.0138
h19     .word     292   ; 21      0.0089
h20     .word      0    ; 20     0.0000
h21     .word    -323   ; 19     -0.0098
h22     .word    -551   ; 18     -0.0168
h23     .word    -584   ; 17     -0.0178
h24     .word    -383   ; 16     -0.0117
h25     .word      0    ; 15     -0.0000
h26     .word     438   ; 14      0.0134
h27     .word     763   ; 13      0.0233
h28     .word     827   ; 12      0.0252
h29     .word     557   ; 11      0.0170
h30     .word      0    ; 10     0.0000
h31     .word    -681   ;  9     -0.0208
h32     .word   -1240   ;  8     -0.0378

h33     .word   -1417   ;  7     -0.0432
h34     .word   -1022   ;  6     -0.0312
h35     .word      0    ;  5     -0.0000
h36     .word    1533   ;  4      0.0468
h37     .word    3307   ;  3      0.1009
h38     .word    4960   ;  2      0.1514
h39     .word    6131   ;  1      0.1871
;cero   .word    6554   ;  0      0.2000
h40     .word    6131   ;  1      0.1871
h41     .word    4960   ;  2      0.1514
h42     .word    3307   ;  3      0.1009
h43     .word    1533   ;  4      0.0468
```

```
h44    .word      0   ;  5      -0.0000
h45    .word   -1022   ;  6       -0.0312
h46    .word   -1417   ;  7       -0.0432
h47    .word   -1240   ;  8       -0.0378
h48    .word    -681   ;  9       -0.0208

h49    .word      0   ; 10       0.0000
h50    .word    557   ; 11        0.0170
h51    .word    827   ; 12        0.0252
h52    .word    763   ; 13        0.0233
h53    .word    438   ; 14        0.0134
h54    .word      0   ; 15       -0.0000
h55    .word   -383   ; 16       -0.0117
h56    .word   -584   ; 17       -0.0178
h57    .word   -551   ; 18       -0.0168
h58    .word   -323   ; 19       -0.0098
h59    .word      0   ; 20       0.0000
h60    .word    292   ; 21        0.0089
h61    .word    451   ; 22        0.0138
h62    .word    431   ; 23        0.0132
h63    .word    255   ; 24        0.0078
h64    .word      0   ; 25       -0.0000
h65    .word   -236   ; 26       -0.0072
h66    .word   -367   ; 27       -0.0112
h67    .word   -354   ; 28       -0.0108
h68    .word   -211   ; 29       -0.0065
h69    .word      0   ; 30       0.0000
h70    .word    198   ; 31        0.0060
h71    .word    310   ; 32        0.0095
h72    .word    301   ; 33        0.0092
h73    .word    180   ; 34        0.0055
h74    .word      0   ; 35       -0.0000
h75    .word   -170   ; 36       -0.0052
h76    .word   -268   ; 37       -0.0082
h77    .word   -261   ; 38       -0.0080
h78    .word   -157   ; 39       -0.0048
h79    .word      0   ; 40       0.0000


XN     .word   0,0,0,0,0,0,0,0,0,0   ; 80 localizaciones de datos para 80
XN1    .word   0,0,0,0,0,0,0,0,0,0   ; líneas de retraso de estado
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;

        .ps     0080ah
rint:   B       RECIBIR         ; para recibir
xint:   B       TRANSMISION IR      ; para TRANSMISION

        .ps     0a00h
        .entry              ;
        ;-------------------------------------------------------------

        SETC    INTM
        LDP     #0
        OPL     #0834h,PMST
        LACC    #0
        SAMM    CWSR
        SAMM    PDWSR
        SETC    SXM         ; SXM debe ser activado
        SPLK    #022h,IMR   ; activa únicamente el interruptor de recepción

        CALL    AICINIT     ;
        SPLK    #12h,IMR
        CLRC    OVM
        SPM     0           ;
        CLRC    INTM        ;

ESPERA:                     ; aquí debe ir un programa principal
        B       ESPERA      ;

RECIBIR:
        LDP     #XN
        CLRC    INTM        ; habilita interrupciones para usos de
                            ; DEBUGGER del DSK

; ----- Generador de Ruido Aleatorio ----------------
        LACC    origen,1    ;
        XOR     origen      ;
        SACL    TEMP,2      ;
        XOR     TEMP ;      ;
        AND     #8000h      ;
```

```
      ADD    origen,16      ;
      SACH   origen,1       ; Reduce la salida en al menos 1/8
      LACC   origen,11      ; para prevenir el overflow
      AND    #0FFFCh,15     ;
      RPT    #14            ;
      SFR                   ;

; ----- Fin del generador de ruido --------------------
                            ; La siguiente línea no debería estar comentada si
                            ; el código del generador de encima esta comentado.
;     LAMM DRR              ; carga acumulador con palabras recibidas del AIC
      SACL   XN             ; almacena valor de la palabra recibida en variable

      LAR    AR0,#XNLAST ; carga AR0 con la dirección del ultimo elemento
                           ; de retardo
      ZAP                  ; ZERO ACC Y REGISTROS DE PRODUCTO
      MAR    *,AR0         ; AR0 es el registro AR usado

      RPT    #79           ; Repite la siguiente instrucción 80 veces a
      MACD   #h0 ,*-       ; través de la tabla completa de coeficientes.
      APAC                 ; Acumula el ultimo producto.

      SACH   SALIDA,1      ; ACC -> SALIDA
      LACC   SALIDA        ;
      SFL
      AND    #0fffch       ; Dos LSB's deben ser cero para el AIC
      SAMM   DXR           ; escribe la palabra de salida para TRANSMISIÓN iir
                           ; al registro
      RETE                 ;


TRANSMISIONIR:
      RETE


;*****************************************************************
;  DESCRIPCION: Esta rutina inicializa el puerto serie 'C50     *
;           y el TLC320C40's (AIC) TA, RA, TB, RB y los         *
;           registros de control                                *
;*****************************************************************
;
AICINIT: SPLK   #20h,TCR          ; Para generar 10 MHz desde Tout
      SPLK   #01h,PRD          ; para reloj del AIC
```

```
        MAR    *,AR0
        LACC   #0008h          ; Modo no continuo
        SACL   SPC             ; FSX como entrada
        LACC   #00c8h          ; palabras 16 bit
        SACL   SPC
        LACC   #080h           ; Pone el AIC a reset manteniéndolo bajo
        SACH   DXR
        SACL   GREG
        LAR    AR0,#0FFFFh
        RPT    #10000          ; y lo toma alto después de 10.000 ciclos
        LACC   *,0,AR0         ; (.5ms a 50ns)
        SACH   GREG
        ;-----------------------
        LDP    #TA             ;
        SETC   SXM             ;
        LACC   TA,9            ; Inicializa registros TA y RA
        ADD    RA,2            ;
        CALL   AIC_2ND         ;
        ;-----------------------
        LDP    #TB
        LACC   TB,9            ; Inicializa registros TB y RB
        ADD    RB,2            ;
        ADD    #02h            ;
        CALL   AIC_2ND         ;
        ;-----------------------
        LDP    #AIC_CTR
        LACC   AIC_CTR,2       ; Inicializa registro de control
        ADD    #03h            ;
        CALL   AIC_2ND         ;
        RET                    ;

AIC_2ND:
        LDP    #0              ; Puntero de pagina de datos es 0 (MM regs)
        SACH   DXR             ; envía ACChi 00
        CLRC   INTM            ; habilita interrupciones
        IDLE                   ; espera interrupción
        ADD    #6h,15          ; 0000 0000 0000 0011 XXXX XXXX XXXX XXXX b
        SACH   DXR             ; envía ACChi para iniciar protocolo secund
        IDLE                   ; espera interrupción
        SACL   DXR             ; envia los datos del registro T
        IDLE                   ; espera interrupción
        LACL   #0              ; borra ACClo
        SACL   DXR             ; envía otra para asegurar que 1 palabra sea enviada
```

```
IDLE            ; espera interrupción
SETC   INTM
RET             ;
.END
```

Ejemplo 43. FILTROS PASO-ALTO.

```
;
;  -> FILTRO FIR
;     Este programa utiliza un 81 tap FIR filter para implementar un
;     filtro PASOALTO. El filtro deja pasar frecuencias mayores de 3KHz.
;
;  -> AIC
;     Parámetros de set-up del AIC. Para el DSK trabajando a 40 mhz, la
;     frecuencia de muestreo es 10,000,000/2/TA/TB para TRANSMISION ir y es
;     10,000,000/2/RA/ RB para recibir. El valor máx para TA, TB, RA, RB es 63
;     mientras los registros sean de 6 bits.
;
;     GENERADOR DE RUIDO ALEATORIO
;  -> Un generador de ruido aleatorio se incluye en este código para permitir al
;     usuario oír las diferencias en la salida filtrada o verlas utilizando un
;     analizador de frecuencias. El generador de ruido aleatorio puede ser
;     bipaseada colocando un punto y coma ; en la columna de mas a la
;     izquierda de cada estado asm del generador de ruido aleatorio.
;
;
;
;
;
; ---------------------------------------------------------------------

        .MMREGS
        .ds     0f00h
TA      .word   16      ;
RA      .word   16      ; Este establecimiento de los registros AIC
                        ; dan una frecuencia de muestreo de 10.081 Hz
                        ;
TB      .word   31      ;
RB      .word   31      ;
AIC_CTR .word   08h
SALIDA  .word   0
TEMP    .word   0               ; localización del almacenamiento temporal
TEMP1   .word   0               ;
origen  .word   07e6dh          ; origen para el generador de ruido aleatorio

; Frec Fl corte filtro Pasoalto = 3000hz aprox.
h0      .word       0   ; 40      -0.0000
h1      .word       0   ; 39       0.0001
h2      .word      -1   ; 38      -0.0001
```

```
h3      .word     -3    ; 37     -0.0003
h4      .word      8    ; 36      0.0010
h5      .word      0    ; 35      0.0000
h6      .word    -19    ; 34     -0.0024
h7      .word     17    ; 33      0.0021
h8      .word     24    ; 32      0.0029
h9      .word    -52    ; 31     -0.0063
h10     .word      0    ; 30     -0.0000
h11     .word     87    ; 29      0.0107
h12     .word    -68    ; 28     -0.0084
h13     .word    -86    ; 27     -0.0105
h14     .word    172    ; 26      0.0210
h15     .word      0    ; 25      0.0000
h16     .word   -256    ; 24     -0.0312
h17     .word    191    ; 23      0.0233
h18     .word    228    ; 22      0.0279
h19     .word   -440    ; 21     -0.0537
h20     .word      0    ; 20     -0.0000
h21     .word    612    ; 19      0.0747
h22     .word   -443    ; 18     -0.0541
h23     .word   -518    ; 17     -0.0632
h24     .word    975    ; 16      0.1190
h25     .word      0    ; 15      0.0000
h26     .word  -1311    ; 14     -0.1601
h27     .word    938    ; 13      0.1145
h28     .word   1086    ; 12      0.1326
h29     .word  -2037    ; 11     -0.2486
h30     .word      0    ; 10     -0.0000
h31     .word   2763    ; 9      0.3373
h32     .word  -2007    ; 8      -0.2450
h33     .word  -2384    ; 7      -0.2911
h34     .word   4653    ; 6      0.5680
h35     .word      0    ; 5      0.0000
h36     .word  -7345    ; 4      -0.8966
h37     .word   6161    ; 3      0.7521
h38     .word   9359    ; 2      1.1425
h39     .word -30518    ; 1      -3.7254
Czero   .word  32767
h40     .word -30518    ; 1      -3.7254
h41     .word   9359    ; 2      1.1425
h42     .word   6161    ; 3      0.7521
h43     .word  -7345    ; 4      -0.8966
h44     .word      0    ; 5      0.0000
```

```
h45   .word    4653   ;  6      0.5680
h46   .word   -2384   ;  7     -0.2911
h47   .word   -2007   ;  8     -0.2450
h48   .word    2763   ;  9      0.3373
h49   .word       0   ; 10     -0.0000
h50   .word   -2037   ; 11     -0.2486
h51   .word    1086   ; 12      0.1326
h52   .word     938   ; 13      0.1145
h53   .word   -1311   ; 14     -0.1601
h54   .word       0   ; 15      0.0000
h55   .word     975   ; 16      0.1190
h56   .word    -518   ; 17     -0.0632
h57   .word    -443   ; 18     -0.0541
h58   .word     612   ; 19      0.0747
h59   .word       0   ; 20     -0.0000
h60   .word    -440   ; 21     -0.0537
h61   .word     228   ; 22      0.0279
h62   .word     191   ; 23      0.0233
h63   .word    -256   ; 24     -0.0312
h64   .word       0   ; 25      0.0000
h65   .word     172   ; 26      0.0210
h66   .word     -86   ; 27     -0.0105
h67   .word     -68   ; 28     -0.0084
h68   .word      87   ; 29      0.0107
h69   .word       0   ; 30     -0.0000
h70   .word     -52   ; 31     -0.0063
h71   .word      24   ; 32      0.0029
h72   .word      17   ; 33      0.0021
h73   .word     -19   ; 34     -0.0024
h74   .word       0   ; 35      0.0000
h75   .word       8   ; 36      0.0010
h76   .word      -3   ; 37     -0.0003
h77   .word      -1   ; 38     -0.0001
h78   .word       0   ; 39      0.0001
h79   .word       0   ; 40     -0.0000


XN     .word   0,0,0,0,0,0,0,0,0,0   ; 80 localizaciones de datos para 80
XN1    .word   0,0,0,0,0,0,0,0,0,0   ; líneas de retraso de estado
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    0,0,0,0,0,0,0,0,0,0   ;
XNLAST  .word    0;

        .ps     0080ah
rint:   B       RECEIVE
xint:   B       TRANSMISION

        .ps     0a00h
        .entry              ;
        ;-------------------------------------------------------------

        SETC    INTM
        LDP     #0
        OPL     #0834h,PMST
        LACC    #0

        SAMM    CWSR
        SAMM    PDWSR
        SETC    SXM         ;  SXM debe ser activado
        SPLK    #022h,IMR     ; Esto activa únicamente las interrupciones
                        ; de recepción

        CALL    AICINIT      ;
        SPLK    #12h,IMR
        CLRC    OVM
        SPM     1           ;
        CLRC    INTM        ;

ESPERA:                     ; un programa principal iría aquí
        B       ESPERA       ;

RECEIVE:
        LDP     #XN
        CLRC    INTM        ; Habilitación de interrupciones para usar en el
                        ; Debugger del DSK

; ----- Generador de ruido aleatorio ----------------
        LACC    origen,1     ;
        XOR     origen      ;
        SACL    TEMP,2       ;
        XOR     TEMP ;      ;
        AND     #8000h      ;
```

```
        ADD     origen,16       ;
        SACH    origen,1        ; Reduce la salida por lo menos 1/8
        LACC    origen,11       ; para prevenir el overflow
        AND     #0FFFCh,15      ;
        RPT     #14             ;
        SFR                     ;

; ----- Fin del Generador de ruido ----------------

                        ; La siguiente línea no debería estar comentada si
                        ; el código del generador de encima esta comentado.
;       LAMM DRR                ; carga acumulador con palabras recibidas del AIC
        SACL    XN              ; almacena valor de la palabra recibida en variable

        LAR     AR0,#XNLAST ; carga AR0 con la dirección del ultimo elemento
                        ; de retardo
        ZAP                     ; ZERO ACC Y REGISTROS DE PRODUCTO
        MAR     *,AR0           ; AR0 es registro actual AR

        RPT     #80             ; Repite la siguiente instrucción 81 veces a
        MACD    #h0 ,*-         ; través de la tabla completa de coeficientes.
        APAC                    ; Acumula el ultimo producto.

        SACH    SALIDA,1        ; ACC -> SALIDA  Un bit extra de signo.
        LACC    SALIDA          ;
        SFL
        AND     #0fffch         ; Dos LSB's deben ser cero para el AIC
        SAMM    DXR             ; escribe la palabra de salida para TRANSMISIÓN iir
                        ; al registro
        RETE                    ;


TRANSMISION:
        RETE


;*****************************************************************
;  DESCRIPCION: Esta rutina inicializa el puerto serie 'C50      *
;               y el TLC320C40's (AIC) TA, RA, TB, RB y los      *
;               registros de control                             *
;*****************************************************************
;
;
AICINIT: SPLK   #20h,TCR        ; Para generar 10 MHz desde Tout
```

```
        SPLK   #01h,PRD          ; para AIC master clock
        MAR    *,AR0
        LACC   #0008h            ; Modo no continuo
        SACL   SPC               ; FSX como entrada
        LACC   #00c8h            ; palabras 16 bit
        SACL   SPC
        LACC   #080h             ; Pone el AIC a reset manteniéndolo bajo
        SACH   DXR
        SACL   GREG
        LAR    AR0,#0FFFFh
        RPT    #10000            ; y lo toma alto después de 10000 ciclos
        LACC   *,0,AR0           ; (.5ms a 50ns)
        SACH   GREG
        ;-----------------------
        LDP    #TA               ;
        SETC   SXM               ;
        LACC   TA,9              ; Inicializa registros TA y RA
        ADD    RA,2              ;
        CALL   AIC_2ND           ;
        ;-----------------------
        LDP    #TB
        LACC   TB,9              ; Inicializa registros TB y RB
        ADD    RB,2              ;
        ADD    #02h              ;
        CALL   AIC_2ND           ;
        ;-----------------------
        LDP    #AIC_CTR
        LACC   AIC_CTR,2         ; Inicializa registro de control
        ADD    #03h              ;
        CALL   AIC_2ND           ;
        RET                      ;

AIC_2ND:
        LDP    #0                ; Puntero de pagina de datos es 0 (MM regs)
        SACH   DXR               ; envía ACChi 00
        CLRC   INTM              ; habilita interrupciones
        IDLE                     ; espera interrupción
        ADD    #6h,15            ; 0000 0000 0000 0011 XXXX XXXX XXXX XXXX b
        SACH   DXR               ; envía ACChi para iniciar protocolo secund
        IDLE                     ; espera interrupción
        SACL   DXR               ; envia los datos del registro T
        IDLE                     ; espera interrupción
        LACL   #0                ; borra ACClo
```

```
SACL   DXR          ; envía otra para asegurar que 1 palabra sea enviada
IDLE               ; espera interrupción
SETC   INTM

RET                ;
.END
```

Ejemplo 44. FILTROS PASO-BANDA.

```
;
;  -> FILTRO FIR
;     Este programa utiliza un 81 tap FIR filter para implementar un filtro
PASOBANDA.
;     El filtro deja pasar frecuencias entre 3 y 4 KHz.
;
;  -> AIC
;     Parámetros de set-up del AIC. Para el DSK trabajando a 40 Mhz, la frecuencia
;     de muestreo es 10,000,000/2/TA/ TB para TRANSMISION ir y es
10,000,000/2/RA/ RB
;     para recibir. El valor máximo para TA, TB, RA, RB es de 6 mientras los
;     registros sean de 6 bits.
;
;     GENERADOR DE RUIDO ALEATORIO
;  -> Un generador de ruido aleatorio se incluye en este código para permitir al
;     usuario oír las diferencias en la salida filtrada o verlas utilizando un analizador
;     de frecuencias. El generador de ruido aleatorio puede ser bipaseada colocando
;     un punto y coma ; en la columna de mas a la izquierda de cada estado asm del
;     generador de ruido aleatorio.
;
;
;
;
;
;
;
;
        .MMREGS
        .ds     0f00h
TA      .word   16      ;
RA      .word   16      ;  Este establecimiento de los registros AIC
                        ;  dan una frecuencia de muestreo de 10.081 Hz
                        ;
TB      .word   31      ;
RB      .word   31      ;
AIC_CTR .word   08h
SALIDA  .word   0
TEMP    .word   0               ; localización del almacenamiento temporal
TEMP1   .word   0               ;
origen  .word   07e6dh          ; origen para el generador de ruido aleatorio

;Generador de Coeficiente de Filtro   pasobanda @ flcorte=3K  fhcorte= 4K
```

```
h0      .word     0   ; 40      0.0000
h1      .word     0   ; 39      0.0000
h2      .word     1   ; 38      0.0001
h3      .word    -6   ; 37     -0.0004
h4      .word     5   ; 36      0.0003
h5      .word    22   ; 35      0.0013
h6      .word   -50   ; 34     -0.0031
h7      .word    13   ; 33      0.0008
h8      .word   100   ; 32      0.0061
h9      .word  -153   ; 31     -0.0093
h10     .word     0   ; 30     -0.0000
h11     .word   259   ; 29      0.0158
h12     .word  -287   ; 28     -0.0175
h13     .word   -65   ; 27     -0.0040
h14     .word   445   ; 26      0.0271
h15     .word  -354   ; 25     -0.0216
h16     .word  -156   ; 24     -0.0096
h17     .word   466   ; 23      0.0284
h18     .word  -226   ; 22     -0.0138
h19     .word  -105   ; 21     -0.0064
h20     .word     0   ; 20      0.0000
h21     .word   146   ; 19      0.0089
h22     .word   439   ; 18      0.0268
h23     .word -1264   ; 17     -0.0771
h24     .word   596   ; 16      0.0364
h25     .word  1904   ; 15      0.1162
h26     .word -3394   ; 14     -0.2072
h27     .word   712   ; 13      0.0434
h28     .word  4549   ; 12      0.2777
h29     .word -6030   ; 11     -0.3681
h30     .word     0   ; 10     -0.0000
h31     .word  8181   ; 9      0.4993
h32     .word -8408   ; 8     -0.5132
h33     .word -1810   ; 7     -0.1105
h34     .word 12047   ; 6      0.7353
h35     .word -9661   ; 5     -0.5897
h36     .word -4489   ; 4     -0.2740
h37     .word 15040   ; 3      0.9180
h38     .word -9255   ; 2     -0.5649
h39     .word -7293   ; 1     -0.4451
Ccero   .word  18688   ; 0      0.1000
h40     .word -7293   ; 1     -0.4451
h41     .word -9255   ; 2     -0.5649
```

```
h42   .word   15040   ;  3      0.9180
h43   .word   -4489   ;  4     -0.2740
h44   .word   -9661   ;  5     -0.5897
h45   .word   12047   ;  6      0.7353
h46   .word   -1810   ;  7     -0.1105
h47   .word   -8408   ;  8     -0.5132
h48   .word    8181   ;  9      0.4993
h49   .word       0   ; 10     -0.0000
h50   .word   -6030   ; 11     -0.3681
h51   .word    4549   ; 12      0.2777
h52   .word     712   ; 13      0.0434
h53   .word   -3394   ; 14     -0.2072
h54   .word    1904   ; 15      0.1162
h55   .word     596   ; 16      0.0364
h56   .word   -1264   ; 17     -0.0771
h57   .word     439   ; 18      0.0268
h58   .word     146   ; 19      0.0089
h59   .word       0   ; 20      0.0000
h60   .word    -105   ; 21     -0.0064
h61   .word    -226   ; 22     -0.0138
h62   .word     466   ; 23      0.0284
h63   .word    -156   ; 24     -0.0096
h64   .word    -354   ; 25     -0.0216
h65   .word     445   ; 26      0.0271
h66   .word     -65   ; 27     -0.0040
h67   .word    -287   ; 28     -0.0175
h68   .word     259   ; 29      0.0158
h69   .word       0   ; 30     -0.0000
h70   .word    -153   ; 31     -0.0093
h71   .word     100   ; 32      0.0061
h72   .word      13   ; 33      0.0008
h73   .word     -50   ; 34     -0.0031
h74   .word      22   ; 35      0.0013
h75   .word       5   ; 36      0.0003
h76   .word      -6   ; 37     -0.0004
h77   .word       1   ; 38      0.0001
h78   .word       0   ; 39      0.0000
h79   .word       0   ; 40      0.0000


XN     .word   0,0,0,0,0,0,0,0,0,0   ; 80 localizaciones de datos para 80
XN1    .word   0,0,0,0,0,0,0,0,0,0   ; líneas de retraso de estado
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    0,0,0,0,0,0,0,0,0,0   ;
XNLAST  .word    0;

      .ps     0080ah
rint:  B       RECEIVE
xint:  B       TRANSMISION

      .ps      0a00h
      .entry             ;
      ;-------------------------------------------------------

      SETC   INTM
      LDP    #0
      OPL    #0834h,PMST
      LACC   #0
      SAMM   CWSR
      SAMM   PDWSR
      SETC   SXM         ; SXM debe ser activado
      SPLK   #022h,IMR   ; Esto activa únicamente las interrupciones
                         ; de recepción

      CALL   AICINIT     ;
      SPLK   #12h,IMR
      CLRC   OVM
      SPM    0           ;
      CLRC   INTM        ;

ESPERA:                  ; un programa principal iría aquí
      B      ESPERA      ;

RECEIVE:
      LDP    #XN
      CLRC   INTM        ; Habilitación de interrupciones para usar en el
                         ; Debugger del DSK

;---- Generador de Ruido Aleatorio ----------------------
      LACC   origen,1    ;*
      XOR    origen      ;*
      SACL   TEMP,2      ;*
```

```
      XOR    TEMP ;      ;*
      AND    #8000h      ;*
      ADD    origen,16     ;*
      SACH   origen,1      ;* Reduce la salida por lo menos 1/8
      LACC   origen,11     ;* para prevenir el overflow
      AND    #0FFFCh,15  ;*
      RPT    #14         ;* Cuando NO se use el generador de ruido aleatorio,
      SFR                ;* asegurarse de comentar esta sección usando ;'s

;---- Fin del Generador de Ruido ----------------------

                  ; Cuando no se use el generador de ruido, la
                  ; siguiente línea NO debería ser comentada.
;     LAMM   DRR          ; carga el acumulador con la palabra recibida del AIC
      SACL   XN           ; almacena el valor de la palabra recibida en una variable

      LAR    AR0,#XNLAST ; carga AR0 con la dirección del ultimo elemento
retrasado
      ZAP                ; ZERO ACC Y PRODUCTOS DE LOS REGISTROS
      MAR    *,AR0       ; AR0 es el registro AR actual

      RPT    #80         ; Repite la siguiente instrucción 81 veces por
      MACD   #h0,*-      ; la tabla completa de coeficientes.
      APAC               ; Acumula el ultimo producto.

      SACH   SALIDA,1    ; ACC -> SALIDA  Un bit extra de signo.
      LACC   SALIDA      ; SALIDA -> ACC
      SFL                ; SALIDA*2
      AND    #0fffch     ; Los dos últimos deben ser cero para el AIC
      SAMM   DXR         ; escribe la palabra de salida para TRANSMISIÓN ir
                         ; al registro
      RETE


TRANSMISION:
      RETE


*****************************************************************
;  DESCRIPCION: Esta rutina inicializa el puerto serie 'C50      *
;           y el TLC320C40's (AIC) TA, RA, TB, RB y los          *
;           registros de control                                 *
;*****************************************************************
```

```
;
AICINIT: SPLK   #20h,TCR        ; Para generar 10 MHz desde Tout
      SPLK   #01h,PRD          ; para el AIC master clock
      MAR    *,AR0
      LACC   #0008h           ; Modo no continuo
      SACL   SPC              ; FSX como entrada
      LACC   #00c8h           ; palabras 16 bit
      SACL   SPC
      LACC   #080h            ; Pone el AIC a reset manteniéndolo bajo
      SACH   DXR
      SACL   GREG
      LAR    AR0,#0FFFFh
      RPT    #10000           ; y lo toma alto después de 10000 ciclos
      LACC   *,0,AR0          ; (.5ms a 50ns)
      SACH   GREG
      ;-----------------------
      LDP    #TA              ;
      SETC   SXM              ;
      LACC   TA,9             ; Inicializa registros TA y RA
      ADD    RA,2             ;
      CALL   AIC_2ND          ;
      ;-----------------------
      LDP    #TB
      LACC   TB,9             ; Inicializa registros TB y RB
      ADD    RB,2             ;
      ADD    #02h             ;
      CALL   AIC_2ND          ;
      ;-----------------------
      LDP    #AIC_CTR
      LACC   AIC_CTR,2        ; Inicializa registro de control
      ADD    #03h             ;
      CALL   AIC_2ND          ;
      RET                     ;

AIC_2ND:
      LDP    #0               ; Puntero de pagina de datos es 0  (MM regs)
      SACH   DXR              ; envía ACChi 00
      CLRC   INTM             ; habilita interrupciones
      IDLE                    ; espera interrupción
      ADD    #6h,15           ; 0000 0000 0000 0011 XXXX XXXX XXXX XXXX b
      SACH   DXR              ; envía ACChi para iniciar protocolo secundario
      IDLE                    ; espera interrupción
      SACL   DXR              ; envía los datos del registro T
```

```
IDLE            ; espera interrupción
LACL   #0         ; borra ACClo
SACL   DXR          ; envía otra para asegurar 1¦ palabra fue enviada
IDLE            ; espera interrupción
SETC   INTM
RET             ;
.END
```

Ejemplo 45. FILTROS RECHAZO BANDA.

```
;
; -> FILTRO FIR
;    Este programa utiliza un 81 tap FIR filter para implementar un filtro
;    RECHAZO BANDA. El filtro deja pasar frecuencias entre 1 y 3.5 KHz.
;
; -> AIC
;    Parámetros de set-up del AIC. Para el DSK trabajando a 40 mhz, la
;    frecuencia de muestreo es 10,000,000/2/TA/TB para TRANSMISIÓN ir y es
;10,000,000/2/RA/RB para recibir. El valor máximo para TA, TB, RA, RB es 63
;    mientras los registros sean de 6 bits.
;
;    GENERADOR DE RUIDO ALEATORIO
; -> Un generador de ruido aleatorio se incluye en este código para permitir
;    al usuario oír las diferencias en la salida filtrada o verlas utilizando
;    un analizador de frecuencias. El generador de ruido aleatorio puede ser
;    bipaseada colocando un punto y coma ; en la columna de mas a la izquierda
;    de cada estado asm del generador de ruido aleatorio.
;
;
;
;
;
;
; ---------------------------------------------------------------------

      .MMREGS
      .ds    0f00h
TA    .word  16      ;
RA    .word  16      ; Este establecimiento de los registros AIC
                     ; dan una frecuencia de muestreo de 10.081 Hz
                     ;

TB    .word  31      ;
RB    .word  31      ;
AIC_CTR .word  08h
SALIDA  .word  0
TEMP    .word  0          ; localización del almacenamiento temporal
TEMP1   .word  0          ;
origen  .word  07e6dh     ; origen para el generador de ruido aleatorio

;Generador de Coeficiente de Filtro   rechazo banda flcorte=1K  fhcorte = 3.5K
h0    .word      0   ; 40     -0.0000
```

```
h1      .word     0    ; 39     0.0000
h2      .word    -3    ; 38    -0.0004
h3      .word    -2    ; 37    -0.0003
h4      .word     0    ; 36     0.0000
h5      .word   -11    ; 35    -0.0013
h6      .word    19    ; 34     0.0023
h7      .word    29    ; 33     0.0036
h8      .word     0    ; 32    -0.0000
h9      .word    61    ; 31     0.0074
h10     .word     0    ; 30     0.0000
h11     .word  -103    ; 29    -0.0125
h12     .word     0    ; 28    -0.0000
h13     .word  -147    ; 27    -0.0179
h14     .word  -170    ; 26    -0.0207
h15     .word   177    ; 25     0.0216
h16     .word     0    ; 24     0.0000
h17     .word   167    ; 23     0.0203
h18     .word   591    ; 22     0.0722
h19     .word   -82    ; 21    -0.0100
h20     .word     0    ; 20    -0.0000
h21     .word   114    ; 19     0.0139
h22     .word -1148    ; 18    -0.1401
h23     .word  -452    ; 17    -0.0552
h24     .word     0    ; 16     0.0000
h25     .word  -952    ; 15    -0.1162
h26     .word  1296    ; 14     0.1583
h27     .word  1609    ; 13     0.1964
h28     .word     0    ; 12    -0.0000
h29     .word  2393    ; 11     0.2921
h30     .word     0    ; 10     0.0000
h31     .word -3246    ; 9     -0.3963
h32     .word     0    ; 8     -0.0000
h33     .word -4089    ; 7     -0.4991
h34     .word -4601    ; 6     -0.5617
h35     .word  4831    ; 5      0.5897
h36     .word     0    ; 4     0.0000
h37     .word  5384    ; 3      0.6572
h38     .word 24230    ; 2      2.9578
h39     .word -5679    ; 1     -0.6933
Ccero   .word 32767
h40     .word -5679    ; 1     -0.6933
h41     .word 24230    ; 2      2.9578
h42     .word  5384    ; 3      0.6572
```

```
h43   .word     0   ;  4      0.0000
h44   .word    4831   ;  5     0.5897
h45   .word    -4601   ;  6    -0.5617
h46   .word    -4089   ;  7    -0.4991
h47   .word     0   ;  8      -0.0000
h48   .word    -3246   ;  9    -0.3963
h49   .word     0   ;  10      0.0000
h50   .word    2393   ;  11     0.2921
h51   .word     0   ;  12     -0.0000
h52   .word    1609   ;  13     0.1964
h53   .word    1296   ;  14     0.1583
h54   .word    -952   ;  15    -0.1162
h55   .word     0   ;  16      0.0000
h56   .word    -452   ;  17    -0.0552
h57   .word    -1148   ;  18   -0.1401
h58   .word    114   ;  19      0.0139
h59   .word     0   ;  20      -0.0000
h60   .word    -82   ;  21     -0.0100
h61   .word    591   ;  22      0.0722
h62   .word    167   ;  23      0.0203
h63   .word     0   ;  24      0.0000
h64   .word    177   ;  25      0.0216
h65   .word    -170   ;  26     -0.0207
h66   .word    -147   ;  27     -0.0179
h67   .word    -0   ;  28      -0.0000
h68   .word    -103   ;  29    -0.0125
h69   .word     0   ;  30      0.0000
h70   .word    61   ;  31      0.0074
h71   .word     0   ;  32      -0.0000
h72   .word    29   ;  33      0.0036
h73   .word    19   ;  34      0.0023
h74   .word    -11   ;  35     -0.0013
h75   .word     0   ;  36      0.0000
h76   .word    -2   ;  37      -0.0003
h77   .word    -3   ;  38      -0.0004
h78   .word     0   ;  39      0.0000
h79   .word     0   ;  40      -0.0000


XN     .word   0,0,0,0,0,0,0,0,0,0   ; 80 localizaciones de datos para 80
XN1    .word   0,0,0,0,0,0,0,0,0,0   ; líneas de retraso de estado
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    0,0,0,0,0,0,0,0,0,0   ;
XNLAST  .word    0;

      .ps     0080ah
rint:   B       RECEIVE
xint:   B       TRANSMISION

      .ps     0a00h
      .entry              ;
      ;-------------------------------------------------------

      SETC    INTM
      LDP     #0
      OPL     #0834h,PMST
      LACC    #0
      SAMM    CWSR
      SAMM    PDWSR
      SETC    SXM         ; SXM debe ser activado
      SPLK    #022h,IMR   ; Esto activa únicamente las interrupciones
                          ; de recepción

      CALL    AICINIT     ;
      SPLK    #12h,IMR
      CLRC    OVM
      SPM     0           ;
      CLRC    INTM        ;

ESPERA:                   ; un programa principal iría aquí
      B       ESPERA      ;

RECEIVE:
      LDP     #XN
      CLRC    INTM        ; Habilitación de interrupciones para usar en el
                          ; Debugger del DSK

;----- Generador de Ruido Aleatorio ----------
      LACC    origen,1    ;*
      XOR     origen      ;*
      SACL    TEMP,2      ;*
      XOR     TEMP ;      ;*
```

```
      AND   #8000h      ;*
      ADD   origen,16   ;*
      SACH  origen,1    ;* Reduce la salida por lo menos 1/8
      LACC  origen,11   ;* para prevenir el overflow
      AND   #0FFFCh,15  ;*
      RPT   #14         ;* Cuando NO se use el generador ruido aleatorio,
      SFR               ;* asegurarse de comentar esta sección usando ;'s

; ----- Fin del Generador de Ruido ----------


                  ; La siguiente línea no debería ser comentada si
                  ; el código del generador es comentado.
;     LAMM  DRR         ; carga el acumulador con la palabra recibida AIC
      SACL  XN          ; almacena valor de la palabra recibida en variable

      LAR   AR0,#XNLAST ; carga AR0 en dirección ultimo elemento retardo
      ZAP               ; ZERO ACC Y PRODUCTOS DE LOS REGISTROS
      MAR   *,AR0       ; AR0 es registro actual AR.

      RPT   #80         ; Repite la siguiente instrucción 81 veces a
      MACD  #h0,*-      ; través de la tabla completa de coeficientes.
      APAC              ; Acumula el ultimo producto.

      SACH  SALIDA,1    ; ACC -> SALIDA  Un bit extra de signo
      LACC  SALIDA
      SFL
      AND   #0fffch     ; Los dos LSB's deben ser cero para el AIC
      SAMM  DXR         ; escribe palabra de salida para TRANSMISIÓN ir
                  ; al registro
      RETE              ;


TRANSMISION:
      RETE


;*****************************************************************
; DESCRIPCION: Esta rutina inicializa el puerto serie 'C50      *
;           y el TLC320C40's (AIC) TA, RA, TB, RB y los         *
;           registros de control                                *
;*****************************************************************
;
;
```

```
AICINIT: SPLK   #20h,TCR        ; Para generar 10 MHz desde Tout
      SPLK   #01h,PRD          ; para el AIC master clock
      MAR    *,AR0
      LACC   #0008h           ; Modo no continuo
      SACL   SPC              ; FSX como entrada
      LACC   #00c8h           ; palabras 16 bit
      SACL   SPC
      LACC   #080h            ; Pone el AIC a reset manteniéndolo bajo
      SACH   DXR
      SACL   GREG
      LAR    AR0,#0FFFFh
      RPT    #10000           ; y lo toma alto después de 10.000 ciclos
      LACC   *,0,AR0          ; (.5ms a 50ns)
      SACH   GREG
      ;-----------------------
      LDP    #TA              ;
      SETC   SXM              ;
      LACC   TA,9             ; Inicializa registros TA y RA
      ADD    RA,2             ;
      CALL   AIC_2ND          ;
      ;-----------------------
      LDP    #TB
      LACC   TB,9             ; Inicializa registros TB y RB
      ADD    RB,2             ;
      ADD    #02h             ;
      CALL   AIC_2ND          ;
      ;-----------------------
      LDP    #AIC_CTR
      LACC   AIC_CTR,2        ; Inicializa registro de control
      ADD    #03h             ;
      CALL   AIC_2ND          ;
      RET                     ;

AIC_2ND:
      LDP    #0               ; Puntero de pagina de datos es 0 (MM regs)
      SACH   DXR              ; envía ACChi 00
      CLRC   INTM             ; habilita interrupciones
      IDLE                    ; espera interrupción
      ADD    #6h,15           ; 0000 0000 0000 0011 XXXX XXXX XXXX XXXX b
      SACH   DXR              ; envía ACChi para iniciar protocolo secund
      IDLE                    ; espera interrupción
      SACL   DXR              ; envía los datos del registro T
      IDLE                    ; espera interrupción
```

4. PLIEGO DE CONDICIONES

1.- CONDICIONES ADMINISTRATIVAS .

1.1.- Condiciones generales.

La aprobación definitiva de lo ofertado en este proyecto incluye expresamente la aprobación de nuestras condiciones generales de venta. Cualquier modificación de las mismas, para surtir efecto, deberá ser hecha por escrito y aceptada.

La obra se realizará con sujeción a los documentos del proyecto.

En caso de aceptación del proyecto, no nos responsabilizaremos de realizarlo obligatoriamente. A la vista de perfeccionamientos que puedan realizarse, si después de aceptado el presente proyecto y presupuesto, resultase conveniente para la instalación, alguna mejora no prevista con antelación, o algún aparato, se pondrá en conocimiento del usuario; pero queda bien entendido que no tenemos la obligación de suministrar gratuitamente dichas mejoras o elementos adicionales.

Una vez terminado y revisado por los organismos oficiales pertinentes, declinará todo tipo de reclamaciones.

Los precios del material presupuestado y de los jornales están sujetos a modificaciones que se pueden producir durante el plazo de entrega del proyecto-y suministro, fijando entonces los precios en el día de suministro.

1.2.- Obligaciones del contratista en el montaje.

Este estará obligado a ejecutar los trabajos para la realización de la obra, siguiendo fielmente los documentos del proyecto, así como las órdenes del Ingeniero Técnico en lo referente a partes dentro del transcurso de la instalación.

Los errores cometidos deberán modificarse sin derecho a indemnización, aunque el Ingeniero lo puede aceptar si lo considera aceptable.

El Ingeniero tiene libertad en lo que se refiere a pequeñas modificaciones.

El contratista será el único responsable de su personal en lo referente a la seguridad e higiene en el trabajo y en lo referente a accidentes e inexperiencia de sus trabajadores. Deberá tener lo oportuno en la legislación vigente al respecto en cuanto a accidentes de su personal y en su caso del nuestro, para facilitar los primeros auxilios.

1.3.- Impuestos.

Todos los impuestos que se devenguen con motivo y ocasión del presente trabajo proyectado, serán por cuenta y cargo del peticionario, constructor o propietario, según figure en sus mutuos contratos.

1.4. - Responsabilidades.

El cliente deberá tomar en lo que a él respecta, toda clase de precauciones a fin de evitar accidentes, en el bien entendido de que si nuestro personal se accidenta por falta manifiesta de estas precauciones, correrán a su cargo las responsabilidades a que hubiera lugar.

Nuestros técnicos no pueden adquirir compromiso alguno en nombre de la empresa, sin autorización escrita, especial para cada caso.

1.5.- Contrato de licencia de los programas.

El usuario puede:

a) hacer uso de los programas en un equipo.

b) copiar los programas en cualquier otro equipo para cualquier fin de apoyo o modificación y solo como apoyo de dicha utilización.

c) modificar los programas y /o amalgamarlo con otros programas para emplearlos en un equipo determinado.

d) transferir los programas y la licencia a terceras partes si notifica al programador los datos de la tercera parte y dicha tercera parte acepta los términos y condiciones de este contrato

y pagar lo estipulado a la sazón en concepto de transferencia.

Si el usuario transfiere los programas, deberá efectuar la transferencia simultánea de todas las copias, incluido los originales, o bien destruir todas las copias que no transfiera; esto comprende todas las modificaciones de los programas y porciones de los mismos que estuviesen contenidas o amalgamadas en otros programas.

1.6.- Competencias jurisdiccionales.

Las competencias jurisdiccionales, en caso de litigio, correrán a cargo de los tribunales competentes de Bilbao, y todos los gastos que se originen en caso de pleito, como los derechos reales que habrá que satisfacer por la formalización de este documento serán por cuenta del cliente.

2.- CONDICIONES TÉCNICAS.

2.1. - Recepción de materiales.

Los materiales empleados en el proyecto son de primera calidad.

2.2.- Puesta en marcha y funcionamiento.

Se considera que se puede llevar a cabo la puesta en marcha cuando se haya pasado la revisión del sistema, con sus correspondientes comprobaciones, para asegurarnos que no hay fallos y verificar su buen funcionamiento.

Todo el material usado es sometido a ensayos en las peores condiciones por el fabricante, y la instalación completa es probada antes de su conexión definitiva. Esto se realiza en presencia del usuario o de un ingeniero designado por ellos.

2.3.- Condiciones de montaje.

Habrá un ordenador PC, que tendrá como mínimo la siguiente configuración:

- Monitor en color.
- Disquetera de 3,5".
- Memoria RAM de 16Mb.
- Disco duro de lGb.
- Puerto serie COMl .
- Puerto paralelo.
- Slots de expansión.
- Sistema operativo Windows 98
- Sistema operativo MSDOS.
- Programa LabVIEW. •

El ordenador se instalará sobre una mesa de fácil acceso y habilitada para tal fin, lejos de cualquier sistema que genere campos electromagnéticos, o cualquier otro tipo de interferencia. La ubicación se llevará a cabo dentro de una sala libre de todo tipo de suciedad y protegida de la humedad, en la cual no haya cambios bruscos de temperatura. Si es necesario se instalará un sistema de aire acondicionado.

2.4. - **Condiciones de uso y mantenimiento.**

Con respecto al PC, debido al carácter del proyecto, la manipulación la podrá realizar cualquier persona interesada en él. La empresa instaladora certifica que los programas funcionan perfectamente y da al usuario total libertad de modificarlos a su gusto. En caso de avería la empresa instaladora se encargará de su reparación solo en el caso que se deba a algún fallo de las diferentes tarjetas del sistema.

La placa del DSP TMS320C50 se deberá guardar en su bolsa de protección siempre que no sé este utilizando. El transformador de alimentación del DSP deberá conectarse a la placa un tiempo antes de empezar a trabajar con ella, así como el cable RS-232.

En el caso de utilizar el PCI-6024E se deberá conectar también al módulo del ordenador un tiempo antes de empezar a trabajar, así como los cables de entrada y salida de las señales (RCA IN y RCA OUT).

3.- CONDICIONES ECONÓMICAS.

3.1. - Garantías.

La empresa instaladora garantiza el material de instalación durante un año a partir de la finalización de la instalación, contra todo funcionamiento erróneo de cualquier dispositivo ó desperfecto por mal funcionamiento, obligando esta garantía a reparar ó sustituir a costa de nosotros, en el plazo más corto posible, toda pieza reconocida como defectuosa, sin indemnización alguna de las dos partes.

Con respecto a los programas, el programador garantiza el medio material en que se distribuye el programa, declarando que los materiales y manufactura dedicados a un uso normal estarán libres de defectos durante los siguientes noventa (90) días siguientes a la fecha de instalación.

También se garantiza que los programas adjuntados funcionan correctamente siempre y cuando se ejecuten bajo las condiciones de trabajo.

3.2. - **Condiciones de pago.**

El pago se realizará mediante transferencias bancarias. Salvo pacto en contrato, las condiciones para la forma de ejecutar el pago son las siguientes:

- En caso de aceptación, el primer pago será del 20% del total cuando se acuerde por ambas partes la realización del proyecto.

- El segundo pago será del 30% del total, y se realizará con la entrega de los materiales y el comienzo de la instalación.

- El tercer pago será del 50% del total, y se realizará a los 90 días de la finalización de la instalación.

Por cada plazo se extenderá un recibo con sus cláusulas respectivas para utilizar en caso de incumplimiento por litigio. En caso de retraso en alguno de los plazos, se parará si este retraso llega a ser de 15 días y llevará consigo la obligación de abonar los intereses de demora, de acuerdo con el tipo bancario en curso.

3.3. - Plazos de **montaje.**

Serán impuestos por la fecha de inicio de las obras, según intereses de la compañía instaladora y del usuario, y en el plazo de 3 días estará completamente acabada y probada.

La empresa instaladora se compromete a cumplir este plazo, aunque no garantiza ni acepta responsabilidad alguna en demoras que pudieran ocasionarse en almacenes de suministros, obtención de permisos, retrasos en el transporte.

El receptor del proyecto, no tendrá derecho a anular su pedido sin la previa conformidad escrita de la empresa instaladora.

5 PRESUPUESTO.

1.- UNIDADES O MEDICIONES.

1.1. - Ordenador.

Referencia	Denominación	Tipo	Marca	Cantidad
1	Ordenador	PC		1
2	Monitor	14"	AOC	1
3	Windows		Microsoft	1
4	Ms-dos		Microsoft	1
5	Dsk		Texas Instuments	1
6	Labview	5.1	National Instuments	1

1.2. - Sistema de adquisición de datos.

Referencia	Denominación	Tipo	Marca	Cantidad
11	Tarjeta	PCI-6024E	National Instruments	1
12	Cable Cinta 65 pines	R6868 1 metro	National Instruments	1
13	Tarjeta Conexión I/O	CB-68LP	National Instruments	1

1.3.- Tarjeta DSP.

Referencia	Denominación	Tipo	Marca	Cantidad
21	Placa	TMS320C50PQ	National Instruments	1
22	Fuente de alimentación	9V / 300 mA / 2,7VA		1
23	Cable	RS—232	Roline	1
24	Conector	RCA 1 metro		2

2.-COSTO DE LOS MATERIALES.

2.1. - Ordenador.

Referencia	Nº Materiales	Costo Unidad	Costo Total
1	1	601,01	601,01
2	1	120,2	120,2
3	1	60,l	60,1
4	1	30,05	30,05
5	1	75,13	75,13
6	1	180,3	180,3

2.2. –Sistema de adquisición de datos.

Referencia	Nº Materiales	Costo Unidad	Costo Total
11	1	450,76	450,76
12	1	7,51	7,51
13	1	48,05	48,05

2.3.-Tarjeta DSP.

Referencia	Nº Materiales	Costo Unidad	Costo Total
21	1	150,25	150,25
22	1	30,05	30,05
23	1	16,53	16,53
24	2	0,12	0,24

3.-PRESUPUESTO DE LA MANO DE OBRA.

Traba.io	Horas	Categoría	€ / hora
Búsqueda de información.	150		5
Instalación del equipo.	2		8
Elaboración de programas y pruebas.	80	Ingeniero Técnico	18
Elaboración del proyecto.	135	Ingeniero Técnico	18

4. - PRESUPUESTO TOTAL.

- Ordenador.	1066,79 €
- Sistema de adquisición de datos.	506,32 €
- Tarjeta DSP.	197,07 €
- TOTAL MATERIALES.	1770,18 €
- TOTAL MANO DE OBRA.	4636,00 €
- TOTAL EN EUROS.	6406,18 €
+ 15% de imprevistos.	960,93 €
+ 10% de beneficios.	640,62 €
Total en euros.	8007,73 €
+ 16% I.V.A.	1281,24 €
TOTAL EUROS.	**9288,97** €

El presente proyecto asciende a la cantidad de:

NUEVE MIL DOSCIENTOS OCHENTA Y OCHO EUROS CON NOVENTA Y SIETE CÉNTIMOS.

BILBAO, a 11 de enero de 2002. Los ingenieros:

6. BIBLIOGRAFÍA

- LIBROS CONSULTADOS.

  - DSP Teaching Kit. Instructor guide. 1996. Texas Instruments.
  - A simple approach to DIGITAL SIGNAL PROCESSING. 1992. Texas Instruments.
  - TMS320 DSP Development Support. Reference guide. 1997. Texas Instruments.
  - TMS320C5X DSP Starter Kit. User's Guide. 1996: Texas Instruments.
  - TMS320C5X User's Guide. 1997. Texas Instruments.
  - TMS320C5X General-Purpose Applications User's Guide. 1997. Texas Instruments.
  - DAQ 6023E/ 6024E/ 6025E. USER MANUAL. National Instruments.
  - LabVIEW Evaluation guide. 1998. National Instruments.

- PAGINAS WEB VISITADAS.

  - Texas Instruments. www.ti.com
  - National Instruments. www.ni.com
  - Búsquedas de : "Manuales'', "Ejemplos", "TMS320C5X" , "TMS320C50", "*.asm" y otros en www.Google.com , www.altavista.com , www.e-bay.com , www.wanadoo.com , www.terra.es , www.ya.es , www.yahoo.com , www.technicalsupport.com , www.alltheweb.com , www. dspguide.com y otras. '

- Búsquedas en universidades: Navarra, Santiago, Madrid, Valencia, Barcelona, Perú, México, Texas, Berkley, Londres, Standford, Londres y otras.

- Búsquedas de algún programa que traduzca lenguaje "C" a ensamblador, y de manuales, ejemplos y diversa documentación acerca de filtros y de procesado digital de señales analógicas en lenguaje "C".