PROCESAMIENTO DIGITAL DE SEÑALES: DISEÑO Y CONSTRUCCIÓN DE UNA TARJETA DE PROPÓSITO GENERAL.

ALONSO DE JESÚS RETAMOSO LLAMAS

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICOMECANICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES

MAESTRÍA EN POTENCIA ELÉCTRICA

BUCARAMANGA

2005.

PROCESAMIENTO DIGITAL DE SEÑALES: DISEÑO Y CONSTRUCCIÓN DE UNA TARJETA DE PROPÓSITO GENERAL.

ALONSO DE JESÚS RETAMOSO LLAMAS

Trabajo de Investigación
presentado para optar al título de Magíster en Potencia Eléctrica.

Director
GABRIEL ORDÓÑEZ PLATA
Ingeniero Electricista, PhD.

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICOMECANICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES

MAESTRÍA EN POTENCIA ELÉCTRICA

BUCARAMANGA

2005.

DEDICATORIA

A Dios,
A mi Esposa, mi motivación y fortaleza,
A mi Familia,
A mi Papá por su insistencia,
Gracias por estar a mi lado.

AGRADECIMIENTOS

El autor expresa sus agradecimientos a:

Gabriel Ordóñez Plata, Doctor Ingeniero Industrial. Profesor Titular de la Universidad Industrial de Santander. Por sus invaluables aportes como Director de este trabajo de investigación, constante motivación, entrega profesional y calidad humana.

Jaime Guillermo Barrero Pérez, Magíster en Potencia Eléctrica. Director de la Maestría en Potencia Eléctrica de la Universidad Industrial de Santander. Por su colaboración y gestión para la culminación de los estudios de maestría.

A Mónica María Rojas G por su comprensión y paciencia a lo largo de estos cinco años.

A todos los compañeros de la Maestría en Potencia Eléctrica de la Universidad Industrial de Santander por sus aportes y reflexiones, así como por su constante entusiasmo y alegría en la búsqueda del conocimiento.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

TITULO:

PROCESAMIENTO DIGITAL DE SEÑALES: DISEÑO Y CONSTRUCCION DE UNA TARJETA DE PROPOSITO GENERAL.[*]

AUTOR:

ALONSO DE JESÚS RETAMOSO LLAMAS[**]

PALABRAS CLAVE: **Procesamiento digital de señales, DSP, FFT en base 2, FFT en base 4, FFT en base mixta, Goertzel, CHIRP, transformada discreta de *Fourier* (DFT).**

DESCRIPCIÓN:

En este documento se evalúa el rendimiento computacional del ADSP2181 de la familia ADSP21xx. Específicamente, se estudia su funcionamiento con la implementación de algoritmos de Transformada Rápida de Fourier en base 2, 4 y mixta, Goertzel y CHIRP.

En el capítulo 1 se presentan los desarrollos matemáticos de los algoritmos FFT en base 2, base 4, base mixta, Goertzel y CHIRP.

En el Capítulo 2, se analiza: el núcleo central del procesador de señales DSP específicamente, los componentes básicos como los apuntadores de direcciones de memoria de datos DAGs, unidad para el manejo de ciclos repetitivos, unidad de multiplicación y acumulación MAC, unidad aritmético-lógica ALU, Unidad de corrimientos SHIFTER y los periféricos mas importantes que componen el DSP.

En el Capítulo 3 se estudia la construcción, diseño y software que maneja la tarjeta adaptable utilizada para ampliar el número de canales de entrada y salida analógica y el número de canales de entrada y salida digital de la tarjeta de evaluación EZKIT2181.

En el capítulo 4 se implementan en software los algoritmos mencionados en el capítulo 1 y se realiza la evaluación computacional del DSP y se comparan el lenguaje Ensamblador con el lenguaje C. Esta última comparación se realiza con base en el tamaño que ocupa el programa generado por los dos lenguajes en la memoria de programa del DSP.

En el Capítulo 5 se diseñan tres guías de laboratorio con los siguientes temas: ensayos preliminares de la tarjeta adaptable; filtros FIR e IIR; algoritmos para el cálculo de la transformada discreta de Fourier.

En el Capítulo 6 se presentan las principales conclusiones y se proponen trabajos futuros.

---

[*] Trabajo de Investigación

[**] UIS, Maestría en Ingeniería, Gabriel Ordóñez Plata

SUMMARY

TITLE:

DIGITAL SIGNAL PROCESSING: DESIGN AND CONSTRUCTION OF GENERAL PURPOSE BOARD*

AUTHOR:

ALONSO DE JESÚS RETAMOSO LLAMAS**

KEYWORDS: **Digital signal processing, FFT radix 2, FFT radix 4, FFT mixed radix, Goertzel, CHIRP, Discrete Fourier Transform (DFT).**

DESCRIPTION:

In this document the computational performance of the ADSP2181 of the family ADSP21xx is evaluated. Specifically, its operation with the implementation of algorithms of Fast Fourier Transform radix 2, radix 4, mixed radix, Goertzel and CHIRP is studied.

In the chapter 1 the mathematical developments of the algorithms FFT radix 2, radix 4, mixed radix, Goertzel and CHIRP are presented.

In the chapter 2 the central core of the Digital Signal Processor DSP is analyzed, specifically the basic components like the Data Address Generator DAGs, unit for the handling of repetitive cycles, Multiplication and accumulation unit MAC, Arithmetical-Logical unit ALU, logical and arithmetic displacement unit SHIFTER and the most important peripherals that compose the DSP.

In the chapter 3 the construction, design and software that handle the adaptive board used to increase the number of analogical input and output channels and the number of digital input and output channels of the evaluation board EZKIT2181 are studied.

In the chapter 4 the algorithms mentioned in the chapter 1 are implemented in software and the computational evaluation of the DSP is carried out and the Assembler language with the language C are compared. This last comparison is carried out based on the size that the program generated by the two languages occupies in the program memory of the DSP.

In the Chapter 5 three laboratory guides with the following topics are designed: preliminary test of the adaptive board; FIR and IIR filters; and algorithms for the calculation of the discreet Fourier transform.

In the Chapter 6 the main conclusions and future developments are presented.

---

* Researching work.

** UIS, Master of engineering, Gabriel Ordóñez Plata.

INTRODUCCIÓN

Actualmente, el procesamiento digital de señales tiene cada vez más importancia, debido al auge de los procesadores de propósito específico como los procesadores digitales de señales DSP, por lo cual se hace necesario evaluar y entender cada una de las características esenciales en el manejo de esta nueva herramienta.

El procesamiento digital de señales comprende muchos aspectos entre los cuales se encuentran dos muy importantes: calculo completo de la Transformada discreta de Fourier DTF mediante algoritmos que aprovechen las propiedades de dicha transformada y el cálculo parcial del espectro de una señal. En el grupo de algoritmos que calculan todo el espectro de una señal se encuentran la transformada rápida de Fourier, FFT sigla en ingles, implementados a raíz del auge que tuvo a finales de la década de los años 50 el tratamiento digital de señales con el advenimiento de las primeras computadoras con capacidad real de procesamiento. En el cálculo parcial del espectro en frecuencia de una señal discreta se destacan los algoritmos de Goertzel y CHIRP, los cuales tienen como base el filtrado lineal.

Los aspectos más importantes que se deben tener en cuenta en el desarrollo de estos algoritmos al ser implementados en un procesador digital de señales son: el desbordamiento aritmético, el ancho del *bus* de datos del procesador, el ruido externo en la adquisición de la señal, la frecuencia de muestreo y la capacidad de almacenamiento de programa y de datos. El desbordamiento aritmético puede causar en el peor de los casos que el espectro calculado no sea el espectro real de la señal; este aspecto de la implementación de los algoritmos esta muy ligado con el ancho del *bus* de datos del procesador y con el tipo de aritmética que maneja el mismo si es de coma fija o de coma flotante. El ancho del *bus* de datos del procesador determina la precisión máxima con la cual se pueden realizar los cálculos en el procesador. El ruido externo generado en el interior del canal de adquisición de señales altera en gran medida el espectro y proviene de las siguientes fuentes: la inducción magnética provocada por la red de 60 Hz; el aumento de la temperatura lo cual se traduce directamente en un aumento de los niveles de *offset* de los amplificadores operacionales y los cambios en los valores de las resistencias y los capacitores del canal de adquisición de datos.

La frecuencia con la cual se toman muestras de la señal de entrada analógica determina el ancho de banda de las señales que se pueden adquirir sin generar el fenómeno de solapamiento. Si el anterior aspecto no se trata con cuidado, el procesador digital puede registrar o procesar señales que no corresponden a la señal analógica de entrada. La capacidad interna de almacenamiento de datos y espacio de memoria para almacenar el programa, tienen que ver con la cantidad de muestras que se adquieren de las señales de entrada y con el tipo de lenguaje de programación que se emplea para realizar el *software*.

El corazón de los sistemas que se encargan de realizar el procesamiento de señales discretas es el DSP; en este caso, se utilizará un procesador de 16 bits, 2 grupos de apuntadores de direcciones DAGs cada uno de ellos con 4 registros punteros, 1 grupo de manejo de ciclos repetitivos, memoria de datos de 16 *Kbytes*, memoria de programa de 16 *Kbytes* y *buses* de direcciones de datos y direcciones de programa totalmente independientes. Este procesador esta instalado en una tarjeta de evaluación llamada EZKIT2181, la cual puede comunicarse con el computador a través del puerto serial. La programación, emulación y simulación del DSP se realiza por intermedio del software VISUALDSP++, herramienta que permite programar el DSP en lenguaje Ensamblador, lenguaje C y lenguaje C++.

La tarjeta EZKIT2181 dispone solamente de un canal de entrada y salida de señales analógicas construidos con un CODEC. Esta falencia en cuanto a número de entradas y salidas analógicas se puede cubrir externamente ya que la tarjeta tiene conectores que permiten la interfase con dispositivos externos; en el caso de este proyecto se diseñará y construirá una tarjeta que expande las posibilidades de la tarjeta EZKIT2181 brindándole la posibilidad de recibir información desde 8 canales analógicos de entrada, enviar información hacia 8 canales de salida digital y recibir o enviar entradas digitales de 0 a 5 volts a través de un puerto de entrada y salida de 8 *bits* de longitud.

Para realizar una evaluación de rendimiento computacional en un procesador de señales se deben tener en cuenta aspectos como la cantidad de espacio de memoria, el número de ciclos y la cantidad de recursos internos que emplea el mismo en la ejecución de un programa. En el presente trabajo de investigación se evaluará no solo el procesador ADSP2181 como tal sino también el software VISUALDSP++ en cuanto a la conversión de código de lenguaje C a lenguaje de máquina entendible por el procesador. La evaluación de rendimiento computacional se establecerá mediante la

implementación de los algoritmos de la FFT en base 2, FFT en base 4, FFT en base mixta, Goertzel y CHIRP, en lenguaje ensamblador y lenguaje C. Además, debido a la escasa experiencia en la universidad con respecto al algoritmo FFT en base mixta, se pondrá un énfasis especial en ello con la mira puesta en establecer hasta que punto es conveniente o no utilizar este algoritmo a favor o en contra de los otros algoritmos de la FFT en base 4 o en base 2. Además se elaborará una guía de prácticas de laboratorio que permitirá a los futuros estudiantes tener un buen acercamiento al funcionamiento del ADSP2181, de la tarjeta adaptable y de los algoritmos previamente mencionados.

La organización de este documento se describe a continuación. En el primer capítulo, se presentan los aspectos generales de los algoritmos FFT en base 2, FFT en base 4, FFT en base mixta, Goertzel y CHIRP.

El funcionamiento interno del núcleo central del procesador DSP de la familia ADS21xx, se analiza en el capítulo 2.

Lo relacionado con el diseño, la construcción y el *software* implementado para el funcionamiento de la tarjeta adaptable del DSP se presenta en el capítulo 3.

En el Capítulo 4 se explica la implementación, en el *software* del DSP, de los algoritmos analizados previamente en el capítulo 1.

El Capítulo 5 presenta el diseño de tres guías de laboratorio: la primera acerca del manejo básico de la tarjeta adaptable, la segunda se elaboró con base en los filtros FIR e IIR y la tercera se utiliza para establecer comparaciones de rendimiento computacional de los algoritmos vistos en los capítulos 1 y 4.

Finalmente las Principales conclusiones de este trabajo y futuros desarrollos que se pueden derivar del mismo se presentan en el capítulo 6.

1. TRANSFORMADA DE FOURIER DE SEÑALES DISCRETAS

La Transformada Discreta de Fourier (DFT), es un algoritmo de gran utilidad para el análisis y tratamiento de señales discretas.

Este capítulo consta de dos partes, las cuales tratan acerca de la obtención del espectro de una señal discreta utilizando la DFT ya sea en su totalidad o en forma parcial. En la primera parte se analizan los algoritmos rápidos y directos y en la segunda se tratan los algoritmos de Goertzel y Z-chirp.

Si el lector desea obtener mayor información acerca de lo que se trata en este capítulo la puede obtener en las referencias: [Proakis & Manolakis, 1997] y [Oppenheim et al, 2000].

1.1. TRANSFORMADA DISCRETA DE FOURIER EN FORMA DIRECTA.

La DFT de una señal discreta de longitud finita L se obtiene del muestreo equiespaciado del espectro continuo de la señal. A continuación se observa este desarrollo, donde la ecuación (3) es la DFT de una señal discreta de longitud L y la ecuación (1) es la misma transformada de Fourier de la señal.

$$X(e^{j\omega}) = \sum_{n=0}^{L-1} x[n]e^{-j\omega n} \tag{1}$$

$$X[k] \equiv X[2\pi k / N] = \sum_{n=0}^{L-1} x[n]e^{-j2\pi kn/N} \tag{2}$$

$$X[k] = \sum_{n=0}^{L-1} x[n]e^{-j2\pi kn/N} \tag{3}$$

1.2. CÁLCULO EFICIENTE DEL ESPECTRO DE UNA SEÑAL DISCRETA

El principal problema de la implementación de la DFT es la gran cantidad de operaciones matemáticas que se deben realizar. Si se tiene en cuenta que tanto la señal como su espectro son complejos, se puede establecer de la ecuación (3) que se requieren N multiplicaciones complejas (4N multiplicaciones reales) y N -1 sumas

complejas (4N-2 sumas reales) para el cálculo de un solo valor de la ecuación (3). Al realizar la proyección para la totalidad del espectro se necesitan N^2 multiplicaciones complejas y $N^2 - N$ sumas complejas. Desde el punto de vista computacional en las primeras épocas de los procesadores era impensable realizar el cálculo directo, por lo tanto se dio inicio a una serie de investigaciones y desarrollos que llevaron a la creación de algoritmos que redujeran la cantidad de operaciones necesarias para el cálculo de la DFT en forma directa. Estos algoritmos son llamados Transformada Rápida de Fourier o por su sigla en inglés: FFT.

El cálculo directo de la DFT al no aprovechar las propiedades de simetría (5) y periodicidad (6) del factor $e^{-j2\pi/N}$ (4) se torna altamente ineficiente. Estas dos propiedades son las que permiten implementar los algoritmos de FFT y son:

$$W_N = e^{-j2\Pi/N} \tag{4}$$

$$W_N^{k+N/2} = -W_N^k \tag{5}$$

$$W_N^{k+N} = W_N^k \tag{6}$$

1.2.1. FFT en base 2.

La FFT en base 2 tiene como requisito que el número de muestras N de la señal cuyo espectro se va a determinar debe ser un múltiplo de 2. Bajo esta premisa $N = 2^r$.

El primer paso para el cálculo de la FFT en base 2 es dividir la señal original en dos partes: par e impar (f_1 y f_2 respectivamente).

$$\begin{aligned} f_1[n] &= x[2n] \\ f_2[n] &= x[2n+1] \\ n &= 0,1,\ldots,\frac{N}{2}-1 \end{aligned} \tag{7}$$

Con esta división la DFT de N puntos (3) se puede expresar en función de las señales f_1 y f_2 de la siguiente forma:

$$X[k] = \sum_{n=0}^{N/2-1} x[2n]W_N^{kn} + \sum_{n=0}^{N/2-1} x[2n+1]W_N^{kn} \tag{8}$$

$$W_N^2 = W_{N/2}$$

$$X[k] = \sum_{n=0}^{N/2-1} x[2n]W_{N/2}^{kn} + W_N^k \sum_{n=0}^{N/2-1} x[2n+1]W_{N/2}^{kn} \tag{9}$$

$$X[k] = F_1[k] + W_N^k F_2[k] \tag{10}$$

$$k = 0,1,...., N-1$$

En la ecuación (10) $F_1[k]$ es la DFT de las muestras pares de la señal $f_1[n]$ y $F_2[k]$ es la DFT de las muestras impares de la señal. Dado que F_1 y F_2 son periódicas con periodo N/2 y el factor de fase es simétrico se obtienen las siguientes expresiones:

$$X[k] = F_1[k] + W_N^k F_2[k] \tag{11}$$

$$X[k + N/2] = F_1[k] - W_N^k F_2[k] \tag{12}$$

$$k = 0,1,..., \frac{N}{2} - 1$$

Si se realiza un análisis de complejidad computacional hasta este punto del algoritmo se puede observar que se requieren $(N/2)^2$ multiplicaciones complejas para calcular $F_1[k]$ y otro tanto para el mismo cálculo de $F_2(k)$; además se debe tener en cuenta la multiplicación del factor de fase W_N^k que agrega otras N/2 multiplicaciones al cálculo previamente hecho. Por lo tanto para el cálculo total de X[k] se requieren $2(N/2)^2 + N/2$ multiplicaciones complejas lo cual quiere decir que solamente en el primer paso del paso del algoritmo se pasa de N^2 multiplicaciones complejas a $N^2/2 + N/2$.

La segunda parte del algoritmo consiste en repetir el proceso para cada una de las secuencias que se obtienen en el final de la primera etapa. Considerando que el resultado de la primera etapa se almacena en un vector, se puede asimilar que $F_1[k]$ equivale a las muestras $f_1[n]$ y $F_2[k]$ equivale a $f_2[n]$ donde f_1 y f_2 son las entradas de datos para la segunda etapa del algoritmo. Por lo tanto ya no se tiene una sola señal de entrada a la siguiente etapa se tienen dos y en cada una de ellas se debe aplicar el mismo procedimiento realizado en la primera etapa. Lo anterior da como resultado una salida de la segunda etapa con 4 señales divididas así: v_{11} y v_{12} para f_1 y v_{21} y v_{22} para la señal f_2.

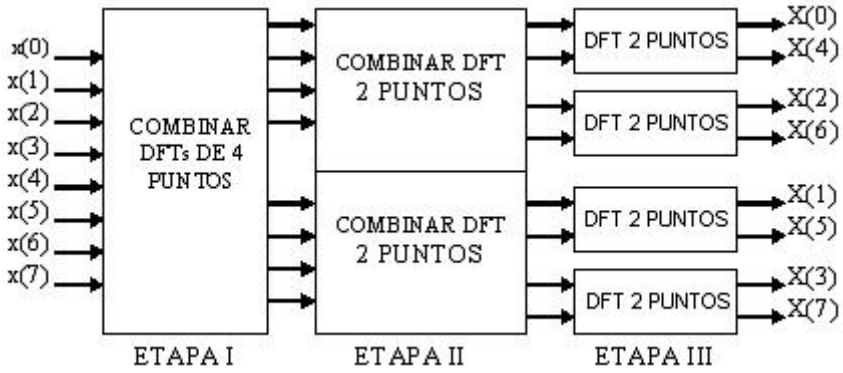
$$\begin{aligned} v_{11}[n] &= f_1[2n] \\ v_{12}[n] &= f_1[2n+1] \\ v_{21}[n] &= f_2[2n] \\ v_{22}[n] &= f_2[2n+1] \\ n &= 0,1,...., \frac{N}{4} - 1 \end{aligned} \tag{13}$$

Al calcular la DFT para las secuencias anteriores se obtienen las siguientes expresiones:

$$
\begin{aligned}
&F_1[k] = V_{11}[k] + W_N^k V_{12}[k] \\
&F_1[k+N/4] = V_{11}[k] - W_N^k V_{12}[k] \\
&F_1[k] = V_{21}[k] + W_N^k V_{22}[k] \\
&F_1[k+N/4] = V_{11}[k] - W_N^k V_{12}[k] \\
&k = 0,1,\ldots,\frac{N}{4}-1
\end{aligned}
\qquad (14)
$$

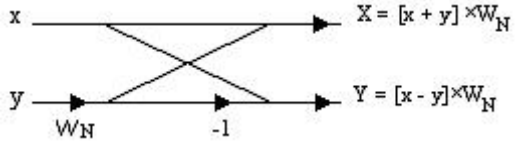
La tercera y las etapas sucesivas siguen repitiendo este procedimiento hasta que el número de etapas sea igual al valor de r, la potencia de 2 que hace la longitud de la muestra N un múltiplo de 2. En la figura 1 se muestra un diagrama de bloques de una FFT en base 2 para una señal con un número de muestras igual a 8.

FIGURA 1. Etapas de cálculo para una FFT en base 2 con N=8.



Fuente: "*Tratamiento digital de señales*". Tercera edición, Proakis, J. G. y Manolakis, D. G.

FIGURA 2. Mariposa básica para el procesamiento de la FFT en base 2.



Fuente: "*Tratamiento digital de señales*". Tercera edición, Proakis, J. G. y Manolakis, D. G.

En las ecuaciones (11) y (12) se puede observar que los cálculos básicos para realizar la FFT son 3: un producto de $F_2[k]$ por el factor de fase $(W_N)^r$ y la resta y la suma de este resultado con $F_1[k]$. Estas tres operaciones se representan en el flujograma, de la figura 2. La forma de este flujograma es similar a una mariposa y corresponde a la mariposa básica requerida para realizar el cómputo de dos muestras.

Otro aspecto importante del cómputo de la FFT en base 2 se aprecia en la figura 1. El resultado final, es decir los puntos del espectro, están en desorden. Para reordenar el vector de salida basta con establecer una sola trayectoria que va desde una entrada a

una salida; por ejemplo, la que une x[3] con X[6]. Para resolver este problema se deben convertir en binario los índices de los vectores y comparar para obtener la solución en orden, ver figura 3.

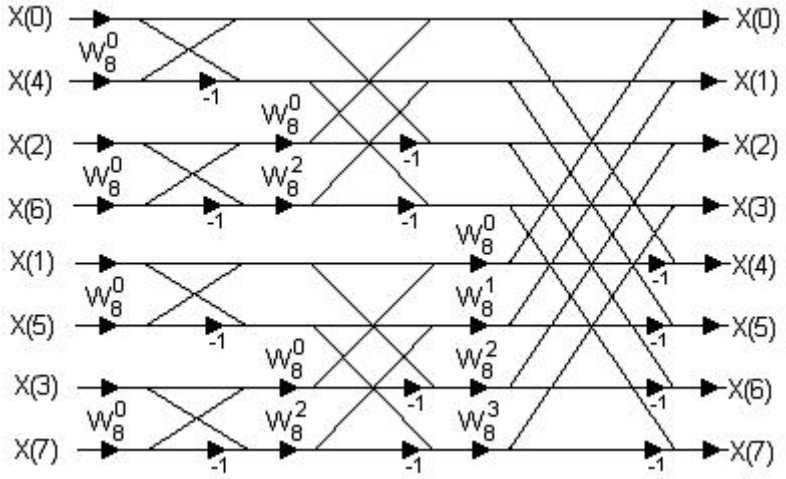
FIGURA 3. Problema de la inversión de bits en la FFT de base 2.

Número	B_0	B_1	B_2
Entrada 3	1	1	0
Salida 6	0	1	1

Fuente: el autor

Para reordenar la salida basta con invertir los bits de los índices del vector de salida. Para el caso de la FFT en base 2 con N igual a 8, se toma B_2 y se intercambia con B_0. Cuando el número de bits es impar el *bit* intermedio no se modifica. Este reordenamiento de los bits de los índices del vector se puede realizar antes de procesar las muestras con lo cual el vector de salida se obtendría en el orden adecuado. La figura 4 muestra el flujograma de una FFT de 8 muestras con diezmado en tiempo y base 2.

FIGURA 4. Flujograma de una FFT en base 2 con diezmado en tiempo.



Fuente: el autor.

1.2.2. FFT en base 4.

Cuando el tamaño de la muestra sea un factor de 4^r el espectro se puede calcular utilizando el algoritmo de la FFT en base 2 pero existe un algoritmo más eficiente para este tipo de cómputo, la FFT en base 4.

El primer paso consiste en dividir la DFT (3) de entrada no en 2 DFTs sino en 4 DFTs. Es decir:

$$X[k] = \sum_{n=0}^{N/4-1} x[n]W_N^{kn} + \sum_{n=N/4}^{N/2-1} x[n]W_N^{kn} + \sum_{n=N/2}^{3N/4-1} x[n]W_N^{kn} + \sum_{n=3N/4}^{N-1} x[n]W_N^{kn} \tag{15}$$

Colocando las sumatorias en los mismos términos inicial y final se obtiene la ecuación:

$$\begin{aligned} X[k] = \sum_{n=0}^{N/4-1} x[n]W_N^{kn} + W_N^{Nk/4} \sum_{n=0}^{N/4-1} x[n+N/4]W_N^{kn} + W_N^{kN/2} \sum_{n=0}^{N/4-1} x[n+N/2]W_N^{kn} + W_N^{3kN/4} \\ + W_N^{3kN/4} \sum_{n=0}^{N/4-1} x[n+3N/4]W_N^{kn} \end{aligned} \tag{16}$$

El siguiente paso consiste en encontrar relaciones útiles para los factores de fase que multiplican a cada sumatoria de la ecuación (16). Para ello se aplican las siguientes equivalencias:

$$\begin{aligned} W_N^{kN/4} &= (-j)^k \\ W_N^{kN/2} &= (-1)^k \\ W_N^{3kN/4} &= (j)^k \end{aligned} \tag{17}$$

Al reemplazar esta equivalencia en la ecuación 16 se obtiene la ecuación:

$$X[k] = \sum_{n=0}^{N/4-1} [x[n]W_N^{kn} + (-j)^k x[n+\frac{N}{4}] + (-1)^k x[n+\frac{N}{2}] + (j)^k x[n+\frac{3N}{4}]]W_N^{kn} \tag{18}$$

La ecuación (18) no es una DFT de N/4 puntos porque el factor de fase no es N/4. Para lograr que estos factores dependan de este valor se debe subdividir la secuencia en cuatro subsecuencias de N/4 puntos, con lo cual se llega a la expresión que forma la mariposa fundamental para una FFT de base 4. Las ecuaciones que se obtienen son las siguientes:

$$X[4k] = \sum_{n=0}^{N/4-1} [x[n] + x[n+\frac{N}{4}] + x[n+\frac{N}{2}] + x[n+\frac{3N}{4}]]W_N^{0}W_{N/4}^{kn} \tag{19}$$

$$X[4k+1] = \sum_{n=0}^{N/4-1} [x[n] - jx[n+\frac{N}{4}] - x[n+\frac{N}{2}] + jx[n+\frac{3N}{4}]]W_N^{n}W_{N/4}^{kn} \tag{20}$$

$$X[4k+2] = \sum_{n=0}^{N/4-1} [x[n] - x[n+\frac{N}{4}] + x[n+\frac{N}{2}] - jx[n+\frac{3N}{4}]]W_N^{2n}W_{N/4}^{kn} \tag{21}$$

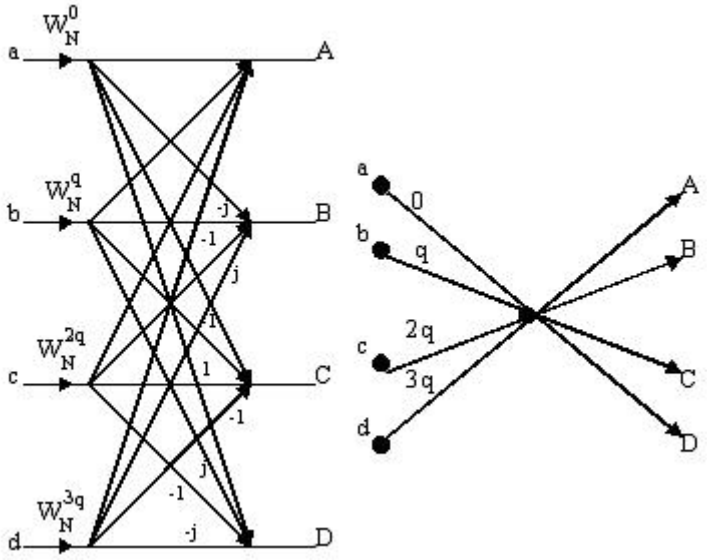
$$X[4k+3] = \sum_{n=0}^{N/4-1} [x[n] + jx[n+\frac{N}{4}] - x[n+\frac{N}{2}] - jx[n+\frac{3N}{4}]]W_N^{3n} W_{N/4}^{kn} \qquad (22)$$

En las expresiones anteriores se tiene en cuenta que $W_N^{4kn} = W_{N/4}^{kn}$ para convertir la ecuación (18) en una DFT de base 4. La etapa anterior del algoritmo se repite hasta llegar al radical que hace la muestra un múltiplo de 4 (Ej. 4^4 se repite 4 veces).

Como en el caso del algoritmo en base 2 también se puede visualizar el cómputo de la DFT si las ecuaciones (19), (20), (21) y (22) se representan en un flujograma. Cuando esto se lleva a cabo se obtiene una nueva mariposa básica, cuya diferencia con el algoritmo en base 2 es que se calculan 4 nuevos valores del vector de salida en lugar de 2. En la figura 5 se representa la mariposa fundamental del algoritmo FFT en base 4.

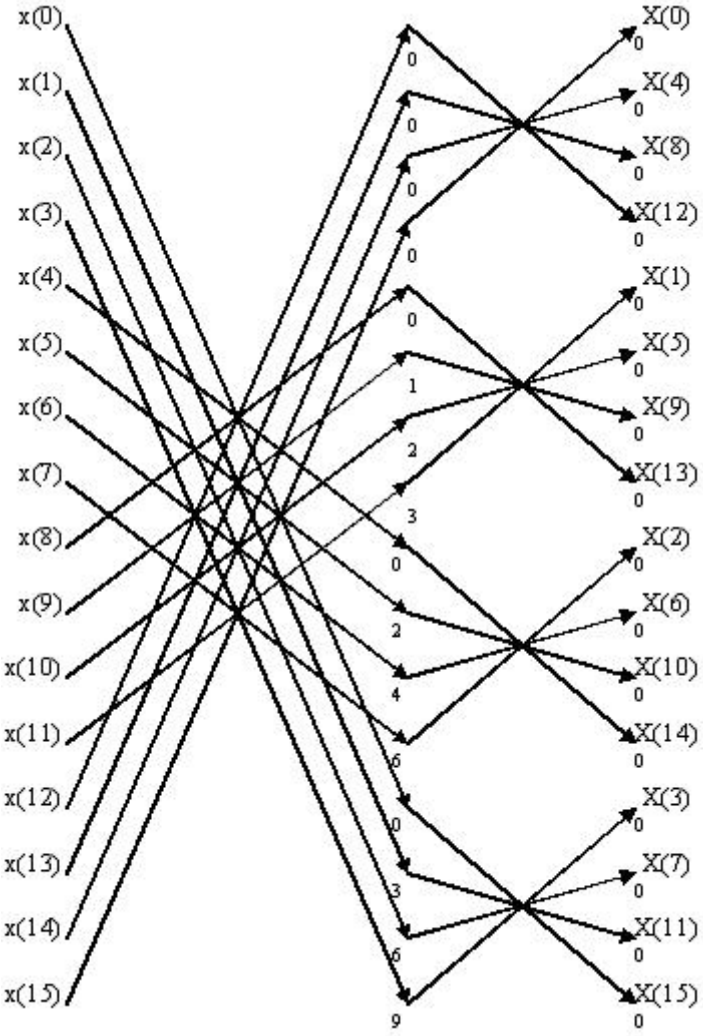
De la misma forma como se representa el algoritmo en base 2 con mariposas se puede hacer con el algoritmo FFT en base 4. La figura 6 muestra el flujograma de una FFT de 16 muestras.

FIGURA 5. Mariposa básica para el algoritmo FFT en base 4.



Fuente: "*Tratamiento digital de señales*". Tercera edición, Proakis, J. G. y Manolakis, D. G.

FIGURA 6. Flujograma del algoritmo en base 4.



Fuente: "*Tratamiento digital de señales*". Tercera edición, Proakis, J. G. y Manolakis, D. G.

El algoritmo FFT en base 4 presenta el mismo inconveniente de la inversión de bits del algoritmo FFT en base 2. La solución es similar a la planteada en el numeral 1.1 pero teniendo en cuenta no un sistema binario sino un sistema ternario, es decir, números compuestos por los enteros 0, 1, 2 y 3.

1.2.3. FFT para N compuesto.

Cuando la longitud de la señal N no es una potencia de 2 ni de 4 se debe descomponer en un producto de factores de la siguiente forma:

$$N = p_1 \times p_2 \times p_3 \ldots\ldots\ldots p_n \tag{23}$$

Como resultado de dividir el tamaño de la señal en sus factores multiplicativos se obtienen mariposas que procesan 3, 5, 6, 8, 10 muestras de una señal de entrada. De un análisis de estos factores se deduce que las mariposas de 6, 8 y 10 muestras se pueden procesar como la combinación de los factores 2×3, 2×4 y 2×5 respectivamente. Es claro que se deben obtener formas básicas para las mariposas

que procesan 3 y 5 muestras para poder ejecutar los algoritmos FFT de base partida o dividida. Ha de notarse como los factores 7, 11, 13 etc. no se utilizan en la obtención de algoritmos de la FFT, porque estos números no son divisores de 360° hecho que desfavorece en gran medida la solución rápida de la DFT. A continuación se muestran los cálculos necesarios para obtener las mariposas básicas de 3 y 5 muestras.

Para el cálculo de la mariposa de la FFT base 3 se parte de una señal cuyo tamaño sea un múltiplo de 3 en este caso N igual a 9 muestras y se realiza el mismo proceso de la FFT en base 2. El primer paso consiste en dividir la sumatoria de la DFT en tres partes como se observa en la ecuación (24).

$$X[k] = \sum_{n=0}^{8} x[n] W_9^{nk} = \sum_{n=0}^{2} x[n] W_9^{nk} + \sum_{n=3}^{5} x[n] W_9^{nK} + \sum_{n=6}^{8} x[n] W_9^{nk} \tag{24}$$

El segundo paso es agrupar en una sumatoria, lo cual conduce a la ecuación (25).

$$X[k] = \sum_{n=0}^{8} x[n] W_9^{nk} = \sum_{n=0}^{2} x[n] W_9^{nk} + x[n+3] W_9^{(n+3)k} + x[n+6] W_9^{(n+6)k} \tag{25}$$

De la ecuación (25) se puede observar que el término W_9^{nk} es común en todos los factores de la sumatoria, por lo tanto se puede factorizar con lo cual se obtiene la ecuación (26) así:

$$X[k] = \sum_{n=0}^{2} \{x[n] + x[n+3] W_9^{3k} + x[n+6] W_9^{6k}\} W_9^{nk} \tag{26}$$

En este momento se puede descomponer el espectro X[k] en tres partes como se especifica en la ecuación (27).

$$\begin{aligned} X[3r] &= \sum_{n=0}^{2} \{x[n] + x[n+3] W_9^{3(3r)} + x[n+6] W_9^{6(3r)}\} W_9^{n(3r)} \\ X[3r+1] &= \sum_{n=0}^{2} \{x[n] + x[n+3] W_9^{3(3r+1)} + x[n+6] W_9^{6(3r+1)}\} W_9^{n(3r+1)} \\ X[3r+2] &= \sum_{n=0}^{2} \{x[n] + x[n+3] W_9^{3(3r+2)} + x[n+6] W_9^{6(3r+2)}\} W_9^{n(3r+2)} \end{aligned} \tag{27}$$

Como los factores W_9^{9r} y W_9^{18r} son iguales a la unidad y el factor W_9^{3nr} es igual al factor W_3^{nr} se pueden cancelar los dos primeros y reemplazar el segundo en la ecuación (27) llegando de esta forma a la ecuación (28) así:

$$X[3r] = \sum_{n=0}^{2}\{x[n] + x[n+3] + x[n+6]\}W_3^{nr}$$

$$X[3r+1] = \sum_{n=0}^{2}[\{x[n] + x[n+3]W_9^3 + x[n+6]W_9^6\}W_9^n]W_3^{nr} \quad (28)$$

$$X[3r+2] = \sum_{n=0}^{2}[\{x[n] + x[n+3]W_9^6 + x[n+6]W_9^{12}\}W_9^{2n}]W_3^{nr}$$

Reemplazando W por su equivalente matemático se obtiene la DFT de tres puntos básica para el procesamiento de los algoritmos FFT en base 3; lo anterior queda resumido en la ecuación (29).

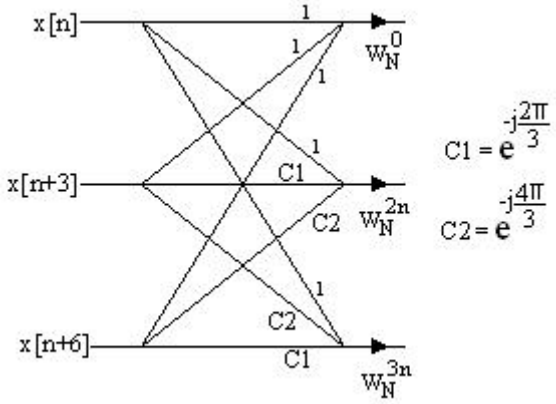
$$X[3r] = \sum_{n=0}^{2}\{x[n] + x[n+3] + x[n+6]\}W_3^{nr}$$

$$X[3r+1] = \sum_{n=0}^{2}[\{x[n] + x[n+3]e^{-j\frac{2\Pi}{3}} + x[n+6]e^{-j\frac{4\Pi}{3}}\}W_9^n]W_3^{nr} \quad (29)$$

$$X[3r+2] = \sum_{n=0}^{2}[\{x[n] + x[n+3]e^{-j\frac{4\Pi}{3}} + x[n+6]e^{-j\frac{2\Pi}{3}}\}W_9^{2n}]W_3^{nr}$$

La ecuación (29) se puede representar gráficamente en una mariposa como la mostrada en la figura 7.

FIGURA 7. Mariposa básica para una FFT en base 3.



Fuente: el autor.

Siguiendo las mismas pautas con las cuales se obtuvo la mariposa para la FFT base 3 se puede obtener la mariposa para la FFT en base 5, es decir, tamaños de señal N potencias del número 5. Como ejemplo ilustrativo se obtendrá la mariposa para una longitud de señal N igual a 25. El primer paso es dividir la señal en 5 partes lo que conduce a la obtención de la ecuación (30) así:

$$X[k] = \sum_{n=0}^{24} x[n]W_{25}^{nk} = \sum_{n=0}^{4} x[n]W_{25}^{nk} + \sum_{n=5}^{9} x[n]W_{25}^{nk} + \sum_{n=10}^{14} x[n]W_{25}^{nk} + \sum_{15}^{19} x[n]W_{25}^{nk} \quad (30)$$

La siguiente acción es utilizar una sola sumatoria; por lo tanto la expresión (30) se convierte en la ecuación (31) así:

$$X[k]=\sum_{n=0}^{24}x[n]W_{25}^{nk}=\sum_{n=0}^{4}x[n]W_{25}^{nk}+x[n+5]W_{25}^{(n+5)k}+x[n+10]W_{25}^{(n+10)k}+x[n+15]W_{25}^{(n+15)k}+x[n+20]W_{25}^{(n+20)k} \quad (31)$$

En la ecuación (31) se puede factorizar el término W_{25}^{nk} llegando a la ecuación (32) así:

$$X[k]=\sum_{n=0}^{4}\{x[n]+x[n+5]W_{9}^{5k}+x[n+6]W_{25}^{10k}+x[n+15]W_{25}^{15k}+x[n+20]W_{25}^{20k}\}W_{25}^{nk} \quad (32)$$

Con la ecuación (32) se puede dividir el espectro X[k] en 5 partes de la siguiente manera:

$$X[5r]=\sum_{n=0}^{4}\{x[n]+x[n+5]W_{25}^{5(5r)}+x[n+10]W_{25}^{10(5r)}+x[n+15]W_{25}^{15(5r)}+x[n+20]W_{25}^{20(5r)}\}W_{25}^{n(5r)}$$

$$X[5r+1]=\sum_{n=0}^{4}\{x[n]+x[n+5]W_{25}^{5(5r+1)}+x[n+10]W_{25}^{10(5r+1)}+x[n+15]W_{25}^{15(5r+1)}+x[n+20]W_{25}^{20(5r+1)}\}W_{25}^{n(5r+1)}$$

$$X[5r+2]=\sum_{n=0}^{4}\{x[n]+x[n+5]W_{25}^{5(5r+2)}+x[n+10]W_{25}^{10(5r+2)}+x[n+15]W_{25}^{15(5r+2)}+x[n+20]W_{25}^{20(5r+2)}\}W_{25}^{n(5r+2)} \quad (33)$$

$$X[5r+3]=\sum_{n=0}^{4}\{x[n]+x[n+5]W_{25}^{5(5r+3)}+x[n+10]W_{25}^{10(5r+3)}+x[n+15]W_{25}^{15(5r+3)}+x[n+20]W_{25}^{20(5r+3)}\}W_{25}^{n(5r+3)}$$

$$X[5r+4]=\sum_{n=0}^{4}\{x[n]+x[n+5]W_{25}^{5(5r+4)}+x[n+10]W_{25}^{10(5r+4)}+x[n+15]W_{25}^{15(5r+4)}+x[n+20]W_{25}^{20(5r+4)}\}W_{25}^{n(5r+4)}$$

Como las expresiones $W_{25}^{25r}, W_{25}^{50r}, W_{25}^{75r}, W_{25}^{100r}$ son iguales a la unidad, W_{25}^{5nr} es igual al factor W_{5}^{nr} y reemplazando W_{25} por su equivalente matemático $e^{-j\frac{2\Pi}{25}}$ se puede obtener la DFT de 5 puntos (34) básica para el procesamiento de señales cuya longitud es una potencia de 5. Lo anterior se resume en el flujograma de la figura 8.

$$X[5r]=\sum_{n=0}^{4}\{x[n]+x[n+5]+x[n+10]+x[n+15]+x[n+20]\}W_{5}^{5nr}$$

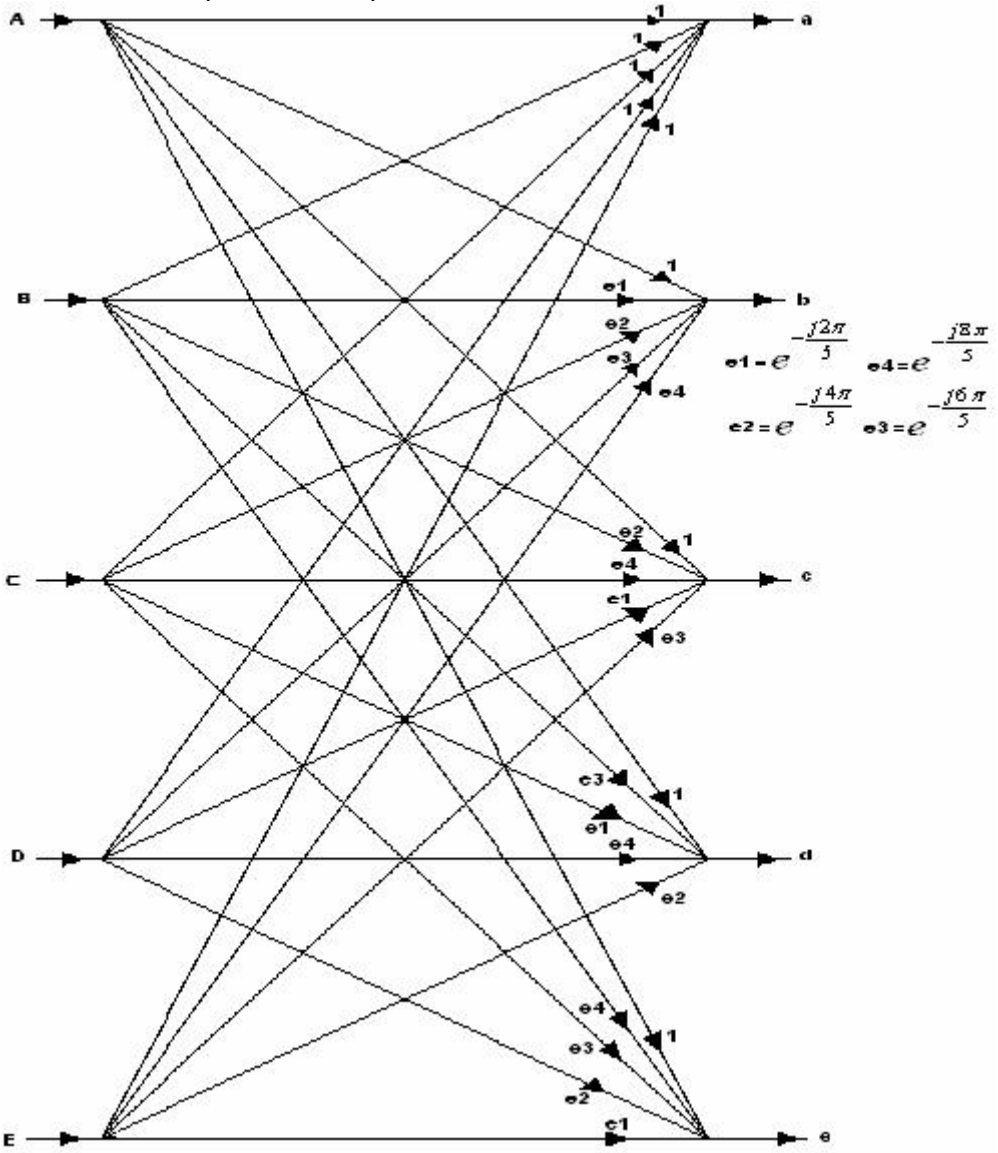
$$X[5r+1]=\sum_{n=0}^{4}[\{x[n]+x[n+5]e^{-j\frac{2\Pi}{5}}+x[n+10]e^{-j\frac{4\Pi}{5}}+x[n+15]e^{-j\frac{6\Pi}{5}}+x[n+20]e^{-j\frac{8\Pi}{5}}\}W_{25}^{n}]W_{5}^{nr}$$

$$X[5r+2]=\sum_{n=0}^{4}[\{x[n]+x[n+5]e^{-j\frac{4\Pi}{5}}+x[n+10]e^{-j\frac{8\Pi}{5}}+x[n+15]e^{-j\frac{2\Pi}{5}}+x[n+20]e^{-j\frac{6\Pi}{5}}\}W_{25}^{2n}]W_{5}^{nr} \quad (34)$$

$$X[5r+3]=\sum_{n=0}^{4}[\{x[n]+x[n+5]e^{-j\frac{6\Pi}{5}}+x[n+10]e^{-j\frac{2\Pi}{5}}+x[n+15]e^{-j\frac{8\Pi}{5}}+x[n+20]e^{-j\frac{4\Pi}{5}}\}W_{25}^{3n}]W_{5}^{nr}$$

$$X[5r+3]=\sum_{n=0}^{4}[\{x[n]+x[n+5]e^{-j\frac{8\Pi}{5}}+x[n+10]e^{-j\frac{6\Pi}{5}}+x[n+15]e^{-j\frac{4\Pi}{5}}+x[n+20]e^{-j\frac{2\Pi}{5}}\}W_{25}^{4n}]W_5^{nr}$$

FIGURA 8. Mariposa básica para una FFT en base 5.



Fuente: el autor.

Como ejemplo particular de una descomposición para N compuesto, se toma N = 18 lo cual lleva a una descomposición de la siguiente forma:

$$N = 2 \times 3 \times 3 \tag{35}$$

Siguiendo la metodología de los algoritmos de base 2 y base 4 se pueden establecer los siguientes pasos para calcular eficientemente el espectro de una señal de 18 muestras así:

El primer paso consiste en dividir la DFT de 18 muestras en dos grupos de 9 de la siguiente forma:

$$X[k]=\sum_{n=0}^{17}x[n]W_{18}^{nk}=\sum_{n=0}^{8}x[n]W_{18}^{nk}+\sum_{n=9}^{17}x[n]W_{18}^{nk} \tag{36}$$

Factorizando y agrupando los dos sumandos en una única sumatoria se obtiene la siguiente expresión:

$$X[k]=\sum_{n=0}^{17}x[n]W_{18}^{nk}=\sum_{n=0}^{8}(x[n]+x[n+9]W_{18}^{9k})W_{18}^{nk} \tag{37}$$

En este punto se dividen las muestras del espectro en dos partes así:

$$\begin{gathered}X[2r]=\sum_{n=0}^{8}(x[n]+x[n+9]W_{18}^{18r})W_{18}^{2nr}=\sum_{n=0}^{8}(x[n]+x[n+9])W_{18}^{2nr}\\ X[2r+1]=\sum_{n=0}^{8}(x[n]+x[n+9]W_{18}^{9(2r+1)})W_{18}^{(2r+1)n}=\sum_{n=0}^{8}(x[n]-x[n+9])W_{18}^{n}W_{18}^{2nr}\\ r=0,1,....,8\end{gathered} \tag{38}$$

Aplicando la igualdad: $W_N^{2rn}=W_{N/2}^{rn}$ se obtienen dos DFTs de 9 puntos como se muestra a continuación:

$$X[2r]=\sum_{n=0}^{8}(x[n]+x[n+9])W_9^{nr};X(2r+1)=\sum_{n=0}^{8}[(x[n]-x[n+9])W_{18}^{n}]W_9^{nr} \tag{39}$$

El segundo paso consiste en dividir cada una de las sumatorias de las DFTs de nueve puntos en 3 grupos así:

$$\begin{gathered}g_1[n]=x[n]+x[n+9];g_2[n]=(x[n]-x[n+9])W_{18}^{n}\\ G_1[r]=\sum_{n=0}^{8}g_1[n]W_9^{nr}=\sum_{n=0}^{2}g_1[n]W_9^{nr}+\sum_{n=3}^{5}g_1[n]W_9^{nr}+\sum_{n=6}^{8}g_1[n]W_9^{nr}\\ G_2[r]=\sum_{n=0}^{8}g_2[n]W_9^{nr}=\sum_{n=0}^{2}g_2[n]W_9^{nr}+\sum_{n=3}^{5}g_2[n]W_9^{nr}+\sum_{n=6}^{8}g_2[n]W_9^{nr}\end{gathered} \tag{40}$$

Aplicando nuevamente la factorización y el agrupamiento de sumatorias se obtienen las siguientes expresiones:

$$\begin{gathered}G_1[r]=\sum_{n=0}^{8}g_1[n]W_9^{nr}=\sum_{n=0}^{2}(g_1[n]+g_1[n+3]W_9^{3r}+g_1[n+6]W_9^{6r})W_9^{nr}\\ G_2[r]=\sum_{n=0}^{8}g_2[n]W_9^{nr}=\sum_{n=0}^{2}(g_2[n]+g_2[n+3]W_9^{3r}+g_2[n+6]W_9^{6r})W_9^{nr}\end{gathered} \tag{41}$$

Al obtener estas dos sumatorias se subdividen en 3 grupos quedando de la siguiente forma:

$$G_1[3p]=\sum_{n=0}^{2}(g_1[n]+g_1[n+3]W_9^{3(3p)}+g_1[n+6]W_9^{6(3p)})W_9^{n(3p)}=\sum_{n=0}^{2}(g_1[n]+g_1[n+3]+g_1[n+6])W_9^{3np}$$

$$G_1[3p+1]=\sum_{n=0}^{2}(g_1[n]+g_1[n+3]W_9^{3(3p+1)}+g_1[n+6]W_9^{6(3p+1)})W_9^{n(3p+1)}=\sum_{n=0}^{2}[(g_1[n]+g_1[n+3]e^{\frac{-j2\pi}{3}}+g_1[n+6]e^{\frac{-j4\pi}{3}})W_9^{n}]W_9^{2np}$$

$$G_1[3p+2]=\sum_{n=0}^{2}(g_1[n]+g_1[n+3]W_9^{3(3p+2)}+g_1[n+6]W_9^{6(3p+2)})W_9^{n(3p+2)}=\sum_{n=0}^{2}[(g_1[n]+g_1[n+3]e^{\frac{-j4\pi}{3}}+g_1[n+6]e^{\frac{-j2\pi}{3}})W_9^{2n}]W_9^{2np}$$

$$G_2[3p]=\sum_{n=0}^{2}(g_2[n]+g_2[n+3]W_9^{3(3p)}+g_2[n+6]W_9^{6(3p)})W_9^{n(3p)}=\sum_{n=0}^{2}(g_2[n]+g_2[n+3]+g_2[n+6])W_9^{3np} \quad (42)$$

$$G_2[3p+1]=\sum_{n=0}^{2}(g_2[n]+g_2[n+3]W_9^{3(3p+1)}+g_2[n+6]W_9^{6(3p+1)})W_9^{n(3p+1)}=\sum_{n=0}^{2}[(g_2[n]+g_2[n+3]e^{\frac{-j2\pi}{3}}+g_2[n+6]e^{\frac{-j4\pi}{3}})W_9^{n}]W_9^{2np}$$

$$G_2[3p+2]=\sum_{n=0}^{2}(g_2[n]+g_2[n+3]W_9^{3(3p+2)}+g_2[n+6]W_9^{6(3p+2)})W_9^{n(3p+2)}=\sum_{n=0}^{2}[(g_2[n]+g_2[n+3]e^{\frac{-j4\pi}{3}}+g_2[n+6]e^{\frac{-j2\pi}{3}})W_9^{2n}]W_9^{2np}$$

Al aplicar la identidad $W_9^{3np}=W_3^{np}$ se obtienen seis DFTs de 3 puntos de la siguiente forma:

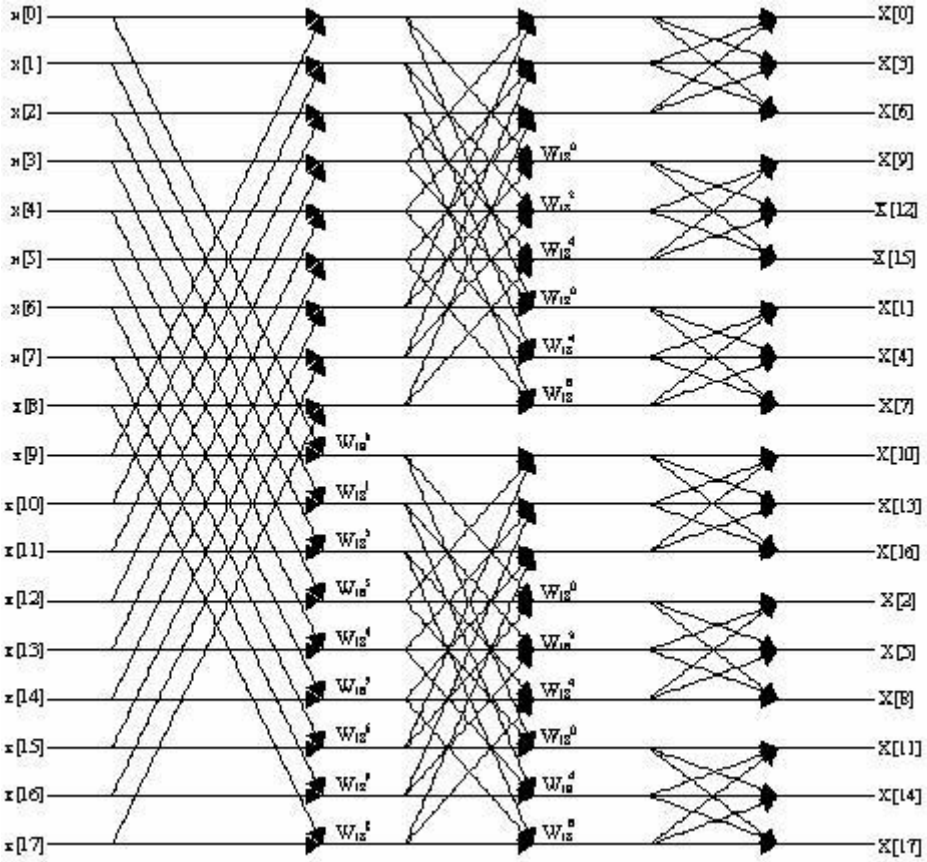
$$G_1[3p]=\sum_{n=0}^{2}(g_1[n]+g_1[n+3]+g_1[n+6])W_3^{np};G_2[3p]=\sum_{n=0}^{2}(g_2[n]+g_2[n+3]e^{\frac{-j2\pi}{3}}+g_2[n+6]^{\frac{-j4\pi}{3}})W_3^{np}$$

$$G_1[3p+1]=\sum_{n=0}^{2}(g_1[n]+g_1[n+3]e^{\frac{-j2\pi}{3}}+g_1[n+6]e^{\frac{-j4\pi}{3}})W_3^{np};G_2[3p+1]=\sum_{n=0}^{2}(g_2[n]+g_2[n+3]e^{\frac{-j2\pi}{3}}+g_2[n+6]e^{\frac{-j4\pi}{3}})W_3^{np} \quad (43)$$

$$G_1[3p+2]=\sum_{n=0}^{2}(g_1[n]+g_1[n+3]e^{\frac{-j4\pi}{3}}+g_1[n+6]e^{\frac{-j2\pi}{3}})W_3^{2np};G_2[3p+1]=\sum_{n=0}^{2}(g_2[n]+g_2[n+3]e^{\frac{-j4\pi}{3}}+g_2[n+6]e^{\frac{-j2\pi}{3}})W_3^{2np}$$

El proceso anterior se observa gráficamente en el flujograma de la figura 9.

FIGURA 9. FFT de 18 muestras con N = 2×3×3.

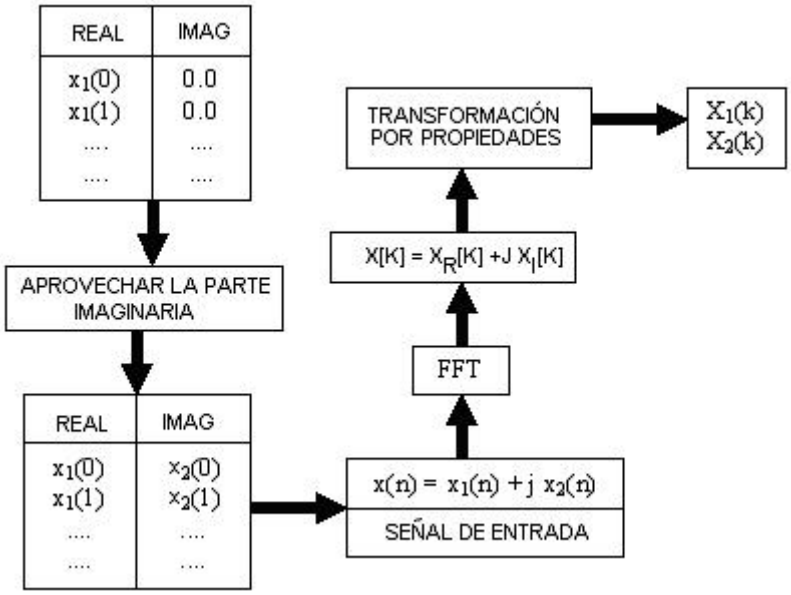


Fuente: el autor.

1.2.4. Algoritmo para el cálculo de los espectros de dos señales reales y discretas con base en el algoritmo FFT.

En este numeral se analiza un método que se utiliza para calcular en forma eficiente los espectros de dos señales reales en función de los algoritmos FFT. Como las señales reales y discretas solo ocupan la mitad de la memoria requerida por el algoritmo FFT para su realización, la parte de la memoria donde se almacena la parte imaginaria, puede ser utilizada para almacenar una segunda señal real la cual también se procesa para obtener el espectro. En la figura 10 se muestra el flujograma a seguir para realizar este cómputo.

FIGURA 10. Flujograma para obtener el espectro de dos señales discretas a partir del algoritmo FFT.



Fuente: el autor.

Los cálculos que se deben realizar para obtener los espectros de cada una de las señales de entrada se describen a continuación. Una primera consideración es conformar una señal compleja con las dos señales reales a procesar:

$$x[n] = x_1[n] + jx_2[n] \quad 0 \le n \le N\text{-}1 \tag{44}$$

Al aplicar la propiedad de linealidad de la DFT en la ecuación anterior se obtiene que la DFT de x[n] es:

$$X[k] = X_1[k] + jX_2[k] \tag{45}$$

Las señales $x_1[n]$ y $x_2[n]$ se pueden expresar en términos de x[n] de la siguiente forma:

$$x_1[n] = \frac{x[n] + x^*[n]}{2} \qquad x_2[n] = \frac{x[n] - x^*[n]}{2j} \tag{46}$$

Por lo tanto las DFTs de $x_1[n]$ y $x_2[n]$ son iguales a:

$$\begin{aligned} X_1[k] &= \frac{1}{2}\{DFT[x[n]] + DFT[x^*[n]]\} \\ X_2[k] &= \frac{1}{2j}\{DFT[x[n]] - DFT[x^*[n]]\} \end{aligned} \tag{47}$$

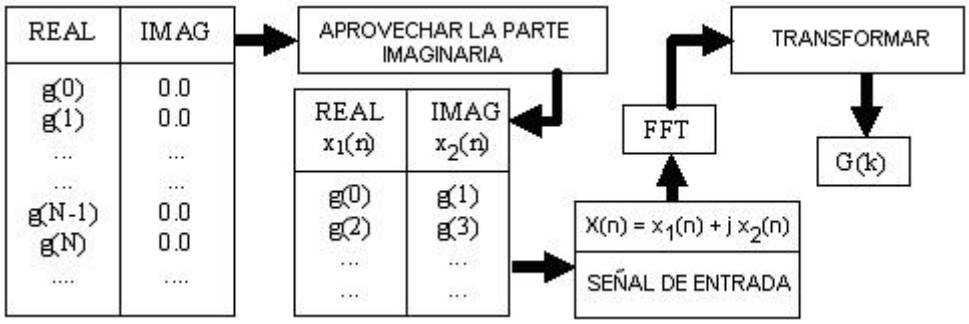
y aplicando la propiedad de la conjugada de la DFT se obtiene que:

$$\begin{aligned} X_1[k] &= \frac{1}{2}\{X[k] + X^*[N-k]\} \\ X_2[k] &= \frac{1}{2j}\{X[k] - X^*[N-k]\} \end{aligned} \tag{48}$$

1.2.5. Cálculo del espectro de una señal real de longitud 2N con una FFT de N muestras.

Como en el numeral anterior, se puede aprovechar el hecho de que la gran mayoría de señales discretas son reales y almacenar la mitad de las muestras de la señal en la memoria asignada para la parte imaginaria. Posteriormente, mediante transformaciones numéricas se obtiene el espectro de la señal original de longitud 2N, realizando una DFT de longitud N. En la figura 11 se muestra un flujograma de la realización de este cómputo.

FIGURA 11. Flujograma para algoritmo que procesa una señal de longitud 2N.



Fuente: el autor.

Para realizar este cómputo las muestras pares de la señal de entrada g[n] se almacenan en el vector $x_1[n]$ y las impares en el vector $x_2[n]$ es decir:

$$\begin{aligned} x_1[n] &= g[2n] \\ x_2[n] &= g[2n+1] \\ n &= 0...N-1 \end{aligned} \tag{49}$$

Aplicando las deducciones hechas en el numeral anterior se llega nuevamente a las ecuaciones (42) y (43). A partir de estas se puede obtener el espectro de la señal original g[n] como:

$$G[k] = \sum_{n=0}^{N-1} g[2n] W_{2N}^{2nk} + \sum_{n=0}^{N-1} g[2n+1] W_{2N}^{(2n+1)k} = \sum_{n=0}^{N-1} x_1[n] W_N^{nk} + W_{2N}^{k} \sum_{n=0}^{N-1} x_2[n] W_N^{nk} \tag{50}$$

Por lo tanto:

$$\begin{aligned} G[k] &= X_1[k] + W_{2N}^{k} X_2[k] \\ G[k+N] &= X_1[k] - W_{2N}^{k} X_2[k] \\ k &= 0,1,\ldots,N-1 \end{aligned} \tag{51}$$

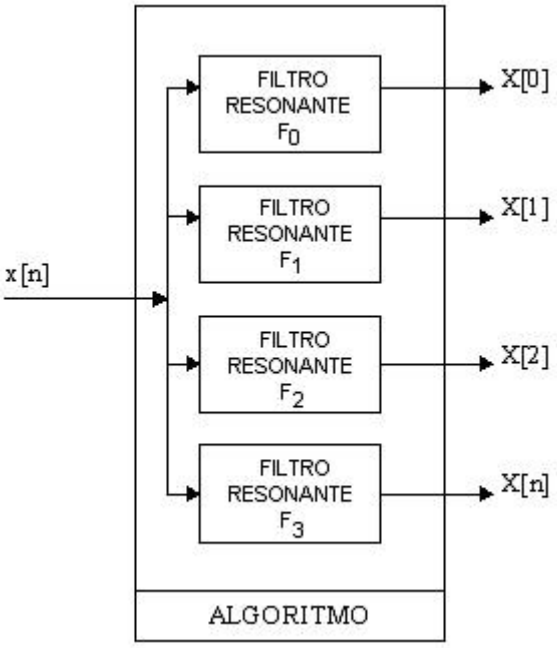
1.3. CÁLCULO PARCIAL DEL ESPECTRO DE UNA SEÑAL DISCRETA

En muchas ocasiones no es necesario calcular la totalidad de las muestras del espectro sino unas cuantas. Para ello se utilizan los algoritmos de Goertzel y Z-chirp. Lo anterior no desconoce que estos algoritmos puedan calcular todo el espectro, el inconveniente está en que no son eficientes como la FFT. Cuando no se requiere la totalidad del espectro, estos algoritmos superan a los algoritmos FFT. Uno de los procedimientos antes mencionado (Goertzel) trata la DFT como una operación de filtrado lineal, es decir, cada una de las componentes del espectro se calcula con base en un filtro resonante a cada una de las frecuencias particulares de la DFT $\omega_k = 2\pi k/N$, k = 0,1,...,N-1. El otro algoritmo (Chirp) permite obtener una parte específica del espectro con la posibilidad de elegir la frecuencia inicial, la resolución en frecuencia y por consiguiente el número de puntos que se desean evaluar del espectro.

1.3.1. Algoritmo de Goertzel.

El esquema del algoritmo de Goertzel se presenta en el flujograma de la figura 12.

FIGURA 12. Flujograma del algoritmo de Goertzel.



Fuente: el autor.

El algoritmo de Goertzel realiza la DFT como una operación de filtrado lineal. Dado que $W_N^{-kN} = 1$, se puede multiplicar la DFT sin alterarla quedando de la siguiente forma:

$$X[k] = W_N^{-kn} \sum_{m=0}^{N-1} x[m] W_N^{km} = \sum_{m=0}^{N-1} x[m] W_N^{-k(N-m)} \qquad (52)$$

La ecuación (44) se puede comparar con la ecuación de la convolución (45) así:

$$y[n] = \sum_{k=0}^{M-1} h[k]x[n-k] \tag{53}$$

La función h[k] o ecuación en diferencias del filtro lineal se reemplaza por el factor de fase de la siguiente forma:

$$h_k[n] = W_N^{-kn} u[n] \tag{54}$$

Por lo tanto la salida de este filtro cuando n es igual N da el valor de la DFT a la frecuencia $\omega_k = 2\pi k/N$. La función de transferencia de este filtro se extrae calculando la transformada Z de $h_k[n]$ así:

$$H_k[z] = Z\{W_N^{-kn}\} = \frac{1}{1 - W_N^{-k} Z^{-1}} \tag{55}$$

La ecuación (55) es al algoritmo de Goertzel como es la mariposa para los algoritmos FFT. Para convertir esta expresión en un bloque de operaciones matemáticas realizables por un procesador, se debe calcular la ecuación en diferencias de la transformada Z dada en (55). Por lo tanto:

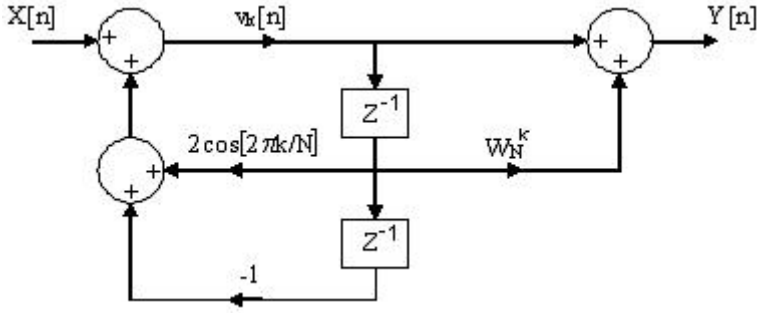
$$\begin{aligned} H_k[z] &= \frac{Y[z]}{X[z]} = Y[z] = H_k[z]X[z] \\ Z^{-1}\{X[z]\} &= Z^{-1}\{Y[z] - W_N^{-k} Z^{-1} Y[z]\} \\ y[n] &= x[n] + W_N^{-k} y[n-1] \quad y_k[-1] = 0 \end{aligned} \tag{56}$$

El valor de la componente del espectro deseada se obtiene en la salida del filtro lineal cuando se han introducido N muestras de la señal que se esta procesando. La ecuación (56) requiere que el procesador realice sumas y multiplicaciones complejas, esto se evita uniendo dos filtros de primer orden que tengan polos complejos conjugados; en cuyo caso se obtiene la siguiente función de transferencia para el filtro básico de Goertzel:

$$H_k(z) = \frac{1 - W_N^k Z^{-1}}{1 - 2\cos(2\pi k / N) Z^{-1} + Z^{-2}} \tag{57}$$

El flujograma que representa esta función de transferencia se muestra en la figura 13.

FIGURA 13. Filtro resonador con polos complejos conjugados para el algoritmo de Goertzel.



Fuente: el autor.

Para realizar las operaciones propuestas en el flujograma de la figura 13, se requiere evaluar inicialmente la función $v_k[n]$ y calcular y[n] en función de ésta y de la señal de entrada x[n]. Por lo tanto se tiene que:

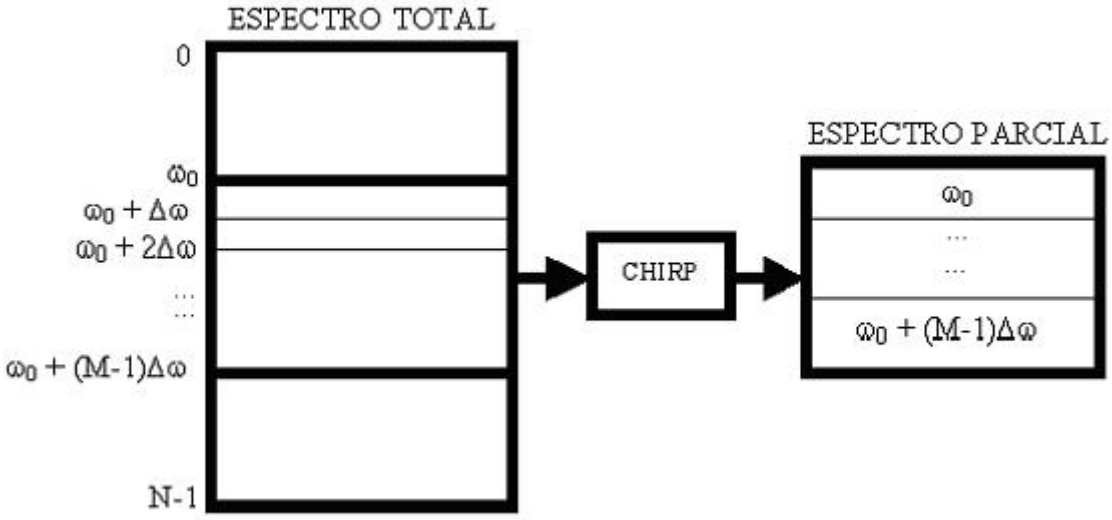
$$v_k[n] = 2\cos(\frac{2\pi k}{N})v_k[n-1] - v_k[n-2] + x[n]$$
$$y[n] = v_k[n] - W_N^k v_k[n-1] \qquad (58)$$

Donde $v_k[-1] = v_k[-2] = 0$. Cuando se aplica el filtrado de la figura 13, no sólo se obtiene la componente X[k], al mismo tiempo se evalúa la componente del espectro X[N – k].

1.3.2. Algoritmo de la transformada Chirp para obtener la DFT.

El algoritmo de la transformada Chirp permite calcular una parte del espectro de una señal de N muestras teniendo como base la convolución y no el filtrado de señales como en el caso del algoritmo de Goertzel. Este algoritmo permite seleccionar: la frecuencia de inicio, el incremento de frecuencia y el tamaño de la muestra del espectro que se desea calcular. La figura 14 muestra gráficamente las características del algoritmo Chirp.

FIGURA 14. Flujograma algoritmo CHIRP.



Fuente: el autor.

Para obtener el algoritmo CHIRP se parte de una señal discreta x[n] de N muestras y de su espectro $X(e^{j\omega})$. De este espectro se desean tomar M muestras a partir de la frecuencia ω_0 con un incremento entre una y otra muestra de $\Delta\omega$. Lo anterior conduce a la siguiente expresión para la frecuencia ω del espectro $X(e^{j\omega})$:

$$\omega_k = \omega_0 + k\Delta\omega \quad \text{k=0,1, ... ,M-1} \tag{59}$$

Al reemplazar esta expresión en la DFT se obtiene lo siguiente:

$$X(e^{j\omega_k}) = \sum_{n=0}^{N-1} x[n]e^{-j\omega_k} \qquad \text{k=0,1,...,M-1} \tag{60}$$

Utilizando la ecuación (59) y reemplazando en la ecuación (60) se obtiene la ecuación (61) así:

$$X(e^{j\omega_k}) = \sum_{n=0}^{N-1} x[n]e^{-j\omega_0 n}W^{nk} \tag{61}$$

Donde W^{nk} es igual a $e^{-j\Delta\omega}$. Para obtener $X(e^{j\omega})$ en forma de una convolución se utiliza la siguiente identidad:

$$nk = \frac{1}{2}(n^2 + k^2 - (k-n)^2) \tag{62}$$

Al reemplazar la ecuación (62) en la ecuación (61) se obtiene la ecuación (63) así:

$$X(e^{j\omega_k}) = \sum_{n=0}^{N-1} x[n]e^{-jw_0 n}W^{n^2/2}W^{k^2/2}W^{-(k-n)^2/2} \tag{63}$$

La ecuación (63) se puede ver como una convolución de dos señales a partir del siguiente reemplazo:

$$g[n] = x[n]e^{-j\omega_0 n}W^{n^2/2} \tag{64}$$

Con la ecuación (64) se puede rescribir la ecuación (63) de la siguiente forma:

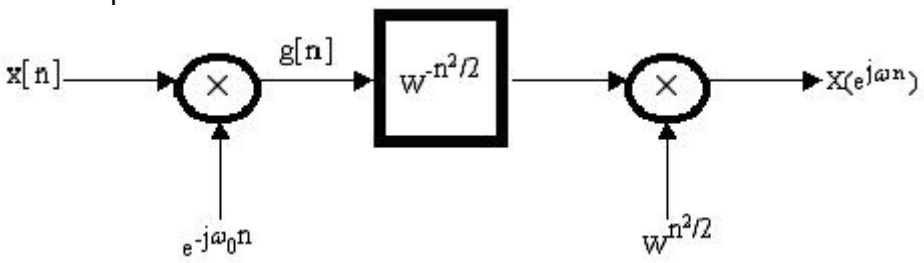
$$X(e^{j\omega_k}) = W^{k^2/2}\sum_{n=0}^{N-1} g[n]W^{-(k-n)^2/2} \quad \text{k=0,1,..., M-1} \tag{65}$$

Para obtener una ecuación similar a la convolución de dos señales se reemplaza la variable n por la variable k obteniéndose la siguiente expresión:

$$X(e^{j\omega_n}) = W^{n^2/2}\sum_{k=0}^{N-1} g[k]W^{-(n-k)^2/2} \quad \text{n=0,1,..., M-1} \tag{66}$$

La ecuación (66) corresponde a la convolución de las señales g[n] con la secuencia $W^{-n^2/2}$ seguida de la multiplicación por la secuencia $W^{n^2/2}$. El flujograma de la figura 15 muestra este procedimiento.

FIGURA 15. Flujograma para calcular el espectro parcial de una señal discreta con el algoritmo Chirp.



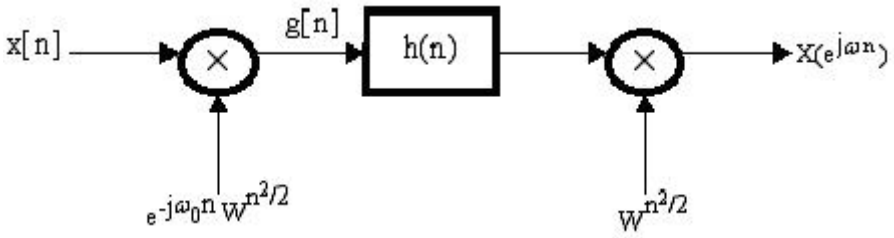
Fuente: el autor.

Para el cálculo parcial del espectro se debe notar que se requiere calcular la convolución entre la señal g[n] y la señal $W^{-n^2/2}$. Para llevar a cabo este cálculo se debe tener en cuenta que la convolución se realiza con una señal de longitud finita lo cual limita el cálculo de la misma al intervalo desde n = -(N-1) hasta n = (M-1) lo anterior conduce a la respuesta impulsional del sistema h(n) definida por la ecuación (67) de la siguiente forma:

$$h[n] = W^{-n^2/2} \quad -(N-1) \le n \le M-1 \quad \text{y h[n]=0 para otro valor de n} \tag{67}$$

Al tener en cuenta esta observación el flujograma de la figura 15 se convierte en el flujograma de la figura 16.

FIGURA 16. Transformada Chirp teniendo en cuenta la respuesta impulsional finita h[n].



Fuente: el autor.

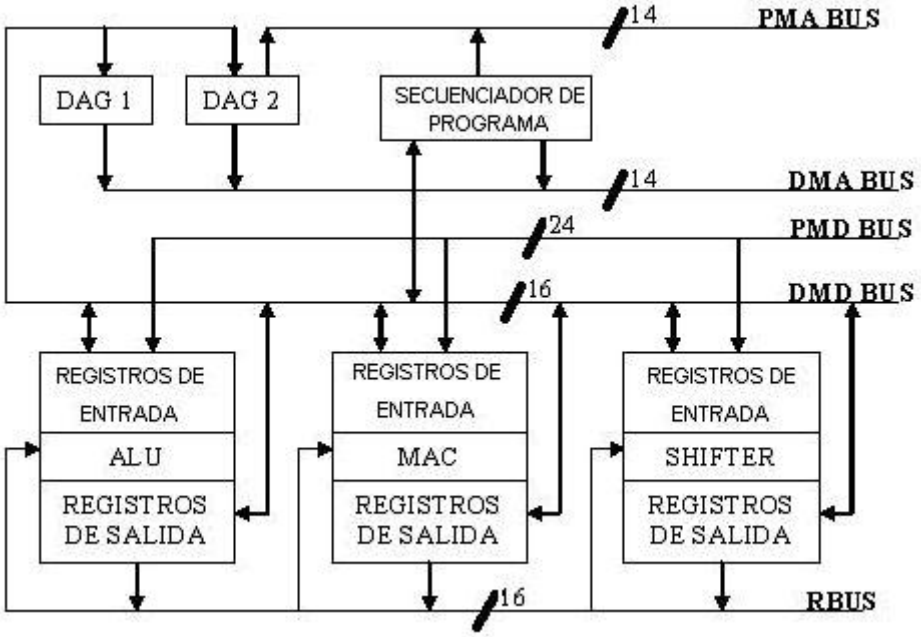
2. PROCESADOR DE SEÑALES DISCRETAS (DSP)

Todas las operaciones matemáticas que están implícitas en las relaciones mencionadas en el capítulo anterior pueden ser manejadas con procesadores de señales digitales (DSPs). En este capítulo se analizarán los componentes distintivos y novedosos que hacen que los DSPs sean una herramienta necesaria en las aplicaciones de procesamiento de señales digitales. Entre estos se encuentran: la unidad aritmética lógica (ALU) con la capacidad de realizar sumas y multiplicaciones acumuladas de manera simultánea, una unidad que realiza direccionamiento de vectores de datos de manera rápida sin consumir recursos de la memoria de acceso aleatorio (RAM) para el funcionamiento del vector de datos, un componente capaz de realizar desplazamientos aritméticos y lógicos sin agotar los recursos de la memoria RAM para dicha operación y una unidad que permite realizar ciclos repetitivos solamente estableciendo el punto final y el incremento del ciclo. Si el lector desea obtener mayor información acerca del ADSP2181 puede obtenerla en [ADSP2100 *Family User´s manual*, 95].

2.1. DESCRIPCIÓN GENERAL DEL ADSP2181.

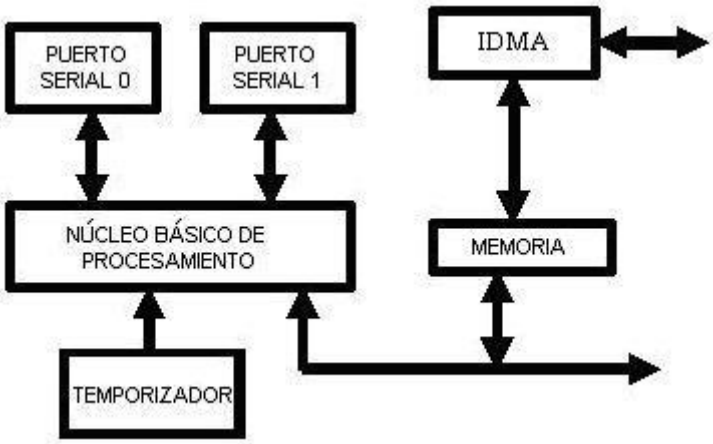
La implementación de los diferentes algoritmos de transformada discreta de Fourier se realizarán con un procesador que tiene una arquitectura de 16 bits, aritmética de coma fija con un núcleo básico formado por la unidad aritmético lógica (ALU), dos generadores de dirección de datos (DAG), un secuenciador de programa, una unidad de desplazamientos aritméticos y lógicos (SHIFTER) y una unidad acumuladora de multiplicaciones (MAC) (ver figura 17). Adicionalmente tiene unidades que le permiten comunicar este núcleo básico con dispositivos. Tales unidades son: puertos seriales, temporizador, acceso directo a memoria IDMA y la memoria RAM que se encuentra dividida en dos partes a saber: memoria de programa y memoria de datos. En la figura 18 se muestra un diagrama de bloques completo donde se observa la interconexión del núcleo con las unidades periféricas.

FIGURA 17. Núcleo básico de procesamiento del ADSP2181.



Fuente: el autor.

FIGURA 18. Interconexión del núcleo con los periféricos en el ADSP2181.



Fuente: el autor.

2.2. NÚCLEO BÁSICO DE PROCESAMIENTO DEL ADSP2181

En la figura 17 se muestran las unidades que componen el núcleo básico de procesamiento del ADSP2181. Estas se encuentran divididas en tres grupos, uno se encarga del procesamiento aritmético y lógico, otro maneja el direccionamiento de vectores de datos y un último grupo que ejecuta los bucles o ciclos repetitivos que se presentan en un programa.

2.2.1. Grupo de procesamiento aritmético.

La unidad ALU realiza operaciones aritméticas, lógicas y divisiones. La unidad MAC realiza multiplicación-adición y multiplicación-sustracción en un solo ciclo de máquina. La unidad SHIFTER se encarga de realizar desplazamientos lógicos y aritméticos, normalización, denormalización y es capaz de obtener el número mayor en un vector de datos.

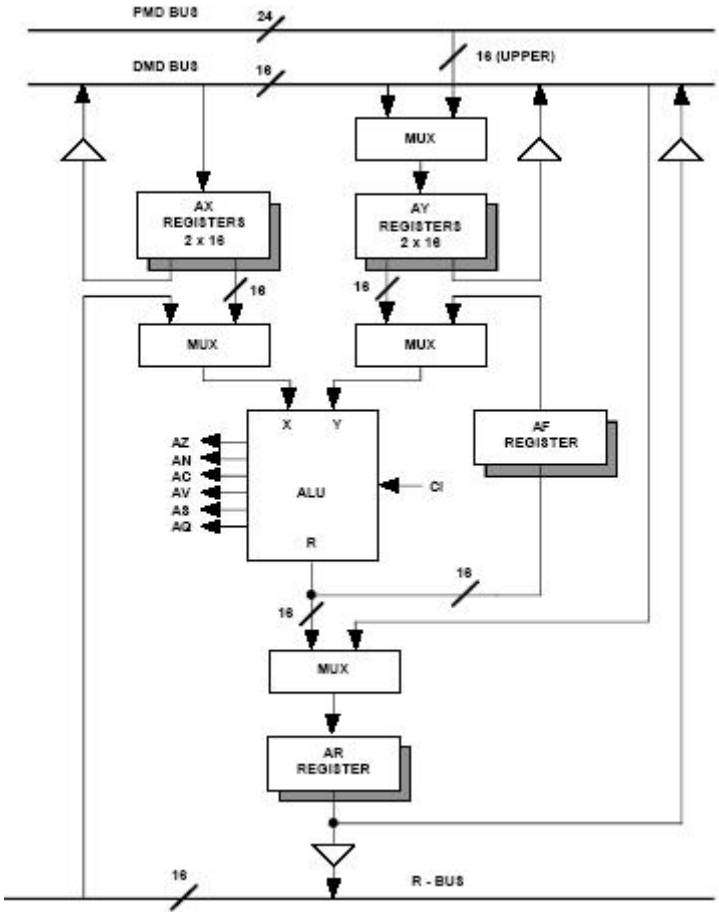
Las unidades de procesamiento aritmético estan dispuestas de forma paralela para que la salida de una unidad pueda ser la entrada de otra unidad en el siguiente ciclo de máquina. Cada unidad tiene registros de entrada y de salida que se encuentran conectados con la memoria de datos a través del DMD-BUS (ver figura 17).

2.2.1.1. Unidad aritmético-lógica (ALU).

Esta unidad tiene una longitud de bits igual a 16, posee 2 puertos de entrada, X e Y, y un puerto de salida R. El puerto X puede recibir señales desde el archivo de registros AX o desde el R-BUS (ver figura 19). El archivo de registros AX posee dos componentes AX0 y AX1 ambos con un ancho de 16 *bits* los cuales se pueden escribir y leer desde el DMD-BUS (ver figura 19). El puerto Y de la ALU acepta datos desde dos fuentes: el archivo de registros AY y desde el registro de realimentación AF. El archivo de registros AY al igual que AX esta conformado por los registros AY0 y AY1 los cuales también se pueden escribir y leer desde el DMD-BUS. El resultado de la última operación se almacena en dos registros: el registro de realimentación AF y en el registro resultado AR. El registro de realimentación AF le permite a la ALU utilizar el resultado como una entrada directa al puerto Y de entrada de la misma unidad. Al principio de un ciclo de máquina los registros de salida y entrada son leídos y al final del mismo se escriben, es decir, que el valor leído en un registro es el resultado de un ciclo previo. En la figura 19 se muestra el diagrama de bloques de la ALU y en la tabla 1 se describen las operaciones que ejecuta la unidad ALU.

Los resultados de la última operación efectuada por la ALU quedan registrados a través de las banderas del registro ASTAT. Estas banderas se evalúan a través de las instrucciones ocasionando bifurcaciones en la ejecución del programa. En la tabla 2 se establecen la definición y la operación de las banderas de este registro.

FIGURA 19. ALU del ADSP2181.



Fuente: *ADSP-2100 FAMILY USER´S MANUAL*.

TABLA 1. Operaciones de la unidad ALU del ADSP2181.

Operación	Descripción
R =X + Y	Sume puertos X e Y
R =X + Y + CI	Sume puertos X e Y con acarreo
R =X – Y	Restar puerto X del puerto Y
R =X – Y+CI -1	Restar puerto X del puerto Y con préstamo
R =Y – X	Restar puerto Y del puerto X
R =Y – X + CI – 1	Restar puerto Y del puerto X con préstamo
R = - X	Complemento a 2 del puerto X
R = - Y	Complemento a 2 del puerto Y
R = Y + 1	Incrementar el puerto Y
R = Y – 1	Decrementar el puerto Y
R = PASS X	Pasar el puerto X al resultado
R = PASS Y	Pasar el puerto Y al resultado
R = 0 (PASS 0)	Borrar el resultado
R = ABS X	Valor absoluto del puerto X

R = X AND Y	Operación lógica AND entre puerto X y puerto Y
R = X OR Y	Operación lógica OR entre puerto X y puerto Y
R = X XOR Y	Operación lógica XOR entre puerto X y puerto Y
R = NOT X	Negación lógica del puerto X
R = NOT Y	Negación lógica del puerto Y

TABLA 2. Banderas del registro ASTAT.

Bandera	Nombre	Definición
AZ	ZERO	Estado alto si el resultado de la ALU es igual a cero.
AN	NEGATIVE	Estado alto si el resultado de la ALU es negativo.
AV	OVERFLOW	Estado alto si existe rebosamiento aritmético de la ALU.
AC	CARRY	Salida de acarreo de la última etapa del sumador de la ALU.
AS	SIGN	Afectado solamente por la instrucción ABS X. Estado alto si el número previo era negativo.
AQ	QUOTIENT	Bit de cociente activado por instrucciones DIVS y DIVQ.

De la figura 17 se observa que las unidades MAC y SHIFTER pueden convertirse en las entradas de los puertos X e Y de la unidad ALU, esto facilita las operaciones aritméticas del procesador ADSP2181. En la tabla 3 se muestran las entradas a los puertos X e Y y los registros en los que se almacena el resultado de la ALU.

TABLA 3. Posibles entradas y salidas de la ALU.

Entradas del puerto X	Entradas del puerto Y	Puerto de salida
AX0, AX1 (ALU)	AY0, AY1 (ALU)	AR (ALU)
AR (ALU)	AF (ALU)	AF (ALU)
MR0, MR1, MR2 (MAC)		
SR0, SR1 (SHIFTER)		

2.2.1.2. Unidad de multiplicación y acumulación (MAC).

La unidad MAC realiza las siguientes funciones aritméticas: la multiplicación, la multiplicación con adición acumulativa, la multiplicación con sustracción acumulativa, la saturación y la eliminación del registro resultado. Una función de realimentación permite que parte de la salida del acumulador se convierta en uno de los multiplicadores de la siguiente operación de la unidad MAC.

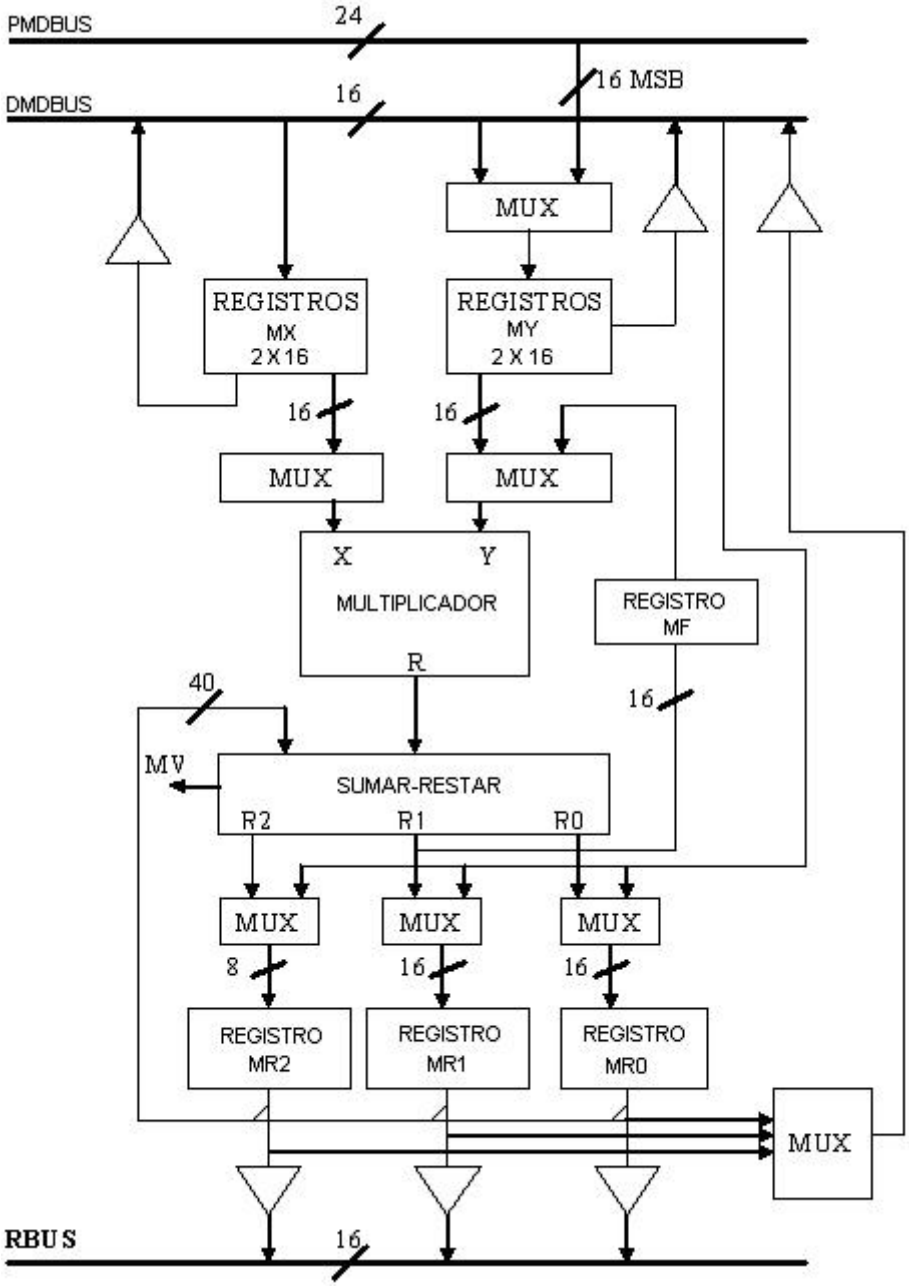
El multiplicador posee dos puertos de entrada X e Y y un puerto de salida P con una capacidad de 32 *bits*. El resultado del multiplicador pasa a una unidad sumadora-restadora de 40 *bits* que lo suma o lo resta con el contenido del registro MR del multiplicador. El registro MR esta compuesto por los registros MR0, MR1 y MR2. Los registros MR0 y MR1 son de 16 *bits* mientras que MR2 es de 8 *bits*. La figura 20 detalla el funcionamiento interno de la unidad MAC.

La unidad MAC esta en capacidad de realizar las operaciones aritméticas listadas en la tabla 4. En la tabla 5 se muestran las posibles entradas y salidas de la unidad MAC.

TABLA 4. Operaciones aritméticas de la MAC.

Operación	**Descripción**
X*Y	Multiplicar los puertos X e Y.
MR + X*Y	Multiplicar el puerto Y con el puerto X y sumar el resultado con el registro MR.
MR – X*Y	Multiplicar el puerto Y con el puerto X y restar el resultado con el registro MR.
0	Borrar el registro MR.

FIGURA 20. Unidad MAC.



Fuente: el autor.

TABLA 5. Posibles entradas y salidas de la unidad MAC.

Entradas para el puerto X	Entradas para el puerto Y	Salidas
MX0, MX1	MY0, MY1	MR (MR0, MR1, MR2)
AR	MF	MF
MR0, MR1, MR2		
SR0, SR1		

Para facilitar operaciones de multiprecisión la MAC acepta números con signo o sin signo como entradas a sus puertos. La precisión de los operandos se especifica en la instrucción del programa. La tabla 6 muestra los formatos aceptados por la MAC.

TABLA 6. Formatos de entrada de los puertos X e Y.

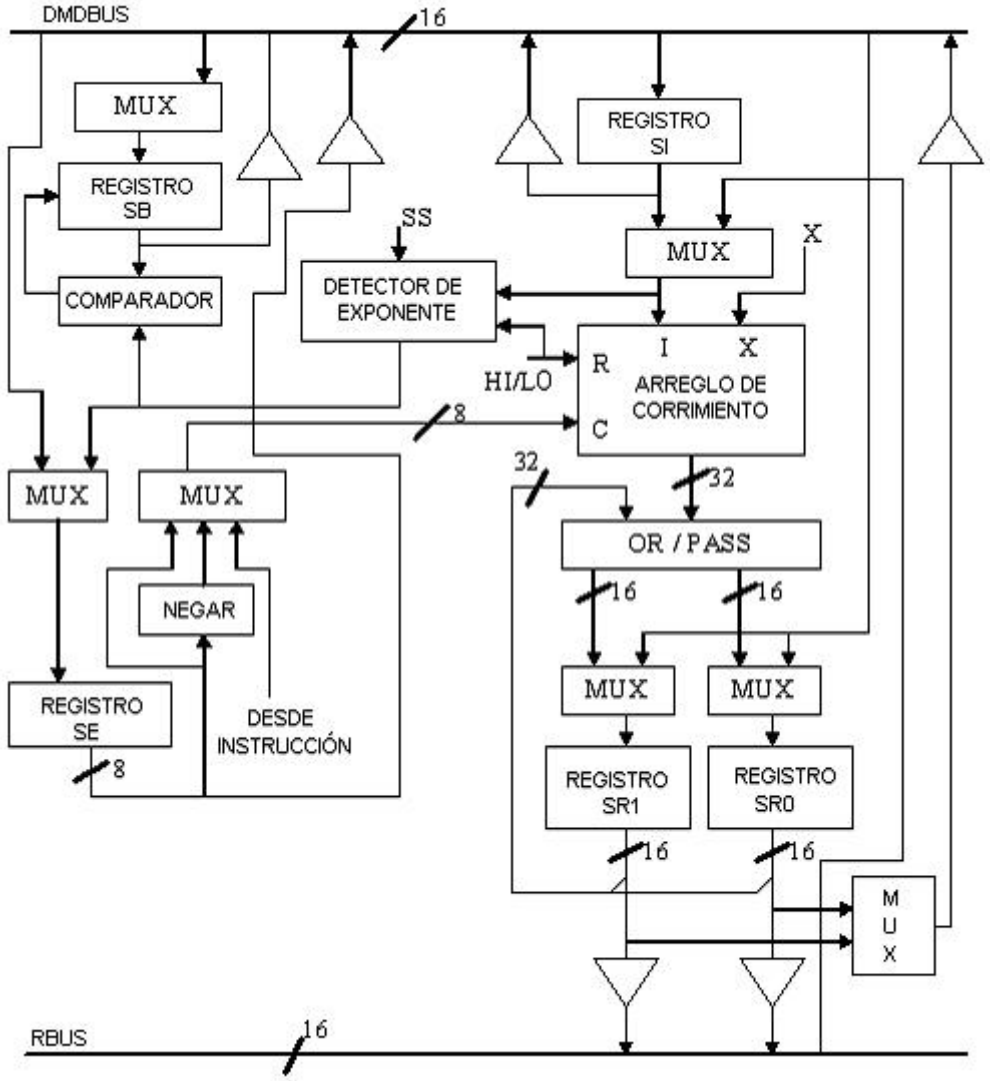
Puerto de entrada X	**Puerto de entrada Y**
Signo	Signo
Sin signo	Signo
Signo	Sin signo
Sin signo	Sin signo

El sumador-restador de la MAC genera una señal de sobrecarga cada vez que la última operación ejecutada por esta unidad sobrepasa el límite de los 32 *bits*. Esto se puede apreciar en el bit MV el cual se pone en estado alto cada vez que el registro resultado MR no tiene sus últimos 9 bits en alto o en bajo. Lo anterior tiene como base el sistema de números binarios en complemento a dos.

2.2.1.3. Unidad de corrimientos aritméticos y lógicos SHIFTER.

En la unidad SHIFTER se generan los corrimientos lógicos y aritméticos y la normalización; además se obtiene el exponente de un número y el exponente común a un grupo de números almacenados en un vector. El vector de corrimientos es un desfasador de 16 a 32 *bits*, es decir, recibe 16 *bits* y los puede acomodar en cualquiera de los 32 *bits* del campo de salida de la unidad. Los registros que componen la unidad de corrimientos son: SI registro de entrada de 16 *bits*, SR es el registro resultado de 32 *bits* dividido en dos registros de 16 *bits* SR0 y SR1, SE es un registro de 8 *bits* que mantiene el valor del exponente en las operaciones de normalización y desnormalización y el registro SB con una longitud de 5 *bits* que almacena el valor del exponente mas grande en un bloque de datos. En la figura 21 se muestra la unidad de corrimientos.

FIGURA 21. Unidad de corrimientos SHIFTER.



Fuente: el autor.

En las tablas 7 y 8 se muestran las entradas y salidas de la unidad de corrimientos SHIFTER y las operaciones que ejecuta respectivamente.

TABLA 7. Operaciones de la unidad de corrimientos SHIFTER.

Operación	Descripción
ASHIFT	Corrimiento aritmético
LSHIFT	Corrimiento lógico
NORM	Normalización de un número
EXP	Extraer el exponente de un número
EXPADJ	Ajustar el exponente de un bloque de números

TABLA 8. Entradas y salidas de la unidad de corrimientos SHIFTER.

Entradas	Salidas
SI	SR (SR0, SR1)
AR	
MR0, MR1, MR2	
SR0, SR1	

2.2.2. Grupo de direccionamiento de vectores de datos.

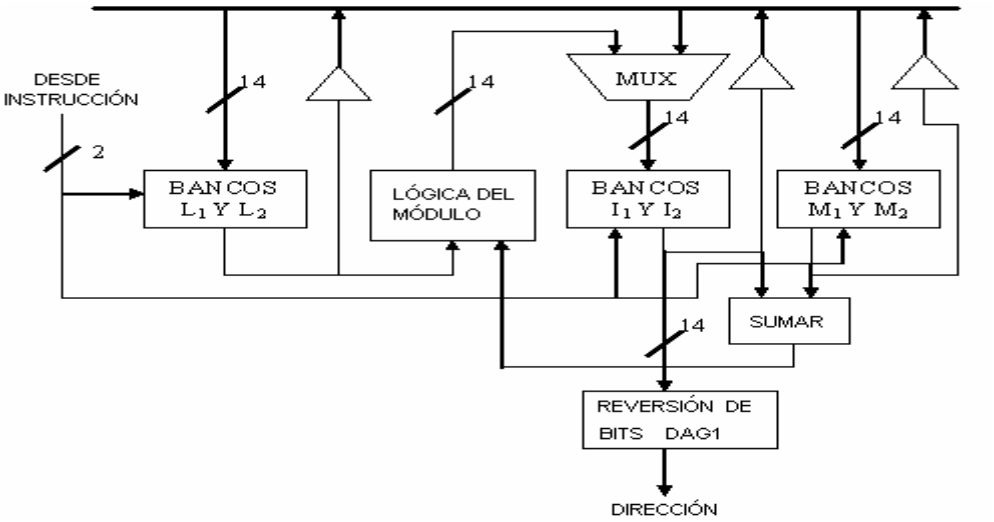
En esta parte del capítulo se analizarán las unidades encargadas de realizar el control de movimiento de datos desde o hacia el procesador. Estas son los generadores de dirección de datos DAGs por su sigla en inglés y la unidad que se encarga del intercambio de información entre la memoria de datos y la memoria de programa del DSP.

2.2.2.1. Generadores de direcciones de datos (DAGs).

Estos dos dispositivos (DAG1 y DAG2) están en capacidad de realizar direccionamiento indirecto, modificación automática de las direcciones de un vector y capacidad de manejo de vectores circulares. El DAG1 tiene dos funciones: suministrar direcciones de la memoria de datos e invertir los *bits* de la dirección de un registro. El DAG2 genera direcciones de la memoria de datos y de la memoria de programa pero no esta en capacidad de generar inversión de bits.

Cada unidad de direccionamiento está conformada por tres bancos de registro; cada banco posee 4 registros. En la tabla 9 se muestra la distribución de los registros en cada una de las unidades DAG y en la figura 22 se muestra el diagrama de bloques de la unidad de dirección de datos.

FIGURA 22. Unidad de dirección de datos.



Fuente: el autor.

TABLA 9. Distribución de los registros en las unidades DAG.

DAG	Índices	Longitud	Modificador
1	I_0, I_1, I_2, I_3	L_0, L_1, L_2, L_3	M_0, M_1, M_2, M_3
2	I_4, I_5, I_6, I_7	L_4, L_5, L_6, L_7	M_4, M_5, M_6, M_7

Los registros índices almacenan el valor de una dirección, los registros de longitud almacenan la cantidad de información que posee un vector de datos y los registros modificadores permiten realizar incremento o decremento de los vectores índices una vez se ha ejecutado una instrucción.

Las unidades DAGs permiten el direccionamiento indirecto, directo y de vectores circulares. Un vector circular es un tipo de almacenamiento de datos que puede ser recorrido de uno en uno de dos en dos o de n en n sin tener que dedicar una variable específica para tal propósito; adicionalmente todo vector circular tiene la característica de identificar el final del mismo y con ello saltar a una dirección que puede ser la inicial u otra dentro del vector circular. Los registros índices almacenan la dirección actualizada del vector circular, los registros modificadores guardan la magnitud del vector circular y L0 lleva el desplazamiento o el valor del incremento en la dirección cada vez que se pasa por la instrucción que hace referencia al vector circular. Un ejemplo de funcionamiento de un vector circular se muestra en la figura 23 donde el desplazamiento se realiza de dos en dos en el vector de 5 datos.

FIGURA 23. Funcionamiento de un vector circular.

BUFFER	ETAPA 1	ETAPA 2	ETAPA 3
DATO 0	← Io		← Io
DATO 1		← Io	
DATO 2	← Io		← Io
DATO 3		← Io	
DATO 4	← Io		← Io

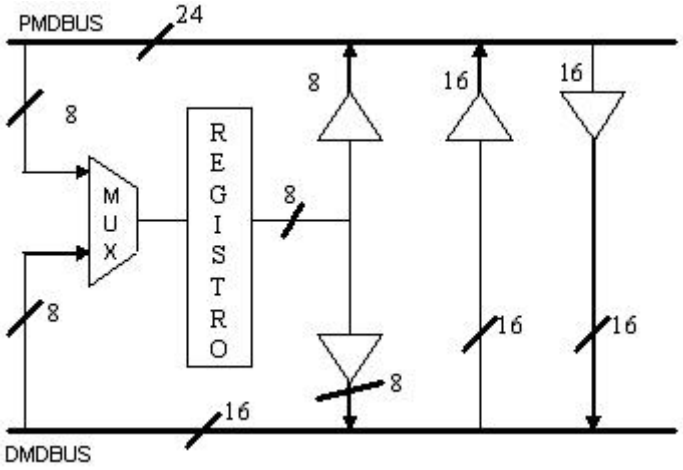
Fuente: el autor.

La inversión de bits se pude lograr en forma automática habilitando un *bit* del registro interno MSTAT. Los únicos registros índices que tienen esta capacidad son I_0, I_1, I_2 e I_3.

2.2.2.2. Unidad de intercambio de datos entre la memoria de programa y la memoria de datos.

Esta unidad es la encargada de acoplar el *bus* de la memoria de datos con el "bus" de la memoria de programa, permitiendo el intercambio de información entre estas dos memorias. Ya que el *bus* de la memoria de programa es de 24 *bits* y el de la memoria de datos es de 16 *bits* solamente se pueden transferir 16 *bits* de la memoria de programa en forma directa; la parte restante se transfiere a través de un registro auxiliar llamado PX. En la figura 24 se muestra la unidad de intercambio.

FIGURA 24. Unidad de intercambio de datos entre la memoria de programa y la memoria de datos.



Fuente: el autor.

2.2.3. Grupo de ejecución de programa (PROGRAM SEQUENCER).

La unidad PROGRAM SEQUENCER controla el flujo de ejecución del programa cargado en la memoria. Permite ejecución de instrucciones de forma secuencial, ejecución de bucles sin instrucciones de revisión de fin de ciclo y aumento de contadores, servicio de interrupciones, saltos y llamados a subrutinas que se ejecutan en un solo ciclo de máquina. La figura 25 muestra la unidad PROGRAM SEQUENCER.

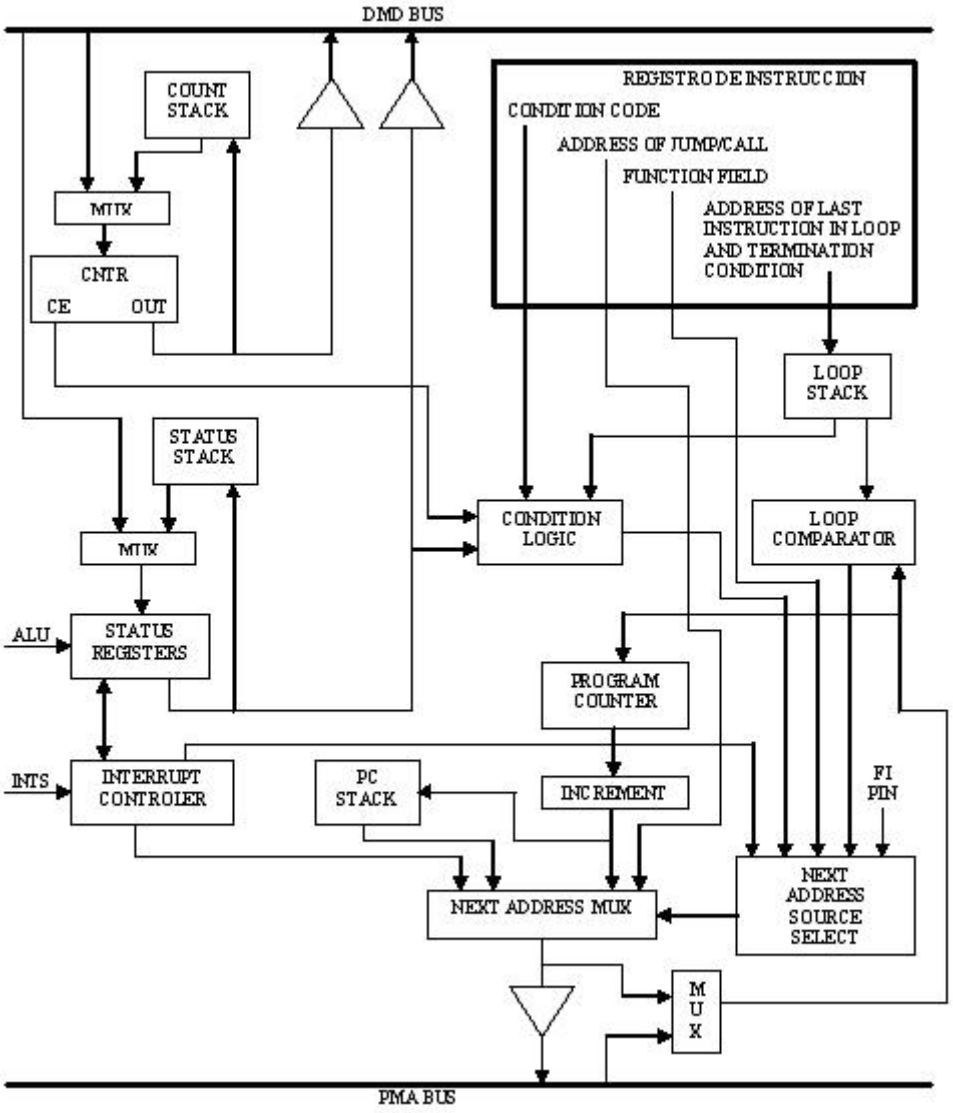
Para llevar a cabo las operaciones internas previamente mencionadas el DSP recurre a las instrucciones de control del programa: DO UNTIL, JUMP, CALL, RTS, RTI e IDLE. La instrucción DO UNTIL ejecuta un bucle hasta que una condición se cumpla. La instrucción JUMP ejecuta un salto incondicional. La instrucción CALL llama una subrutina. La instrucción RTS genera el retorno de una subrutina. La instrucción RTI regresa de una interrupción. La instrucción IDLE genera en el DSP un estado de espera de bajo consumo de potencia hasta que ocurra una interrupción.

2.2.3.1. Circuito de selección de la siguiente dirección.

Esta unidad se encarga de generar la dirección siguiente mientras que el procesador se encuentra procesando la instrucción actual. Esta unidad puede cargarse con el valor del incremento del contador, con el valor almacenado en la pila del contador de programa, con el registro de instrucción y con el controlador de interrupciones.

El valor del incremento del contador es la fuente de la siguiente dirección si el flujo del programa es secuencial, se termina un ciclo DO UNTIL o cuando una condición de una instrucción JUMP es falsa.

FIGURA 25. Unidad secuenciadora de programa PROGRAM SEQUENCER.



Fuente: el autor.

La fuente de la siguiente dirección es la pila del contador de programa cuando el procesador requiere retornar de una subrutina o cuando termina de atender una interrupción.

El registro de instrucción se convierte en el origen de la siguiente dirección, cuando se utiliza un salto directo con la instrucción JUMP.

Finalmente el controlador de interrupción es la fuente de la siguiente dirección, cuando el procesador necesita atender una interrupción.

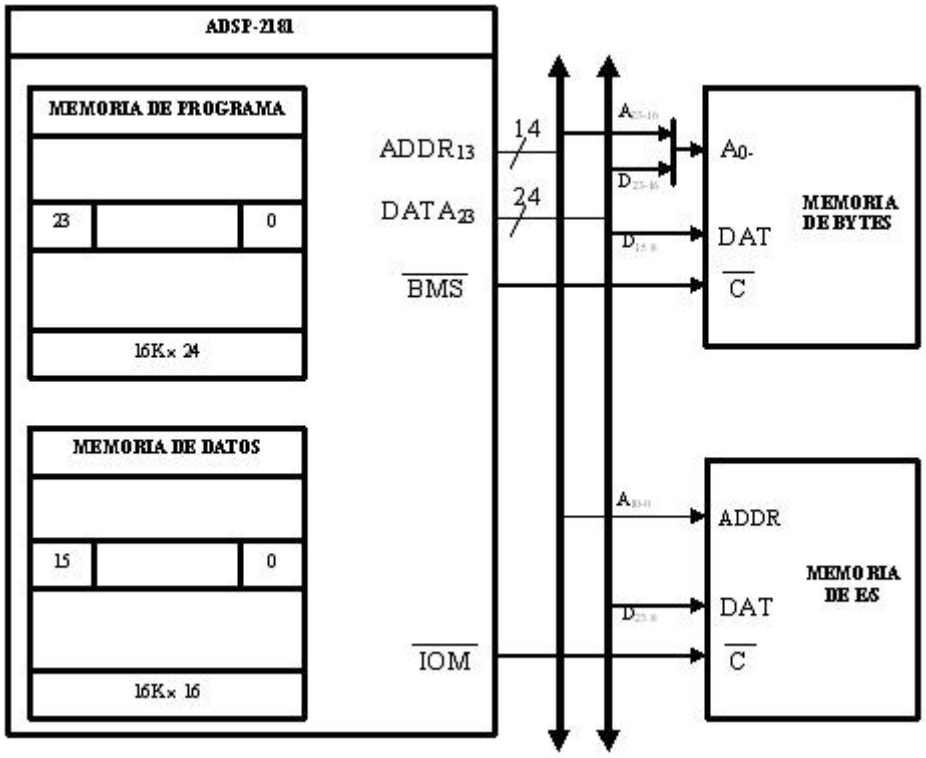
2.2.3.2. Contador y pila de bucles.

Este mecanismo permite ejecutar bucles de manera rápida sin perder tiempo en la verificación de la condición de salida ni en el incremento de la variable interna que almacena el valor de la etapa en la que se encuentra el ciclo. Antes de iniciar un bucle se debe cargar el número de veces que se va a ejecutar el ciclo en el registro CNTR. A continuación cuando el procesador detecta la instrucción DO decrementa automáticamente el valor de CNTR hasta que la bandera CE (COUNTER EXPIRED) se carga con un nivel lógico alto. El final del ciclo se carga en un registro LOOP COMPARATOR; este valor corresponde a la dirección donde se finaliza el bucle. Cuando el contador de programa arriba a esta dirección, automáticamente se genera un retorno a la primera dirección del ciclo DO UNTIL. El procesador cuenta con una pila que almacena esta dirección en caso que el flujo del programa requiera de bucles anidados. Cada vez que aparece una instrucción DO UNTIL sin haber llegado al final del ciclo anterior la dirección final del ciclo precedente se almacena en la pila de bucles. La pila de bucles posibilita la ejecución de cuatro bucles anidados.

2.3. ORGANIZACIÓN DE LA MEMORIA DEL ADSP2181

El procesador ADSP2181 tiene cuatro espacios de memoria diferentes entre los cuales están: la memoria de datos, la memoria de programa, la memoria de puertos de entrada y salida y la memoria de *bytes*. La memoria de programa tiene una capacidad de 16 *Kbytes* cada celda con un ancho de 24 *bits* y se dedica al almacenamiento del programa principal y de datos. El almacenamiento de datos tiene asignado 16 *Kbytes* de memoria con un ancho de 16 *bits* por celda y se dedica exclusivamente a guardar variables de acceso aleatorio. El manejo de los puertos de entrada y salida dispone de 2 *Kbytes* con un ancho de 16 *bits* en cada celda, dedicados a la comunicación con dispositivos externos al procesador. La memoria de *bytes* hace referencia a la capacidad de direccionamiento externo que posee el procesador ADSP2181. Este dispositivo es capaz de direccionar 4 *Mbytes* de memoria externa con un ancho de 8 *bits* cada celda. En la figura 26 se muestra los espacios de memoria del ADSP2181 con las capacidades de memoria, ancho de celda y pines de control.

FIGURA 26. Memoria del ADSP2181.

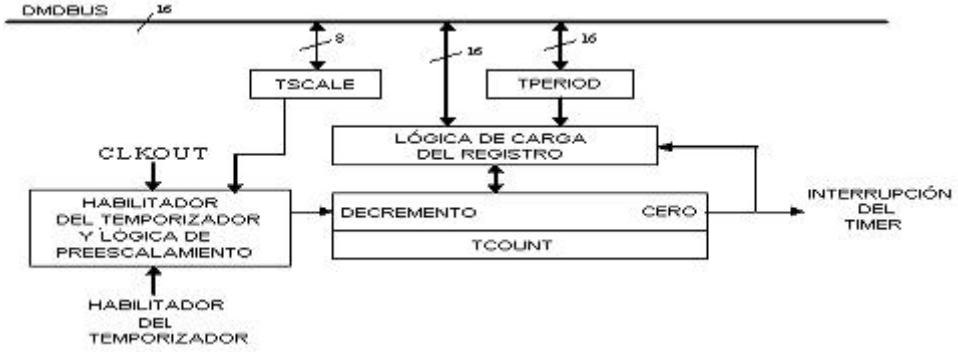


Fuente: el autor.

2.4. TEMPORIZADOR

La unidad de temporización se encarga de programar interrupciones cada cierta cantidad de tiempo. El temporizador posee tres registros que permiten programar el tiempo de los retardos, estos registros son: TCOUNT, TPERIOD y TSCALE. TCOUNT es un registro de 16 *bits* que se utiliza como contador y se decrementa tan a menudo como una vez por cada ciclo de máquina. Cuando llega el contador a cero se recarga con el valor del registro TPERIOD y el conteo de TCOUNT empieza de nuevo. TSCALE es un registro de 8 *bits* que se encarga de cambiar el número de ciclos que le toma al registro TCOUNT en decrementarse. En la figura 27 se muestra el diagrama de bloques del temporizador.

FIGURA 27. Temporizador del ADSP2181.



Fuente: el autor.

2.5. PUERTOS SERIALES

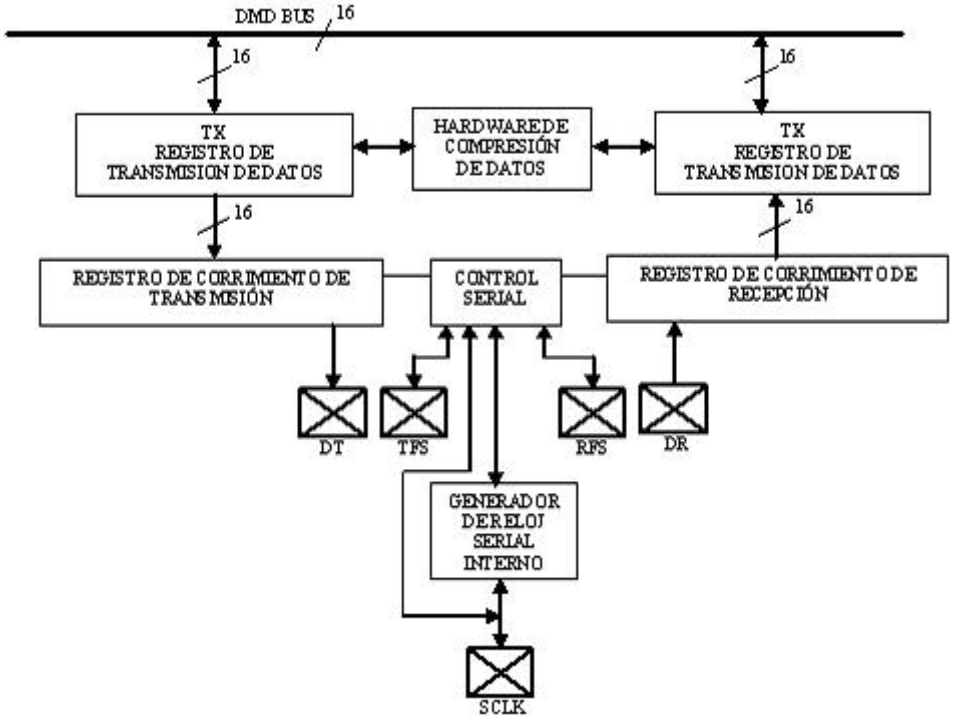
Los dos puertos seriales del procesador ADSP2181 se pueden configurar para aceptar diferentes tipos de protocolos de comunicaciones que le permiten al DSP intercomunicarse con otros procesadores. Las principales características de los puertos seriales del DSP se enumeran a continuación:

- Son bidireccionales, es decir permiten la transmisión y la recepción de datos de manera independiente y simultánea.
- Cada puerto serial del DSP posee un registro de transmisión y otro para la recepción de información.
- Poseen señales de reloj que pueden ser generadas internamente o manejadas por dispositivos externos.
- El tamaño de la palabra recibida o transmitida puede configurarse en longitudes que varían desde los 3 *bits* hasta los 16 *bits*.
- Salidas o entradas que le permiten recibir o enviar señales de sincronismo de dispositivos externos.
- Compresión de datos ley A y ley μ directamente en el *hardware* del puerto serial.
- Utilizando la unidad de direccionamiento de datos DAG se pueden recibir o transmitir vectores circulares de datos de manera automática, es decir, no es necesario utilizar ningún tipo de programación para realizar este propósito.
- Genera interrupciones por final de transmisión o por final de recepción de datos.
- Capacidad de multicanal en los puertos seriales. Lo anterior significa que se pueden conectar de 24 a 32 dispositivos externos a través del mismo puerto serial.

Escribir en el registro TX inicializa el puerto serial del DSP para la transmisión de datos; la señal TFS inicia la transmisión serial de los datos almacenados en el registro

TX. Apenas inicia la transmisión de datos cada *bit* del registro TX se transfiere al registro de corrimiento; cada *bit* se transmite en el flanco ascendente del reloj que maneja el puerto serial. En la recepción de datos, los *bits* recibidos se acumulan sucesivamente en el registro de corrimiento de recepción; cuando se completa toda la trama de datos en el registro RX se almacena el valor del registro de corrimiento de recepción. En la figura 28 se observa un diagrama de bloques de uno de los puertos seriales del ADSP2181 y en la tabla 10 se describen las fuentes de cada uno de los pines del puerto serial.

FIGURA 28. Puerto serial del ADSP2181.



Fuente: el autor.

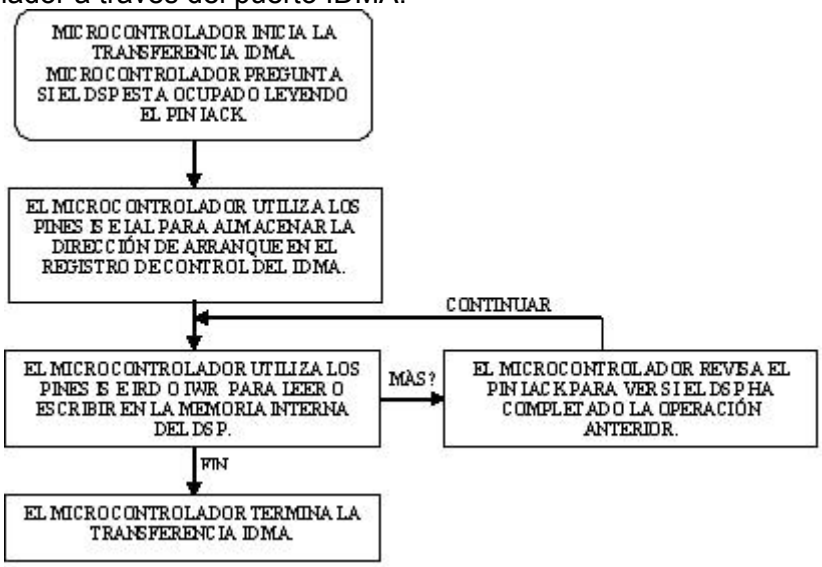
TABLA 10. Función de los pines del puerto serial

Nombre del pin	Función
SCLK	Reloj serial
RFS	Señal de sincronismo para recepción de datos
TFS	Señal de sincronismo para transmisión de datos
DR	Recepción de datos
DT	Transmisión de datos

2.6. PUERTO DE ACCESO DIRECTO A MEMORIA (IDMA).

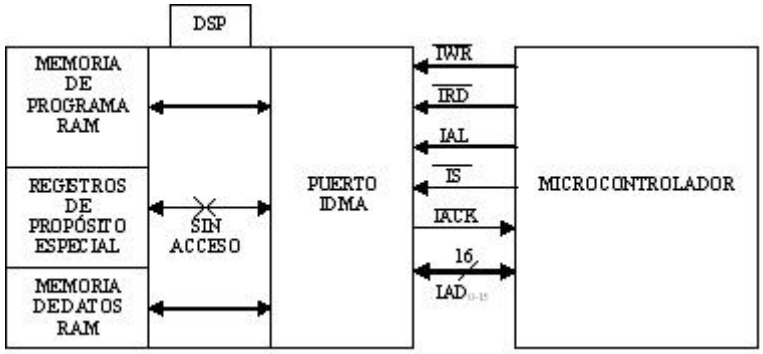
El puerto IDMA permite el acceso a la memoria interna de manera directa sin interrumpir el programa en curso. Las únicas posiciones a las cuales el puerto IDMA no tiene permiso de acceso son los registros de propósito especial encargados de configurar el funcionamiento interno del DSP.

FIGURA 29. Diagrama de flujo de una transferencia típica entre un DSP y un microcontrolador a través del puerto IDMA.



Fuente: el autor.

FIGURA 30. Conexión de un DSP con un microcontrolador a través del puerto IDMA.



Fuente: el autor.

Para llevar a cabo esta operación se necesita utilizar un espacio de la memoria de datos que se pueda acceder a través del IDMA para el almacenamiento temporal de los registros de propósito especial. Si el microcontrolador necesita leer o escribir en estos registros debe utilizar un pin de interrupción externa con el cual se le indica al DSP que debe iniciar la transferencia de los registros de propósito especial hacia el

espacio de memoria auxiliar. Después de interrumpir el DSP el microcontrolador puede iniciar un ciclo de lectura o escritura IDMA para tener acceso a los registros de propósito especial del DSP. En la figura 29 se muestra un diagrama de flujo que explica un ciclo típico cuando es utilizado el puerto IDMA.

En la figura 30 se muestra como se conecta un microcontrolador con el ADSP2181 a través del puerto IDMA. La tabla 11 describe las funciones de los pines del puerto IDMA.

TABLA 11. Función de los pines del puerto IDMA del ADSP2181.

Nombre del pin	**Salida (S) o entrada (E)**	**Función**
IRD	E	Pulso de lectura
IWR	E	Pulso de escritura
IS	E	Selector de IDMA
IAL	E	Habilitador del "latch" del IDMA
IAD_{0-15}	E/S	Bus de datos del puerto IDMA.
IACK	S	Indica si esta el puerto IDMA esta listo para iniciar comunicaciones con un dispositivo externo.

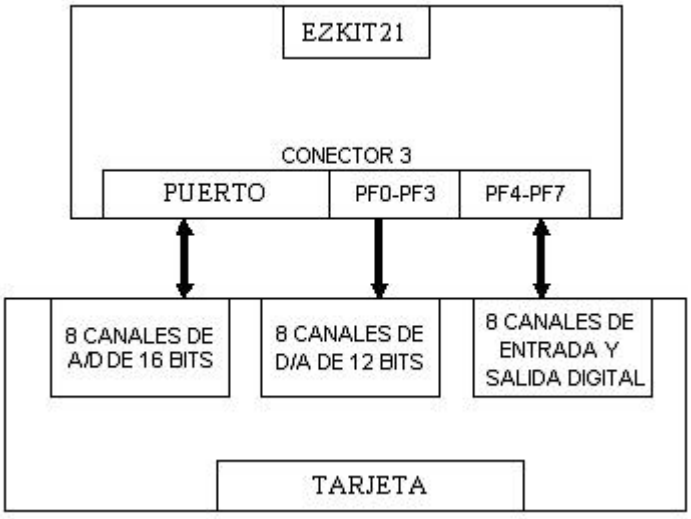
3. CONSTRUCCIÓN DE LA TARJETA ADAPTABLE PARA LA TARJETA DE EVALUACIÓN EZKITLITE 2181

La interfase adaptable se hace necesaria ya que la tarjeta EZKITLITE 2181 solamente posee un canal de adquisición de información analógica y un canal de generación de señales analógicas a través del CODEC AD1847. A pesar de esta limitante la tarjeta brinda la posibilidad al usuario para que construya prototipos y los comunique con ella utilizando los conectores destinados para dicho fin. La tarjeta adaptable tiene tres partes fundamentales las cuales son:

- 8 canales de conversión analógica a digital de 16 *bits* cada uno con ancho de banda ajustable.
- 8 canales de salida analógica de 12 *bits*.
- 8 canales de entrada o salida digital de 0 a 5 voltios.

Un diagrama de bloques de la tarjeta adaptable se muestra en la figura 31 y en este capítulo se describirán las funciones que realizan cada una de las componentes de ella.

FIGURA 31. Conexión de la tarjeta adaptable con la tarjeta EZKIT2181.



Fuente: el autor.

3.1. CANAL DE ADQUISICIÓN DE SEÑALES ANALÓGICAS DE 16 BITS Y ANCHO DE BANDA AJUSTABLE.

Cada uno de los 8 canales de adquisición de señales analógicas esta compuesto por un filtro antisolapamiento de segundo orden con una referencia de 2,5 V y un

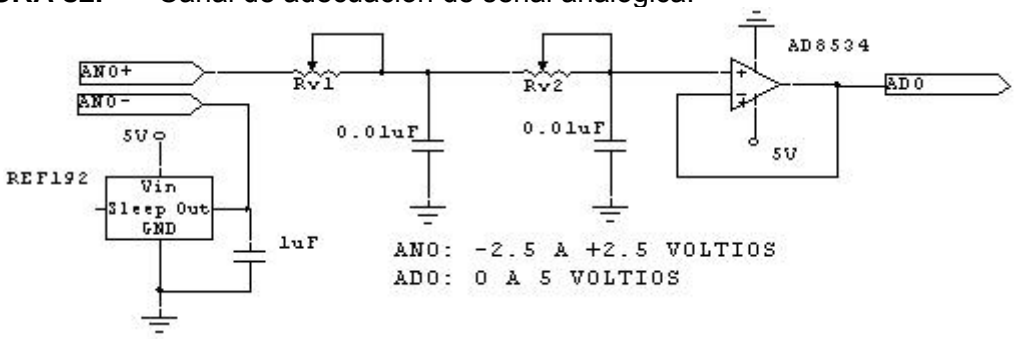
amplificador operacional AD8534. La referencia de tensión de 2,5 V se utiliza para elevar la señal de entrada que fluctúa entre ±2,5 V a una señal que varía entre 0 y 5 V. La señal resultante es la siguiente:

$$V_1 = 2{,}5 + AN_0 \tag{68}$$

El amplificador operacional se conecta en la configuración de seguidor de tensión para brindar las características de baja impedancia que necesita el convertidor analógico-digital para funcionar adecuadamente. El filtro de segundo orden pasa-bajas que delimita el ancho de banda de la señal de entrada tiene como característica la variabilidad de la frecuencia de corte; esto se logra utilizando potenciometros digitales (ver figura 32) que varían su resistencia entre 500 y $1\times10^5\Omega$, logrando con ello que la frecuencia de corte fluctúe entre 61,7 Hz y 12000 Hz. La ecuación (69) relaciona la salida del amplificador operacional con la señal de entrada V_1. La figura 32 muestra uno de los 8 canales de adecuación de señal analógica mientras que la figura 33 muestra el desempeño del filtro para un valor de R_v igual a 500Ω.

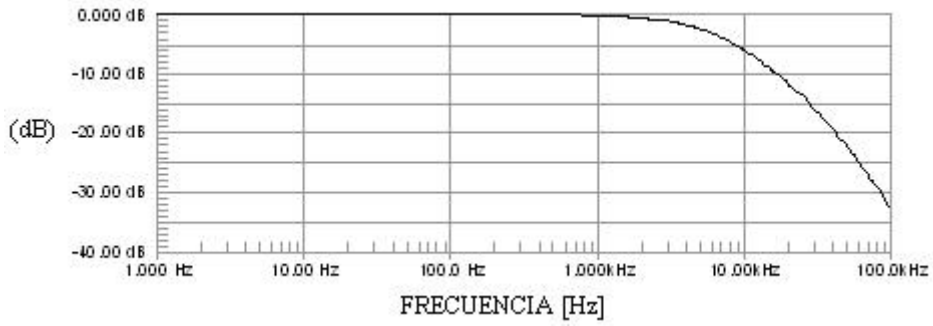
$$AD_0 = [2.5 + AN_0][\frac{1}{1 + SCR_V}]^2 \tag{69}$$

FIGURA 32. Canal de adecuación de señal analógica.



Fuente: El autor. Elaborado con el "software" CIRCUITMAKER.

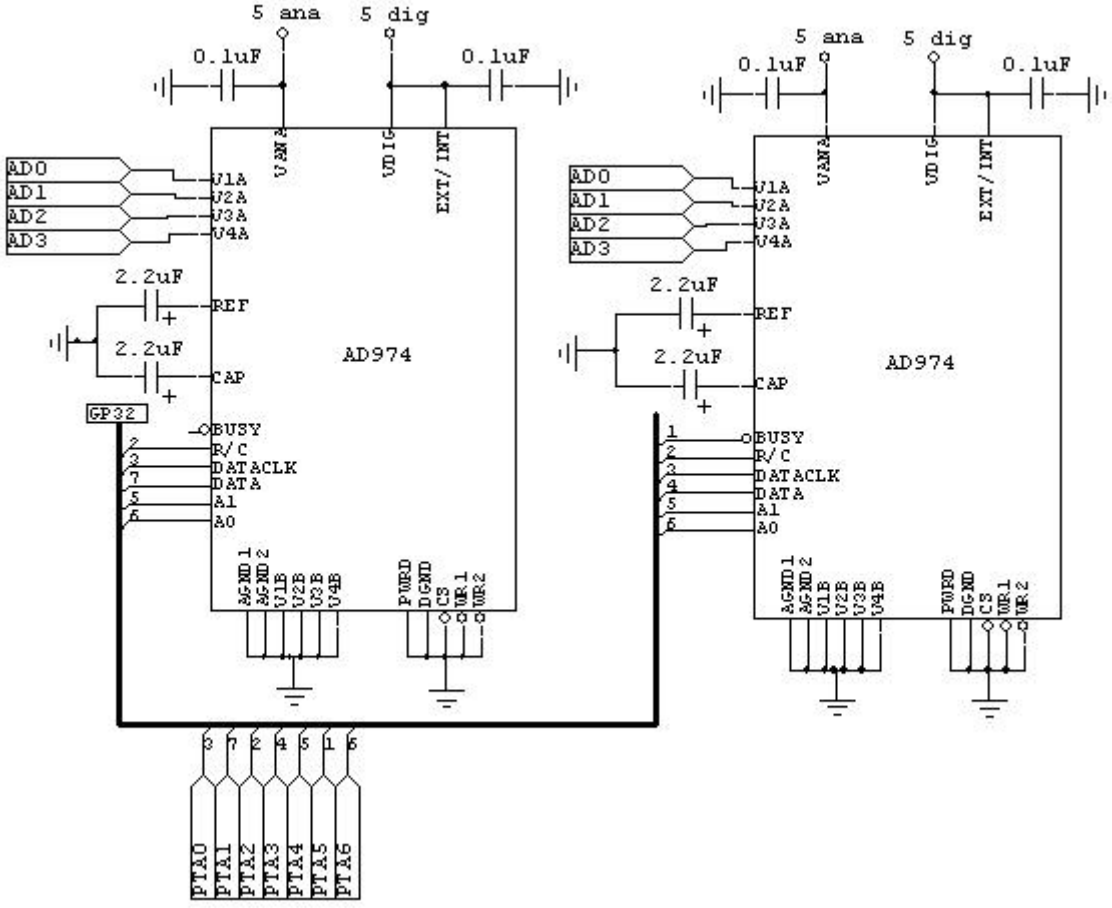
FIGURA 33. Curva de magnitud (dB) Vs frecuencia de la tensión AD0 para R_v igual a 500 Ω.



Fuente: El autor. Elaborado con el "software" CIRCUITMAKER.

El convertidor analógico-digital seleccionado tiene las siguientes características: resolución de 16 *bits*, frecuencia de muestreo máxima de 200×10^3 muestras por segundo, 4 canales analógicos de entrada multiplexados y salida del valor convertido de tensión en formato serial. Para llevar a cabo la tarea de 8 canales de adquisición de señales analógicas se requieren 2 dispositivos fabricados por ANALOG DEVICES cuya referencia es AD974; la conexión de ellos con el circuito de adecuamiento de señal y el microcontrolador se observa en la figura 34; además, en la figura 37 se presenta un flujograma que muestra la subrutina para leer la información digital de 2 AD974 con un microcontrolador 68HC908GP32.

FIGURA 34. Conexión de los AD974 con el canal de adecuamiento y el microcontrolador.



Fuente: El autor. Elaborado con el "software" CIRCUITMAKER.

El canal de acondicionamiento de la señal requiere de potenciometros digitales para variar la frecuencia del filtro pasa-bajas; estos dispositivos tienen 4 resistencias variables que varían desde los 45Ω hasta los 100kΩ, las cuales cubren las necesidades de dos canales de adecuamiento de señal. Estos dispositivos son fabricados por ANALOG DEVICES y tienen la referencia AD5204-100. El valor mas

bajo de resistencia no se puede utilizar ya que los potenciometros están restringidos a un consumo máximo de corriente de 20 mA. Teniendo en cuenta este detalle y el valor máximo de tensión que podría llegar a tener el canal de adecuamiento, 5 V, se deduce que el valor mínimo de resistencia admisible para esta aplicación es de 250Ω. Para prevenir posibles daños en cualquiera de los potenciometros, la subrutina de *software* que controla las resistencias variables debe tener en cuenta esta limitante. La ecuación (70) muestra la relación entre la resistencia y el código digital de entrada al potenciometro digital.

$$R_{WA}(dx) = \{[100k\Omega \times (256 - dx)]/256\} + R_W \quad (70)$$

R_w: resistencia del variador (45Ω).

dx: dato digital de entrada

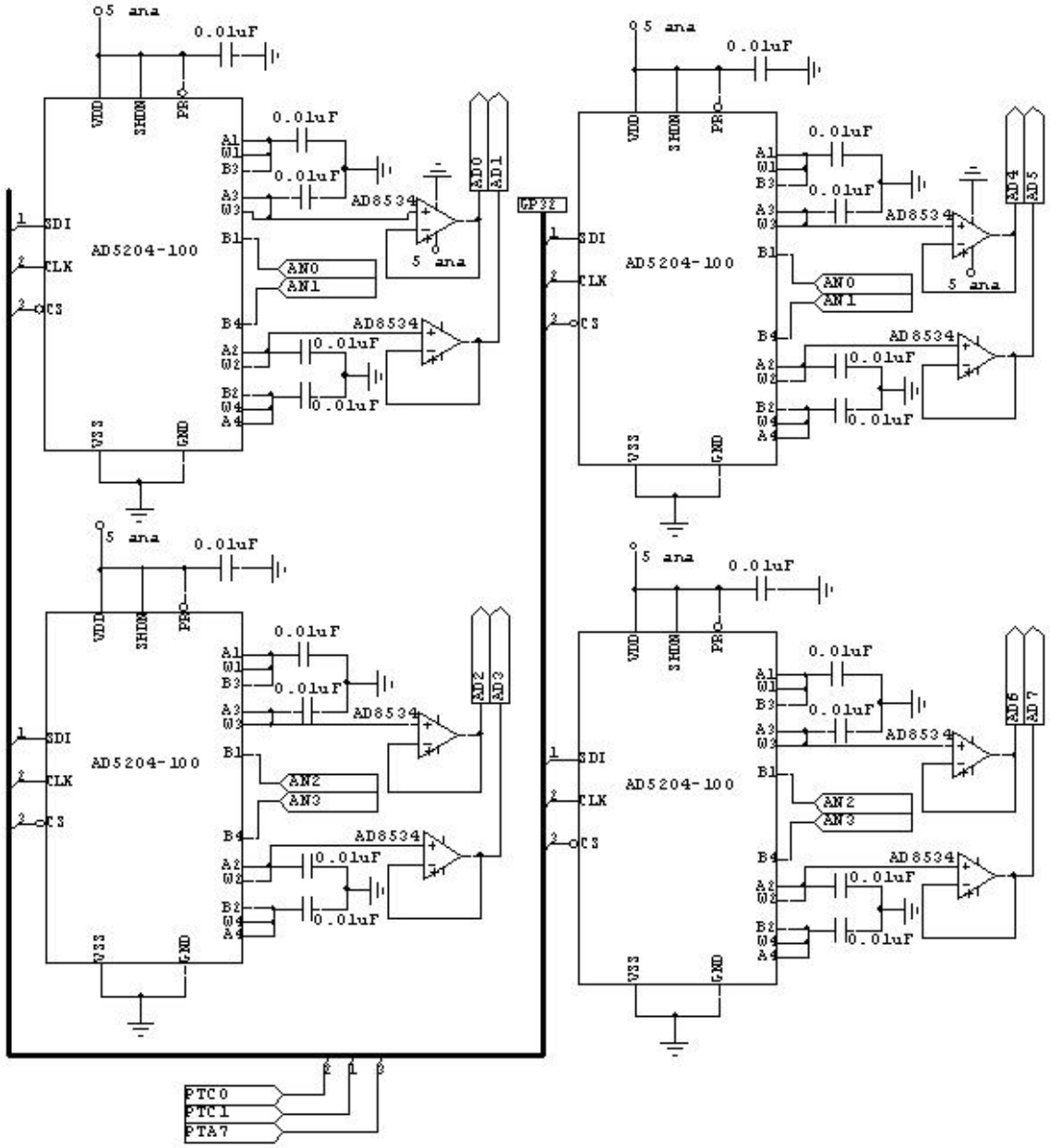
La tabla 12 describe el código digital, la resistencia y la frecuencia de corte para 3 valores de resietencia diferente y en la figura 35 se muestra la conexión de los filtros de antisolapamiento con el microcontrolador y el circuito de acondicionamiento de la señal.

TABLA 12. Relación entre el código digital, la resistencia y la frecuencia de corte del potenciometro digital

CODIGO DIGITAL	RESISTENCIA	FRECUENCIA DE CORTE
0_{10}	$100\times10^3\Omega$	60,933 Hz
128_{10}	$50\times10^3\Omega$	119,66 Hz
255_{10}	435,6Ω	13,86 Hz

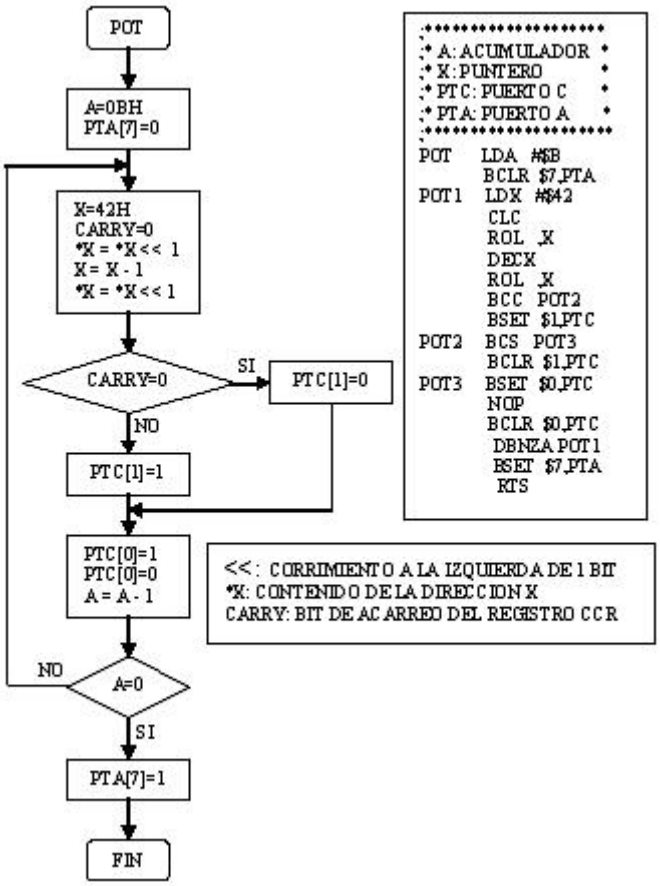
Para controlar la frecuencia de corte de los filtros antisolapamiento se necesita cambiar la resistencia o el condensador; en cuyo caso se cambia la resistencia de cada uno de los 16 potenciometros que se requieren para construir los 8 filtros antisolapamiento de los canales de adquisición de señales analógicas. La figura 35 muestra la conexión serial necesaria para transmitir el valor de resistencia deseado desde el microcontrolador hacia los potenciometros digitales y la figura 36 muestra un flujograma que implementa la comunicación serial entre el microntrooador 68HC908GP32 y los 8 potenciometros digitales.

FIGURA 35. Circuito de filtros antisolapamiento.



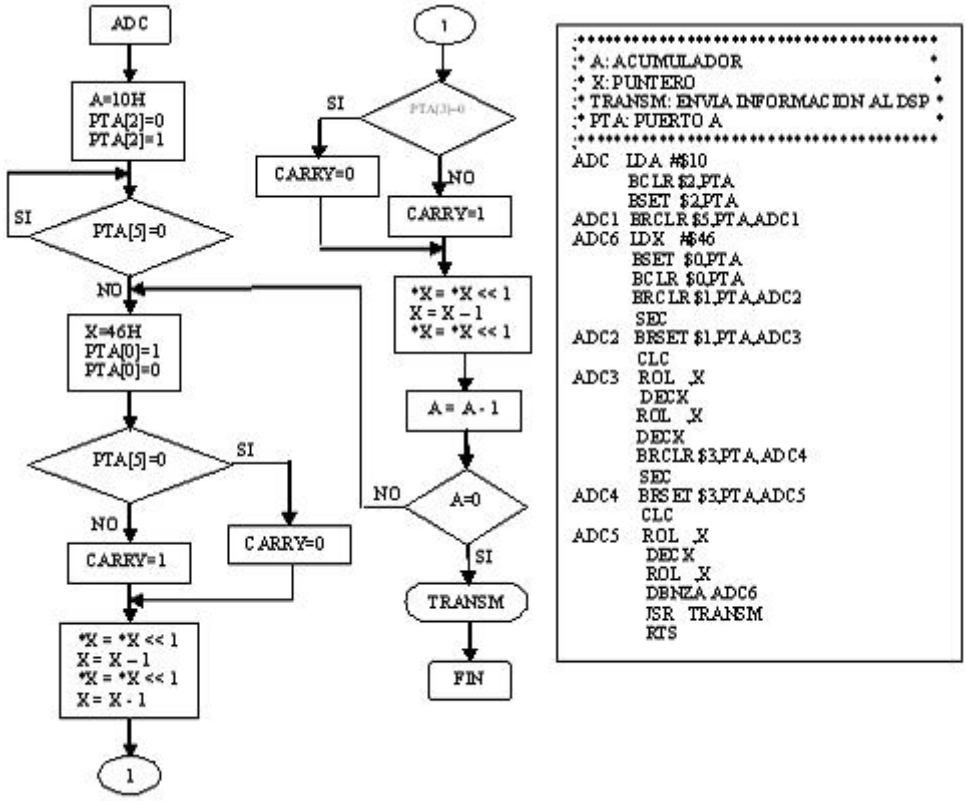
Fuente: El autor. Elaborado con el *software* CIRCUITMAKER.

FIGURA 36. Subrutina para cambiar la resistencia de los potenciometros digitales AD5204.



Fuente: El autor.

FIGURA 37. Subrutina para atender 2 AD974 simultáneamente.

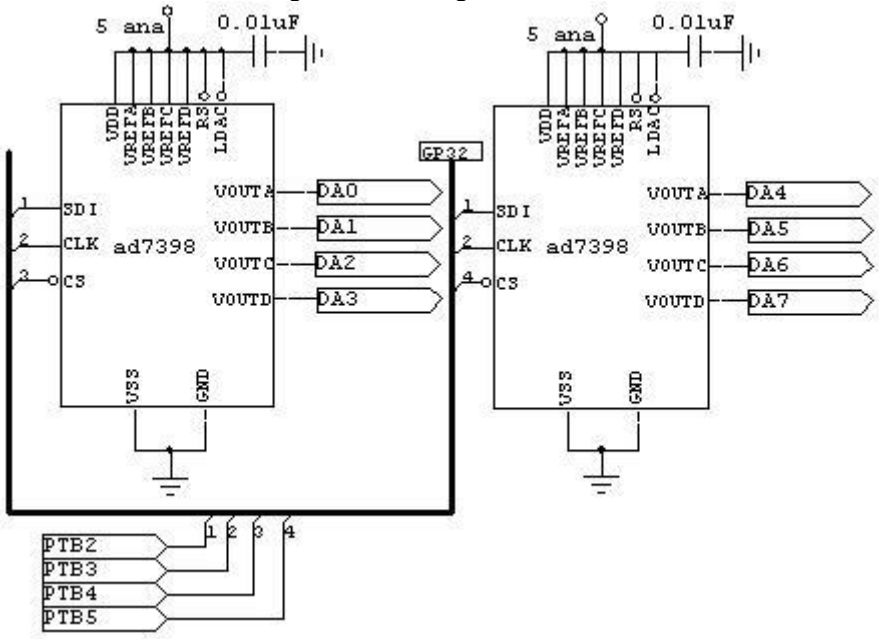


Fuente: El autor.

3.2. CANAL DE GENERACIÓN DE SEÑALES ANALÓGICAS.

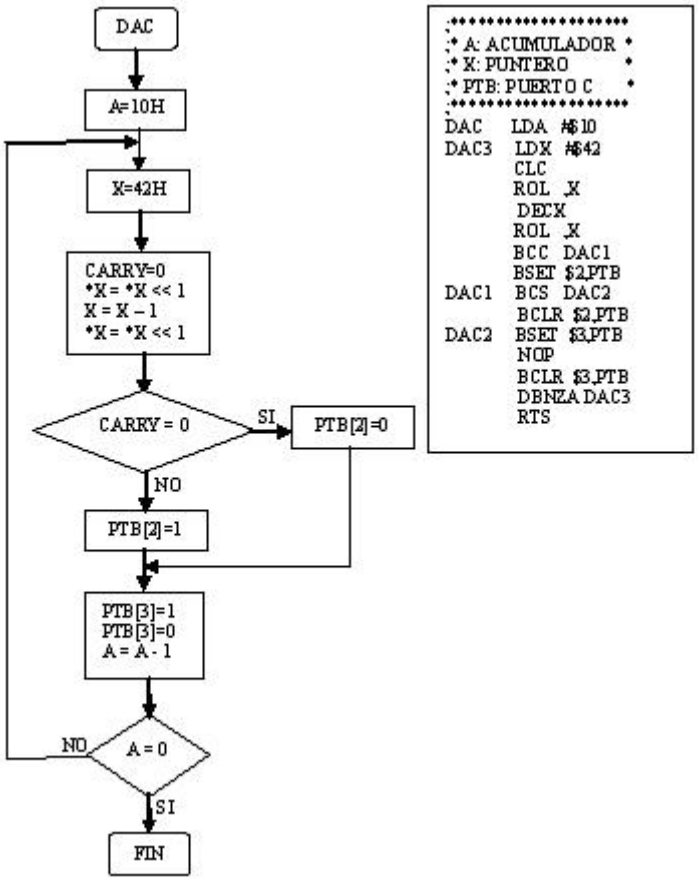
El componente básico para la generación de señales analógicas es el convertidor digital-analógico AD7398 de Analog Devices. Este dispositivo tiene cuatro convertidores DA cada uno de ellos con una resolución de 12 *bits* y capaz de generar señales en el rango de 0 a 5 V. Para completar 8 canales se utilizan 2 de estos conversores. En la figura 38 se muestra la conexión de los conversores con el microcontrolador y en la figura 39 se muestra el diagrama de flujo de la subrutina de que maneja los 8 canales de conversión digital-analógica.

FIGURA 38. Conversores digitales-analógicos AD7398.



Fuente: El autor. Elaborado con el *software* CIRCUITMAKER.

FIGURA 39. Flujograma para cambiar la tensión de un canal analógico de salida.



Fuente: El autor.

3.3. CANAL DE SEÑALES DIGITALES DE ENTRADA Y SALIDA.

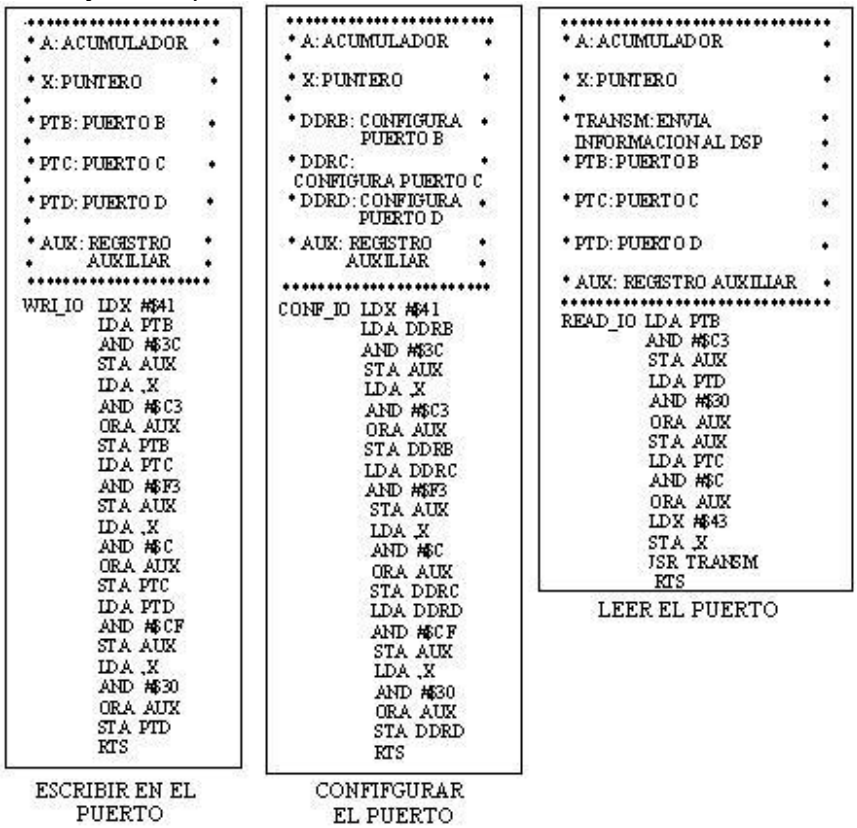
El canal de señales digitales de 0 a 5 V tiene tres partes fundamentales: lectura, escritura y configuración de los pines del puerto digital. Para implementar el canal de 8 bits de señales digitales se utilizan 8 pines del microcontrolador 68HC908GP32 de motorola. Las funciones implementadas para los pines digitales son: configuración, lectura y escritura. La función de configuración se utiliza para seleccionar entre dos posibles funcionamientos del pin: entrada o salida; la función de escritura coloca el pin configurado como salida en estado alto o bajo y la función de lectura almacena en un registro del microcontrolador el estado actual de cada pin del puerto digital. En la figura 40 se relaciona el pin del microcontrolador con el bit correspondiente del puerto digital de 8 *bits*. En la figura 41 se muestran las subrutinas en lenguaje ensamblador para configurar, leer y escribir en el puerto digital.

FIGURA 40. Equivalencia entre los pínes del microcontrolador y los *bits* del puerto digital de la tarjeta adaptable.

PT0	PT1	PT2	PT3	PT4	PT5	PT6	PT7	PTO DIGITAL
PTB0	PTB1	PTC2	PTC3	PTD4	PTD5	PTB6	PTB7	68HC908GP32

Fuente: El autor.

FIGURA 41. Subrutinas para lectura, configuración y escritura de los puertos digitales de la tarjeta adaptable.



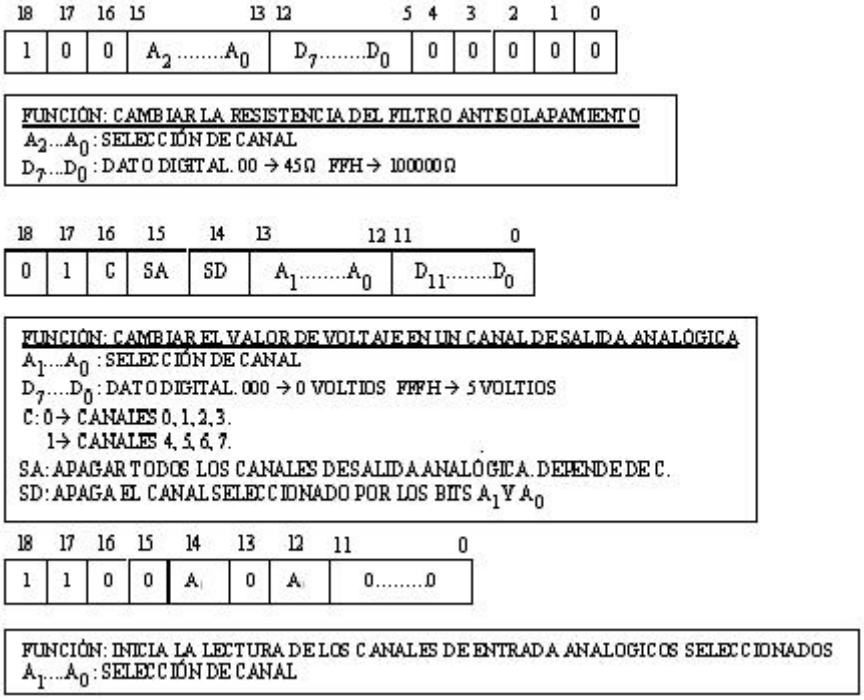
Fuente: El autor.

3.4. UNIDAD CENTRAL DE TRANSMISIÓN Y RECEPCIÓN DE DATOS DESDE y HACIA EL DSP.

El dispositivo central recibe la información de las banderas configurables del ADSP2181 y procesa la misma para activar o desactivar los puertos digitales, leer dos datos de los canales analógicos, enviar un dato hacia una de las salidas analógicas que posee la tarjeta o cambiar el valor de la resistencia del filtro antisolapamiento. Esta tarea es llevada a cabo por un microcontrolador de motorola cuya referencia es 68HC908GP32. La transmisión y recepeción de datos desde y hacia el DSP se realiza utilizando las banderas configurables del mismo; la cantidad de *bits* enviados es de 19 bits y se transmiten desde el microcontrolador hacia el DSP 32 *bits*. El receptor recibe una señal en el pin de interrupción externa, lo cual detiene el programa en curso para atender la recepción de los 19 *bits* del dato serial generado por el DSP en el pin PF2 en sincronismo con un reloj generado por el pin PF3.

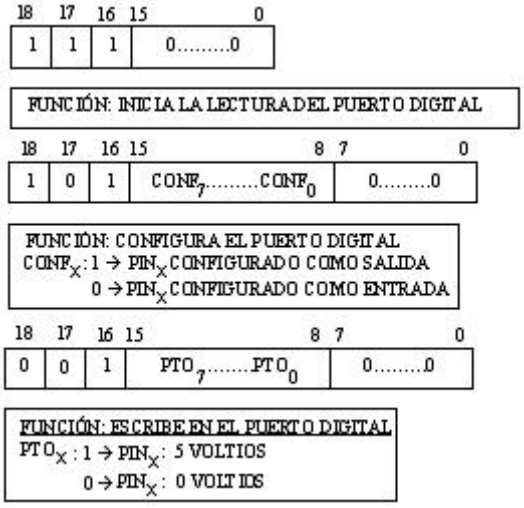
Las figuras 42 y 43 muestran las diferentes funciones que cumplen los 19 *bits* que se transmiten desde el DSP hacia la tarjeta adaptable.

FIGURA 42. Funciones de los 19 *bits* de transmisión. Parte A.



Fuente: El autor.

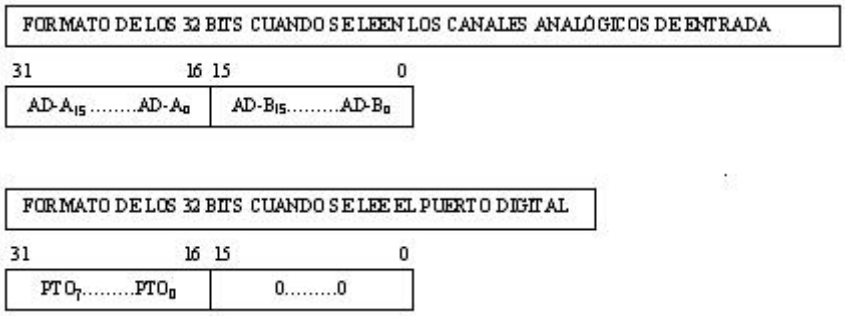
FIGURA 43. Funciones de los 19 *bits* de transmisión. Parte B.



Fuente: El autor.

Los 32 *bits* recibidos por el DSP a través de las banderas configurables, corresponden a las lecturas de los dos datos de 16 *bits* de los convertidores analógicos-digitales de los canales de entrada analógica de la tarjeta o a los pines del puerto digital configurados como entradas. Lo anterior se puede observar en la figura 44.

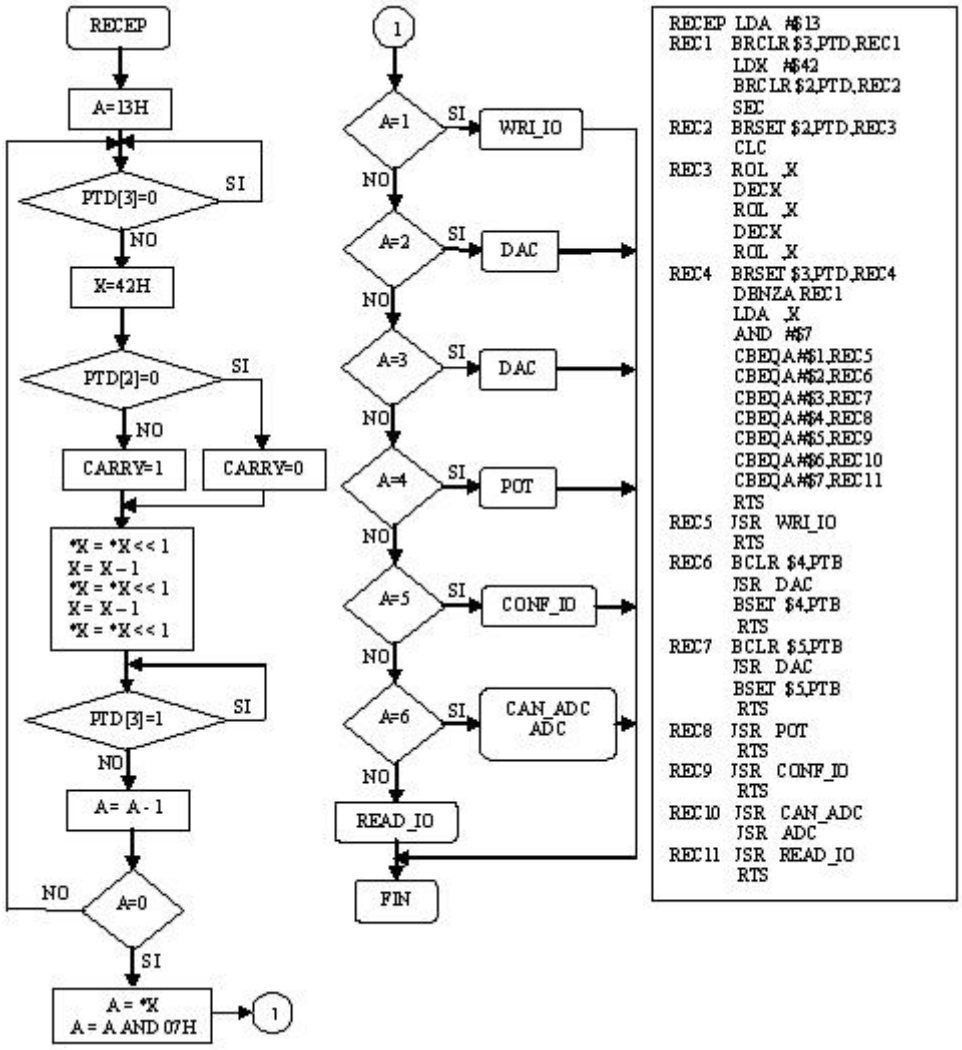
FIGURA 44. Formato de los 32 *bits* cuando se leen los canales analógicos de entrada o cuando se lee el puerto digital.



Fuente: El autor.

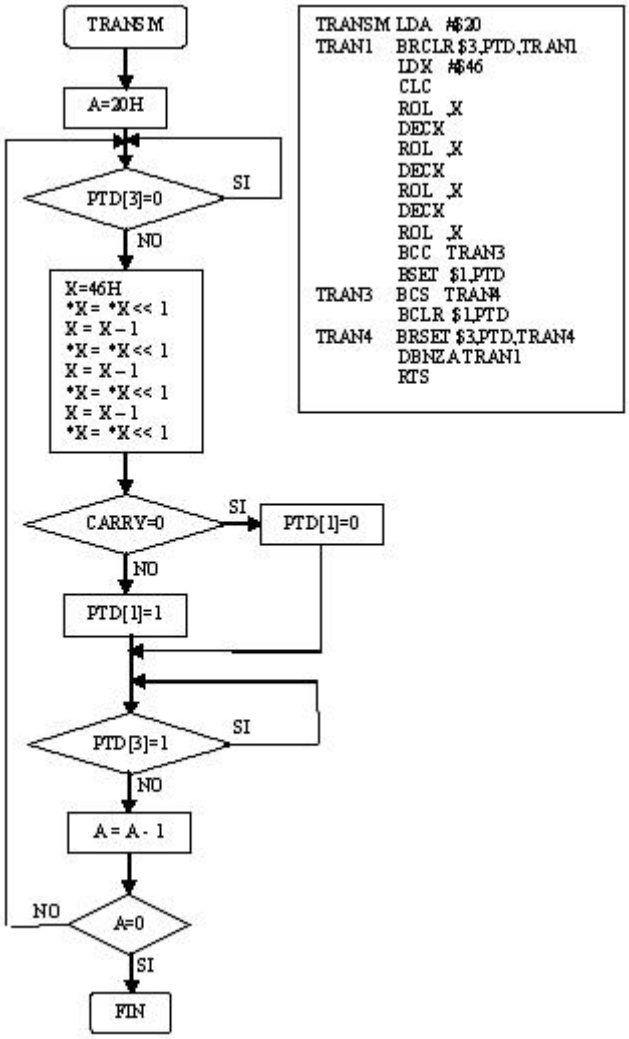
Las subrutinas que emplea el núcleo central de la tarjeta para transmitir y recibir información desde y hacia el DSP se observan en las figuras 45 y 46. En la figura 47 se observa el núcleo central de la tarjeta.

FIGURA 45. Subrutina que recibe 19 *bits* de transmisión serial desde el DSP.



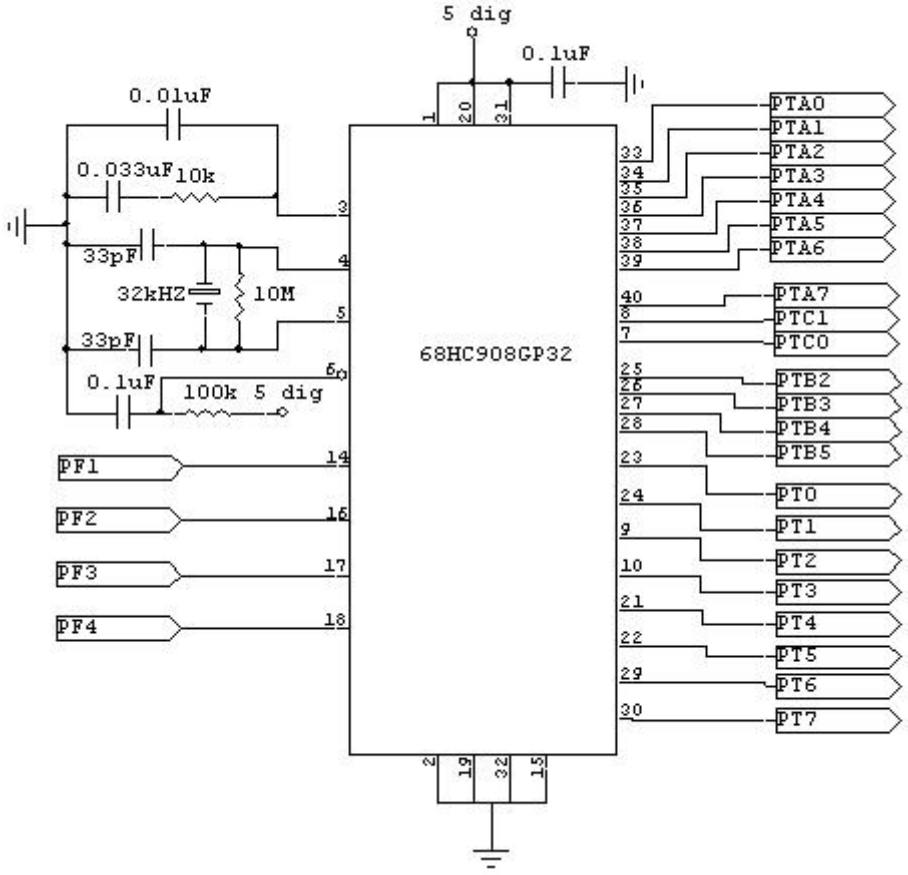
Fuente: El autor.

FIGURA 46. Subrutina que transmite 32 *bits* seriales desde el microcontrolador hacia el DSP.



Fuente: El autor.

FIGURA 47. Núcleo central de la tarjeta adaptable.

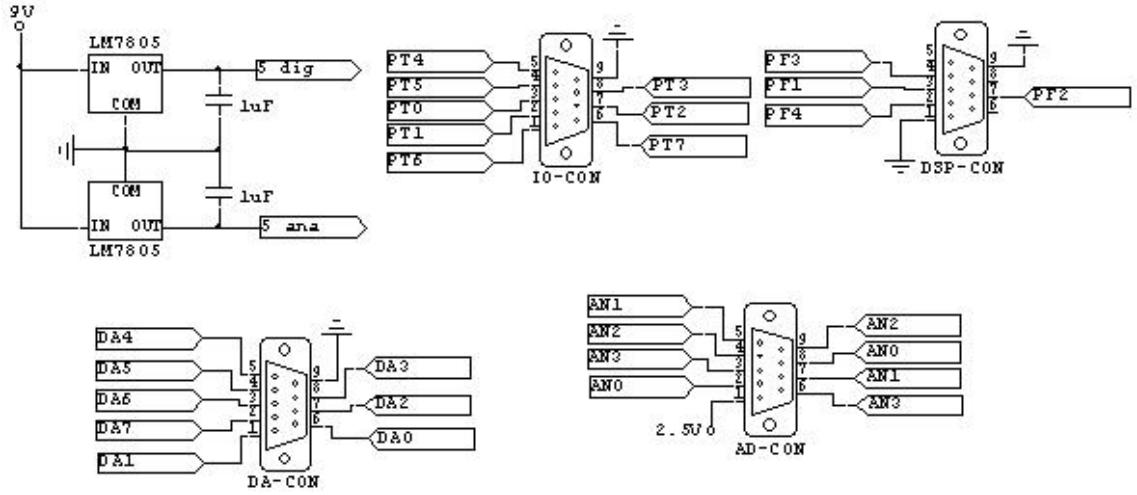


Fuente: El autor. Elaborado con el *software* CIRCUITMAKER.

3.5. CONECTORES Y FUENTES DE ALIMENTACIÓN DE LA TARJETA ADAPTABLE.

La tarjeta adaptable cuenta con cuatro conectores, una fuente de alimentación que suministra energía a dos reguladores de 5 V, uno de ellos alimenta los circuitos analógicos y el otro alimenta los circuitos digitales. El primer conector comunica el DSP con la tarjeta adaptable; los otros tres se utilizan para comunicar los puertos digitales, las salidas analógicas y las entradas analógicas con un circuito externo construido en un *protoboard*. En la figura 48 se observan la fuente de alimentación y los 4 conectores de la tarjeta adaptable.

FIGURA 48. Fuentes de alimentación y conectores.



Fuente: El autor. Elaborado con el *software* CIRCUITMAKER.

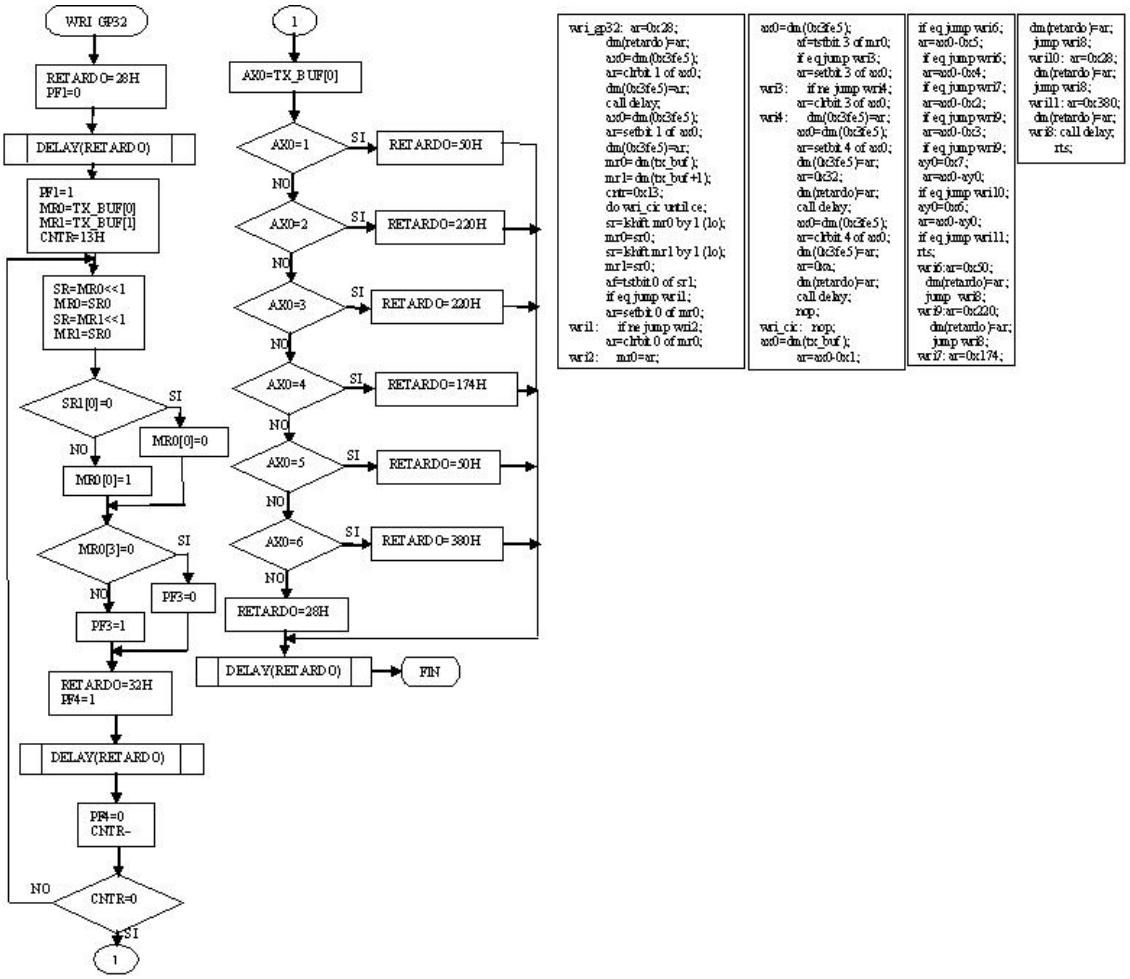
3.6. SUBRUTINAS PARA MANEJAR LA TARJETA ADAPTABLE DESDE LA TARJETA EZKIT2181.

Hasta este punto se ha expuesto lo relacionado con el *software* y el *hardware* de la tarjeta adaptable. El texto que sigue a continuación explica las partes fundamentales del *software* que maneja la tarjeta adaptable desde la tarjeta EZKIT2181. La programación consta de dos subrutinas principales: envio y recepción de datos desde y hacia la tarjeta adaptable; leer, escribir y configurar los puertos digitales; leer dos canales de analógicos de entrada de 16 *bits* cada uno; escribir 12 *bits* en el canal de salida analógica seleccionado y cambiar el valor de la frecuencia de corte de los filtros antisolapamiento de segundo orden.

3.6.1. Subrutinas que manejan la transmisión y recepción de datos seriales desde y hacia la tarjeta adaptable.

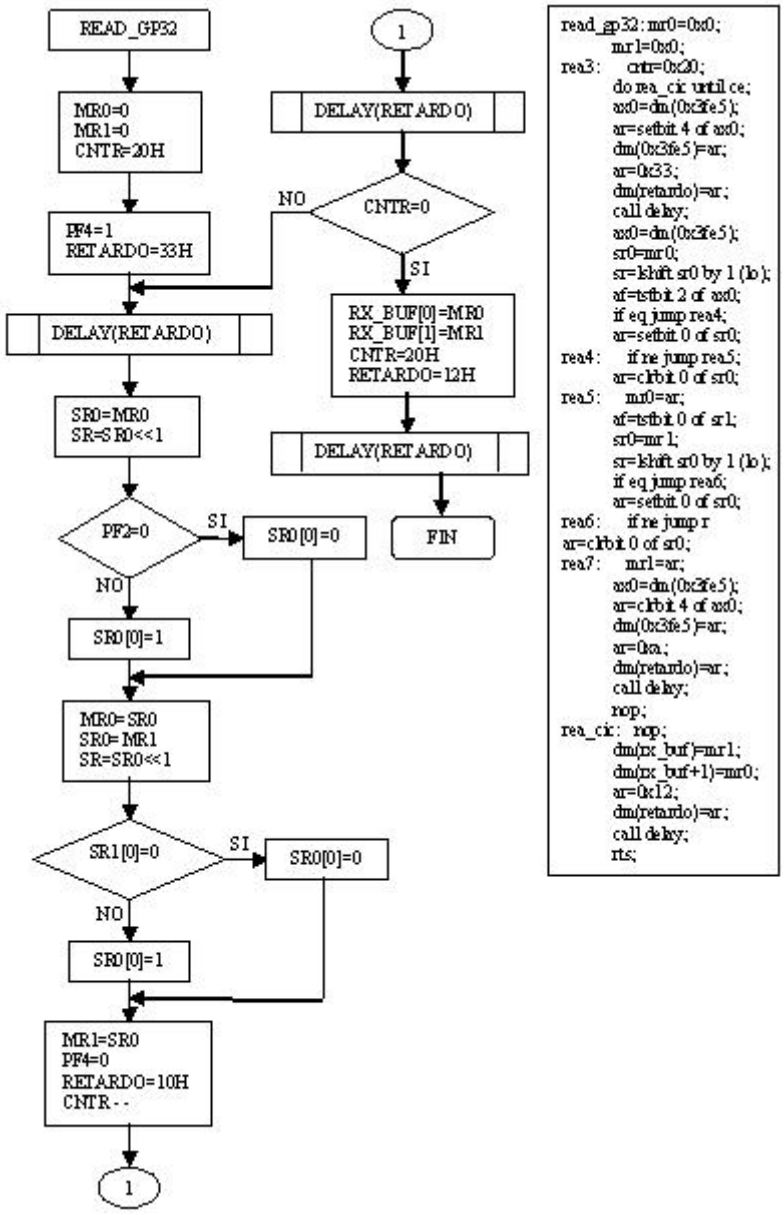
Estas dos subrutinas se encargan de enviar y recibir datos desde la tarjeta adaptable. La tarjeta EZKIT2181 envia 19 *bits* seriales; esta información se encuentra en un *buffer* de datos denominado tx_buf[2]. Cada una de estas variables posee una longitud de 16 bits. La tarjeta adaptable, cuando se solicita, envia 32 *bits* de información que se almacenan en el *buffer* de datos llamado rx_buf[2]. Las figuras 49 y 50 muestran los diagramas de flujo de las subrutinas wri_gp32 y read_gp32 que transmiten y reciben información desde y hacia la tarjeta adaptable. La subrutina retardo se utiliza para sicronizar el DSP (32 MHz) con el microcontrolador (8,2 MHz). La cantidad de tiempo que consume la subrutina se controla con la variable *delay*; si *delay* es igual a uno el tiempo consumido es igual a un ciclo de máquina del microcontrolador.

FIGURA 49. Subrutina para enviar 19 *bits* hacia la tarjeta adaptable.



Fuente: El autor.

FIGURA 50. Subrutina para recibir 32 *bits* desde la tarjeta adaptable.

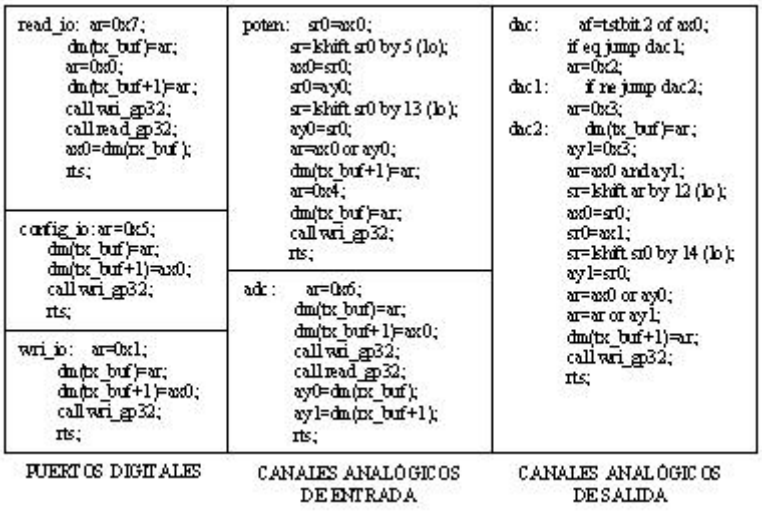


Fuente: El autor.

3.6.2. Subrutinas para comunicarse con los elementos de la tarjeta adaptable.

Dentro de este grupo de subrutinas están: read_io, wri_io y config_io se encargan de leer, escribir y configurar, respectivamente, los puertos de entrada y salida digital de la tarjeta adaptable; adc realiza la lectura de dos canales de entrada analógica; poten cambia la resistencia que controla la frecuencia de corte de los filtros antisolapamiento de los canales de entrada analógica; dac cambia la tensión en uno de los canales de salida analógica. En la figura 51 se observa el código de estas subrutinas.

FIGURA 51. Subrutinas para comunicarse con los periféricos de la tarjeta adaptable.



Fuente: El autor.

4. PROGRAMACIÓN DE LOS ALGORITMOS FFT, GOERTZEL Y CHIRP EN EL DSP ADSP2181.

Este capítulo presenta los diagramas de flujo, el código y la explicación de los programas elaborados para realizar los algoritmos de FFT, Goertzel y CHIRP en un procesador ADSP2181 de ANALOG DEVICES. Los programas de los algoritmos FFT en base 2 y en base 4 se programaron en lenguaje C y Ensamblador para realizar comparaciones de tiempos de ejecución; los demás algoritmos se programaron solo en lenguaje C. El compilador utilizado en ambos casos es el VISUALDSP++. Este ambiente de programación tiene la capacidad de compilar lenguaje C, C++ y Ensambaldor de la familia ADSP21xx de los procesadores de ANALOG DEVICES.

4.1. ALGORITMO Y PROGRAMAS PARA LA TRANSFORMADA RÁPIDA DE FOURIER EN BASE 2.

El programa del algoritmo de FFT en base 2 esta compuesto de tres etapas: la primera etapa define la primera posición, la clase, el tamaño y el incremento de los punteros que se van a utilizar en la aplicación; en este caso se utilizan dos punteros circulares que apuntan a los vectores de coeficientes reales e imaginarios y cuatro punteros adicionales para acceder a los vectores de señal real e imaginaria. La segunda etapa ejecuta el algoritmo como tal y la tercera etapa utiliza la inversión de bits de las direcciones del vector de señal real e imaginaria para ubicar el resultado en las posiciones correctas. En la figura 52 se muestra un flujograma con el procedimiento para evaluar la FFT en base 2.

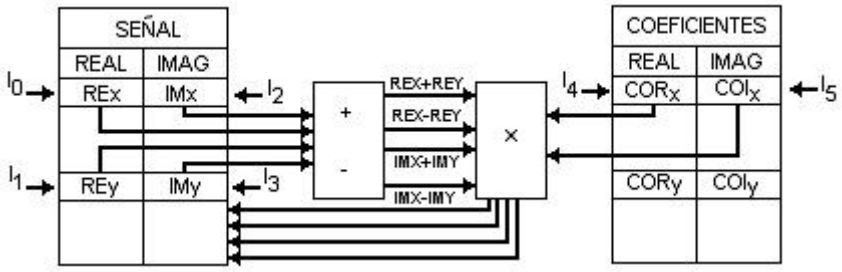
FIGURA 52. Flujograma del programa de la FFT en base 2.

Fuente: El autor.

Los cuatro punteros que apuntan a los vectores de señal real e imaginaria se utilizan asi: los punteros I_0 e I_1 apuntan al dato de entrada de la mariposa con la dirección menor; el primero de ellos se utiliza para extraer la parte real y el segundo la parte imaginaria del dato de entrada a la mariposa. Los punteros I_2 e I_3 direccionan el elemento de la mariposa de dirección mayor; I_2 para la parte real e I_3 para la parte imaginaria del mismo. Los vectores I_4 e I_5 apuntan al vector de coeficientes reales e imaginarios; deben tener la particularidad de autoregularse. Esto quiere decir que cuando detectan el final del vector al cual están apuntando deben reposicionarse

automáticamente. En la figura 53 se observa un ejemplo de una mariposa de la FFT en base 2 con los punteros apuntando a los respectivos elementos.

FIGURA 53. Mariposa de la FFT en base 2 ejecutada por un procesador DSP.



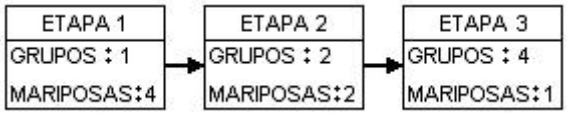
Fuente: El autor.

El algoritmo de la FFT en base 2, segunda etapa de la figura 52, se puede seguir facilmente si se tiene en cuenta la figura 4, FFT en base 2 para 8 muestras. En esta figura se observan tres aspectos fundamentales: etapas, grupos de mariposas por etapa y número de mariposas por grupo. El número de etapas del algoritmo se obtiene mediante la ecuación (71).

$$N = 2^{etapas}$$
$$etapas = \log_2 N \qquad (71)$$

Donde N representa el número de muestras de la señal discreta de la cual se obtendrá su espectro. Para el ejemplo en cuestión N es igual a 8, es decir se requieren 3 etapas para su realización. Los grupos por etapa comienzan con un valor de 1 y se van incrementando a medida que las etapas se van ejecutando; para obtener el número de grupos de la siguiente etapa se desplaza un *bit* a la izquierda el número de grupos de la etapa anterior, para el caso de la FFT en base 2 de 8 muestras el siguiente sería 2 y el de la etapa final 4. Inicialmente las mariposas por grupo son 4 y para obtener el valor para las etapas posteriores basta con realizar un desplazamiento a la derecha del valor de la etapa previa; para el ejemplo, la sucesión que se lográ es 4, 2 y 1. En la figura 54 se muestra un diagrama del algoritmo de la FFT en base 2 desde el punto de vista de las etapas, los grupos y las mariposas por grupo.

FIGURA 54. Progresión de un algoritmo FFT en base 2 para N igual a 8 muestras.



Fuente: El autor.

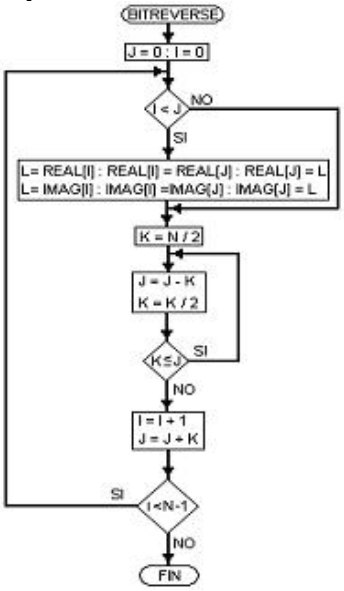
Un aspecto importante a tener en cuenta cada vez que se utilizan los procesadores de coma fija, es el desbordamiento. Para evitar este fenómeno en el procesamiento de la FFT en base 2, antes de procesar una nueva etapa se debe dividir por dos todo el vector de señal antes de procesar la siguiente. En la figura 55 se muestra este procedimiento.

FIGURA 55. Procedimiento para evitar el desbordamiento en un algoritmo FFT en base 2.

Fuente: El autor.

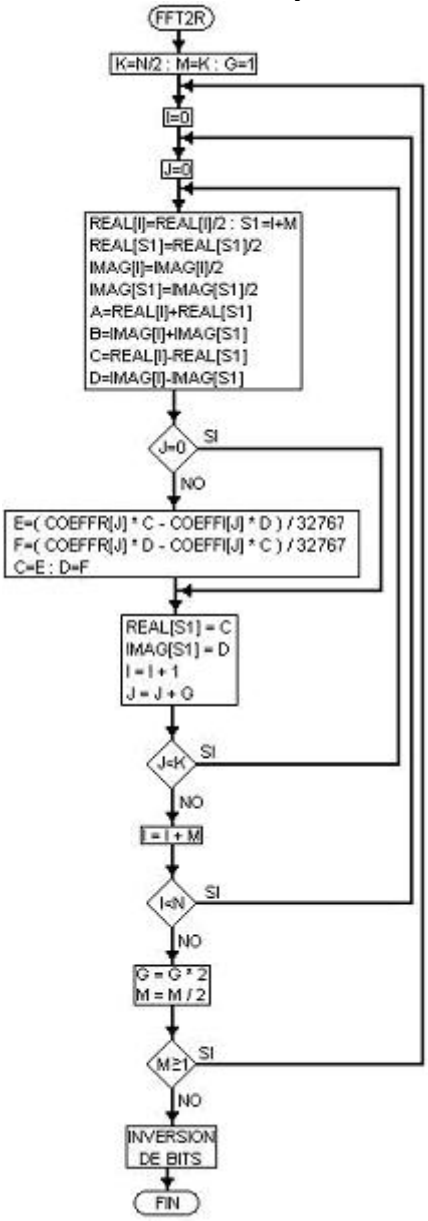
La última parte del procedimiento para ejecutar el algoritmo de la FFT en base 2 en un DSP consiste en la inversión de los *bits* de las direcciones del vector de resultados para obtener el espectro de la señal en las posiciones correctas. La inversión de los *bits* de la parte real e imaginaria del vector de resultados se lleva a cabo mediante la subrutina BITREVERSE en ella se tiene en cuenta que los resultados deben ser almacenados en los vectores originales con el objetivo de ahorrar espacio de memoria. Las figuras 56 y 57 muestran el diagrama de flujo de las subrutinas BITREVERSE y FFT2R que permiten la inversión de los bits del vector de resultados y obtener la DFT de una señal de N muestras con N un múltiplo de 2. Las figuras 58 y 59 muestran el código en lenguaje C y Ensamblador de las subrutinas FFT2R y BITREVERSE respectivamente.

FIGURA 56. Diagrama de flujo de la subrutina BITEREVERSE.



Fuente: El autor.

FIGURA 57. Algoritmo de la FFT en base 2. Ejecutable en un procesador DSP.



Fuente: El autor.

FIGURA 58. Códigos de las subrutinas FFT en base 2 e inversión de los *bits* de la dirección del vector de salida en lenguaje Ensamblador del ADSP2181.

```
/*********218x FFT RADIX 2 Example**********/
#define N 16
#define BASE 4
.section/data data1;

.VAR REAL [N]="real.dat";
.VAR IMAG [N]="imag.dat";
.VAR AUX1;
.VAR AUX2;
.VAR AUX3;

.section/pm                pm_da;
.VAR COEFFR [N]="coeffr.dat";
.VAR COEFFI [N]="coeffi.dat";


.section/pm interrupts;

__reset: JUMP start; NOP; NOP; NOP;
      RTI; NOP; NOP; NOP;
      RTI; NOP; NOP; NOP;
      RTI; NOP; NOP; NOP;
      RTI; NOP; NOP; NOP;
      RTI; NOP; NOP; NOP;
      RTI; NOP; NOP; NOP;
      RTI; NOP; NOP; NOP;
      RTI; NOP; NOP; NOP;
      RTI; NOP; NOP; NOP;
      RTI; NOP; NOP; NOP;
      RTI; NOP; NOP; NOP;

.section/pm program;

start: L0=0;L1=0; L2=0;L3=0;L4=N/2;L5=N/2;
     M1=N/2;M0=0;M2=1;M4=1;
     I4=COEFFR; I5=COEFFI;
    CNTR=BASE;
 DO ETAP UNTIL CE;
     I0=REAL; I1=IMAG; I2=I0;I3=I1;
     MODIFY(I2,M1);MODIFY(I3,M1);
     CNTR=M4;
 DO GRUP UNTIL CE;
     CNTR=M1;
 DO MARIP UNTIL CE;
     SI=DM(I0,M0);SR=ASHIFT SI BY -1 (HI);
     AX0=SR1;SI=DM(I2,M0);
     SR=ASHIFT SI BY -1 (HI);AY0=SR1;
     SI=DM(I1,M0);SR=ASHIFT SI BY -1 (HI);
     AX1=SR1;SI=DM(I3,M0);
     SR=ASHIFT SI BY -1 (HI);
     AR=AX0+AY0,AY1=SR1;
     DM(I0,M2)=AR,AR=AX0-AY0;
     DM(I2,M0)=AR,AR=AX1+AY1;
     DM(I1,M2)=AR,AR=AX1-AY1;
     DM(I3,M0)=AR;AY0=COEFFR;AX0=I4;
     AR=AY0-AX0;
     IF NE JUMP MULTI;
     MODIFY(I5,M4);MODIFY(I4,M4);
     MODIFY(I2,M2);MODIFY(I3,M2);
     JUMP MARIP;
MULTI: MX0=DM(I2,M0);MY0=PM(I4,M4);
     MX1=DM(I3,M0);MY1=PM(I5,M4);
     MR=MX0*MY0 (SS);MR=MR-MX1*MY1 (RND);
     DM(I2,M2)=MR1,MR=MX0*MY1 (SS);
     MR=MR+MX1*MY0 (RND);DM(I3,M2)=MR1;
MARIP: NOP;
          MODIFY(I0,M1); MODIFY(I1,M1);
          MODIFY(I2,M1);
GRUP:    MODIFY(I3,M1);
          SI=M4; SR=LSHIFT SI BY 1 (HI); M4=SR1;
          SI=M1;SR=LSHIFT SI BY -1 (HI);
ETAP:    M1=SR1;
     AX0=REAL;AR=PASS AX0;
     IF NE CALL INV_DIR;
     I0=AR;AX0=IMAG;
     AR=PASS AX0;
     IF NE CALL INV_DIR;
     I1=AR; CNTR=N;
     I0=REAL;I4=REAL_REV;
     I5=IMAG_REV;L4=0;L5=0;
     M1=NBR;M4=1;
     ENA BIT_REV;

IBIT2R: MR0=0;MR1=0;SR0=0;L0=0;L1=0;
          I0=REAL;I2=IMAG;M1=1;M0=0;
 REV4:  AY0=MR1;AF=MR0-AY0;
          IF GE JUMP REV1;
        M2=MR1;I1=REAL;I3=IMAG;MODIFY(I1,M2);
        MODIFY(I3,M2);
        AX0=DM(I0,M0);AX1=DM(I1,M0);
        DM(I0,M0)=AX1;DM(I1,M0)=AX0;
        AX0=DM(I2,M0);AX1=DM(I3,M0);
        DM(I2,M0)=AX1;DM(I3,M0)=AX0;
REV1:  MODIFY(I0,M1);MODIFY(I2,M1);
     SR0=N/2;
REV3:  AY0=MR1;AF=SR0-AY0;
     IF GT JUMP REV2;
     AY0=SR0;
     AR=MR1-AY0;
     MR1=AR;
     SR=LSHIFT SR0 BY -1 (LO);
     JUMP REV3;
REV2:  AY0=SR0;
     AR=MR1+AY0;
     MR1=AR;
     AR=MR0+1;MR0=AR;
     AY0=N-2;
     AF=AR-AY0;
     IF NE JUMP REV4;
     RTS;
```

Fuente: El autor.

FIGURA 59. Subrutinas de la FFT en base 2 e inversión de los *bits* de la dirección del vector de salida en lenguaje ANSI C.

```
#define N 16

int real [N] = {
        #include "real.dat"
        };
int imag [N] = {
         #include "imag.dat"
         };
int pm coeffr [N] = {
         #include "coeffr.dat"
                                };
int pm coeffi [N] = {
         #include "coeffi.dat"
                                };
void fft2r (void);
int ibitr (int);
void bitreverse (void);

main()
{
 fft2r();
return 0;
}

void fft2r(void)
{
 long c,d,e,f;
 int i,jk,gm;
 k=N>>1;
 m=k; g=1;
 while (m>=1)
 {
 i=0;
 while(i<N)
 {
  j=0;
  while(j<k)
  {
  s1=i+m;
   real[i]/=2;real[s1]/=2;
   imag[i]/=2;imag[s1]/=2;
    c=real[i]-real[s1];d=imag[i]-imag[s1];
   real[i]=real[i]+real[s1];imag[i]=imag[i]+imag[s1];
   if (j)
   {
    e=(coeffr[j]*c-coeffi[j]*d)>>15;
    f=(coeffr[j]*d+coeffi[j]*c)>>15;
    c=e;d=f;
   }
   real[s1]=c;imag[s1]=d;
   i=i+1;j=j+g;
  }
   i=i+m;
 }
 g=g<<1;m=m>>1;
 }
 bitreverse();
}
```

```
void bitreverse(void)
{
 int i,jk,tmp;
 j=0;
 for (i=0;i<N-1;i++)
 {
 if(i<j)
 {
  t=real[i];real[i]=real[j];real[j]=t;
  t=imag[i];imag[i]=imag[j];imag[j]=t;
 }
 k=N/2;
 while(k<=j)
 {
  j-=k;
  k/=2;
 }
 j+=k;
}
}
```

Fuente: El autor.

Los dos programas pueden compararse computacionalmente teniendo en cuenta los siguientes parámetros: tamaño en *bytes* del archivo ejecutable generado por el compilador, número de ciclos de máquina empleados y el tiempo empleado por el procesador para ejecutar la subrutina. La comparación se realiza con un procesador de 16 *bits* y una frecuencia de reloj de 32 MHz; la señal escogida fue una onda cuadrada bipolar de 500 mV pico. La figura 60 muestra la comparación computacional.

FIGURA 60. Comparación computacional del algoritmo de la FFT en base 2 ejecutado en lenguaje Ensamblador y lenguaje C.

	ENSAMBLADOR TAMAÑO: 1953 "bytes"		LENGUAJE C TAMAÑO: 6945 "bytes"	
MUESTRAS (N)	CICLOS DE MÁQUINA	TIEMPO (µS)	CICLOS DE MÁQUINA	TIEMPO (mS)
16	1352	42,250	8785	0,2745
32	3100	96,875	22316	0,6973
64	7044	220,12	54601	1,7062
128	15668	489,62	128922	4,0288
256	34622	1081,9	298038	9,3136
512	75611	2362,8	660437	20,638
1024	151799	9487,4375	1432814	44,775

Fuente: El autor.

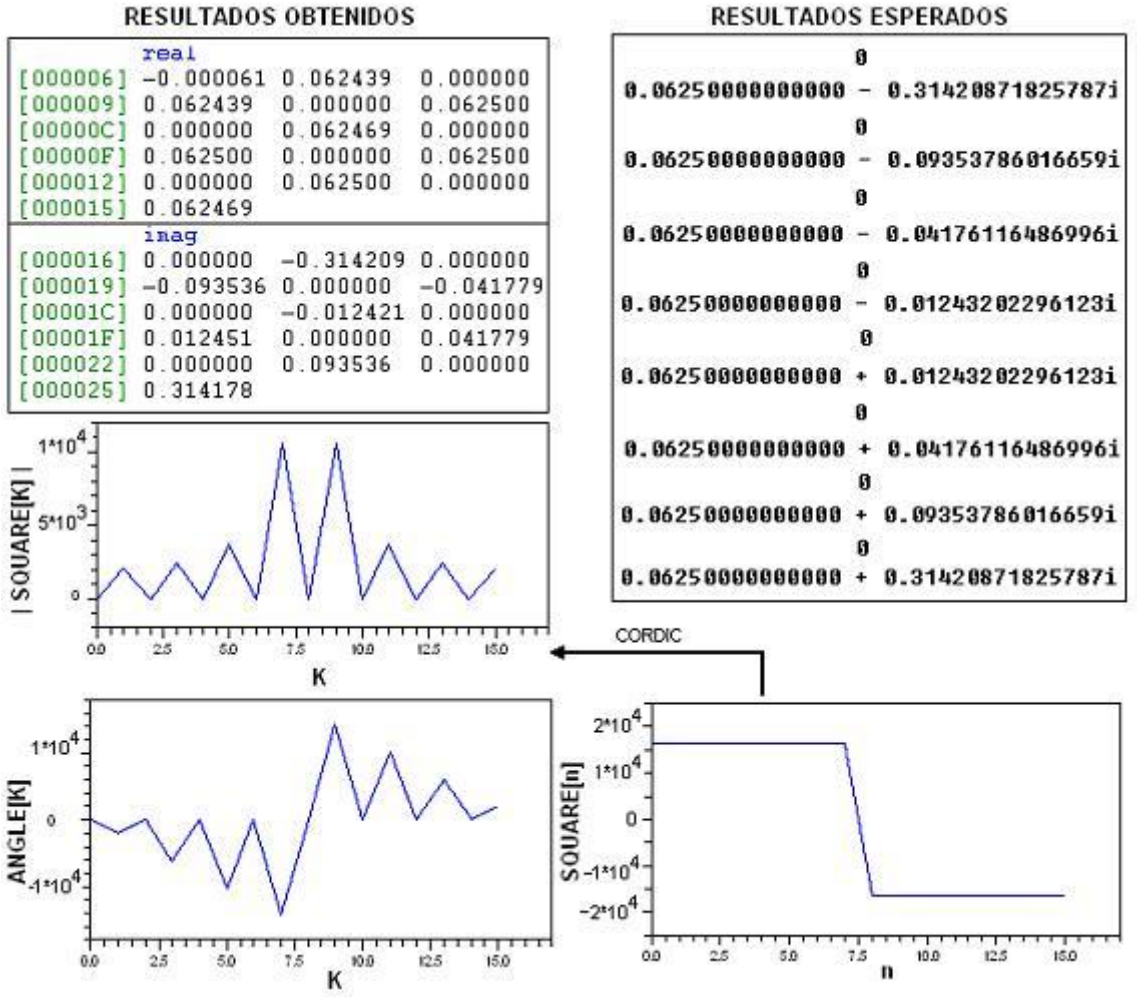
De la tabla de la figura 60 se puede concluir sobre dos aspectos importantes del desempeño computacional (ciclos de máquina) y el espacio de memoria que ocupa cada programa en la memoria del DSP; la cantidad de ciclos de máquina requeridos por los programas elaborados en lenguaje Ensamblador es mucho menor que los empleados por los programas elaborados en lenguaje C. El lenguaje C utilizado no permite explotar de manera directa algunas de las ventajas del procesador DSP como los vectores circulares y los apuntadores automáticos para el post-incremento y post-decremento de un vector de datos. Lo anterior incide notoriamente en el desempeño, porque el *software* debe encargarse por si mismo a través de variables auxiliares de realizar esta labor que el procesador DSP puede realizar en forma automática; una solución a este problema sería utilizar la instrucción asm() del lenguaje C y definir los vectores circulares a través de esta herramienta. (Nota: esta opción no se tuvo en cuenta porque desvirtuaría el propósito de la evaluación como tal). A medida que el tamaño de la señal de entrada aumenta, la diferencia computacional se hace más notoria debido a la cantidad de operaciones adicionales que se deben implementar para emular en lenguaje C los elementos que el DSP brinda de manera automática. Una gran cantidad de operaciones que puede realizar el DSP directamente con los registros internos de las unidades aritméticas que procesan los datos en lenguaje C, se implementan a través de la memoria RAM de datos; este factor influye de manera importante en la cantidad de ciclos empleados por el código escrito en lenguaje C; para realizar esta operación en lenguaje Ensamblador, se carga la variable directamente en el registro interno que se necesita.

El espacio en memoria ocupado por los programas en lenguaje C es mucho mayor que el espacio necesario para realizar la misma labor en lenguaje Ensamblador. El hecho anterior tiene como fundamento lo siguiente: el programa en lenguaje C no solo

necesita para ejecutarse en el procesador el código de usuario sino que también requiere código adicional para que la traducción de lenguaje C a lenguaje Ensamblador sea lo mejor posible. Este código adicional se encuentra en las bibliotecas cabeceras con extensión .H provistas por el lenguaje C para realizar muchas de las labores que permite esta herramienta computacional. A manera de ejemplo demostrativo del hecho anterior se utiliza la ventana PROGRAM MEMORY del programa VISUALDSP++ para deducir el espacio que ocupa el código del programa debido al usuario y el generado por las bibliotecas del lenguaje C; código de usuario: 1959 *bytes* código de librerías del lenguaje C: 4986 *bytes*. El uso del lenguaje C en aplicaciones con memoria interna pequeña se vuelve restrictivo.

La figura 61 muestra los resultados obtenidos en los vectores real e imag, los resultados esperados, la señal cuadrada de entrada almacenada en el vector real con N = 16 muestras, la magnitud y la fase del espectro en frecuencia. Los resultados obtenidos con el DSP son muy similares a los resultados esperados evaluados con la herramienta computacional MATLAB a pesar de pequeñas disparidades que afectan los resultados a partir de la cuarta o quinta cifra decimal. Estas diferencias tienen dos causas: la primera es el diezmado que se hace en cada etapa del algoritmo para evitar el desbordamiento aritmético y la segunda tiene que ver con la resolución aritmética del DSP 16 *bits*.

FIGURA 61. Señal cuadrada y la magnitud de su espectro.



Fuente: El autor.

4.2. ALGORITMO Y PROGRAMAS PARA LA TRANSFORMADA RÁPIDA DE FOURIER EN BASE 4.

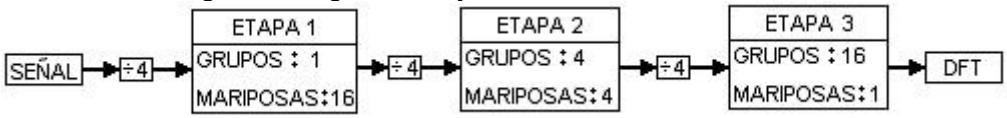
El procedimiento que se debe seguir para calcular la FFT en base 4 en un procesador de señales DSP es similar al que se realiza cuando se calcula una FFT en base 2 en el mismo procesador. Los cambios que se deben tener en cuenta son los siguientes:

- El número de elementos que se procesan por mariposa es igual a cuatro.
- Para evitar el desbordamiento que pueden producir las sumas se debe dividir por 4 y no por 2 los datos de entrada a las mariposas.
- La base del sistema ya no es 2 por lo tanto la inversión de los bits para un número en base 4 cambia con respecto a la que se realiza en un algoritmo base 2.
- La progresión algorítmica es similar pero su base no es el sistema binario sino el sistema ternario; lo anterior se entiende mejor con un ejemplo: si el número de muestras de la señal de entrada es 64, el procedimiento algorítmico posee 3

etapas divididas (el número de etapas se calcula con la ecuación 72) así: la primera etapa tiene 1 grupo con 16 mariposas, la segunda tiene 2 grupos con 4 mariposas por grupo y la última tiene 4 grupos cada uno de ellos con una mariposa. La figura 62 muestra la progresión algorítmica para un número de muestras N igual a 64.

$$N = 4^{etaoas}$$
$$etapas = \log_4 N \tag{72}$$

FIGURA 62. Progresión algorítmica y antirebosamiento.

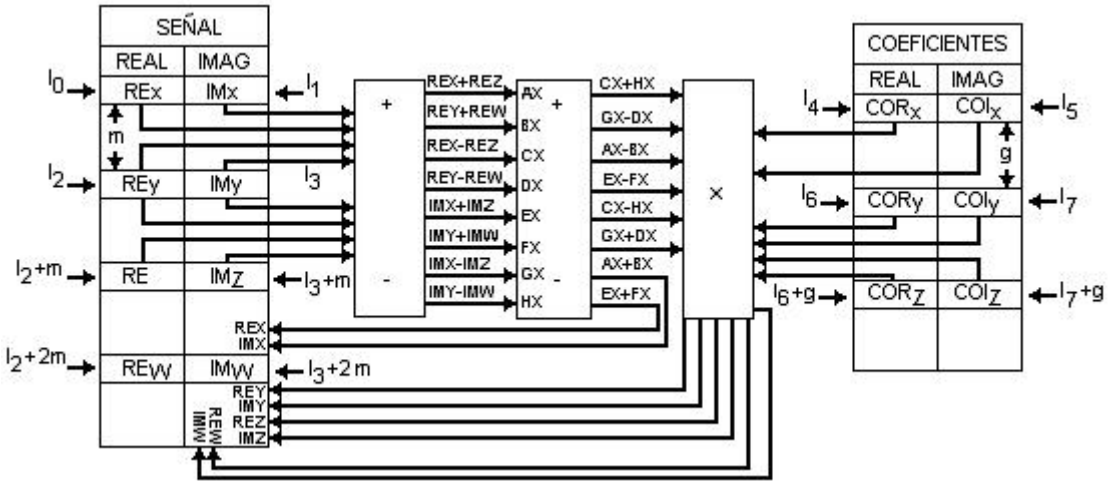


Fuente: El autor.

Para realizar la inversión de la dirección del vector de la señal de entrada cuando se ha finalizado el proceso de la FFT se debe tener en cuenta el procedimiento de la ecuación (73). La figura 63 muestra el procedimiento para evaluar una mariposa en una FFT en base 4. En el caso de este algoritmo los cuatro apuntadores de la unidad DAG_1 se dividen así: I_0 e I_2 se apuntan a los elementos de la parte real de la señal e I_1 e I_3 para apuntar a la parte imaginaria de la misma.

$$4^0 \times a_0 + 4^1 \times a_1 + ... + 4^n \times a_n \text{: orden invertido}$$
$$4^0 \times b_0 + 4^1 \times b_1 + ... + 4^n \times b_n \text{ : orden normal}$$
$$a_0 = b_n, b_0 = a_n, b_1 = a_1 \tag{73}$$
$$a_n = [0,1,2,3], b_n = [0,1,2,3]$$

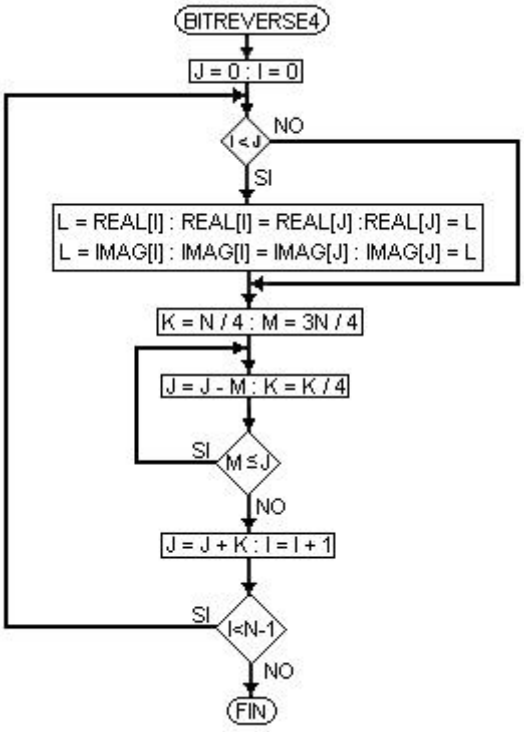
FIGURA 63. Procedimiento para calcular una mariposa en el algoritmo de la FFT en base 4.



Fuente: El autor.

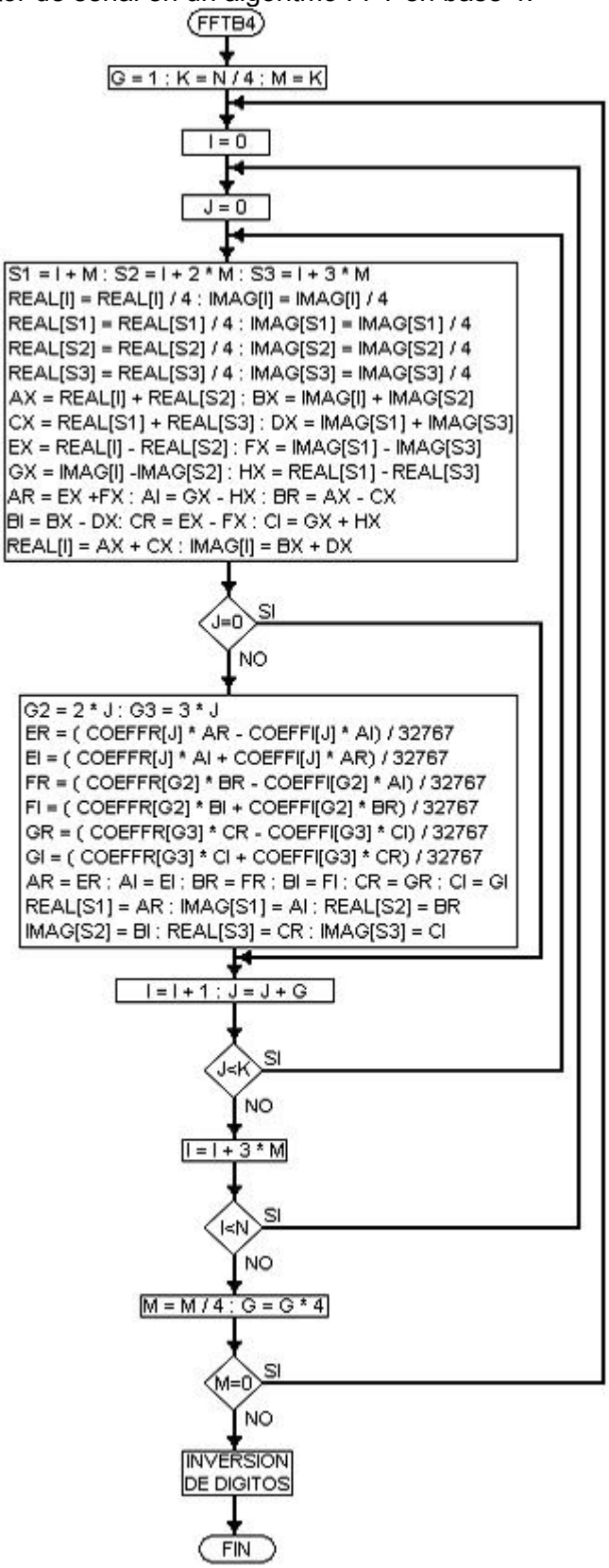
Los 4 punteros de la unidad DAG_2 se utilizan así: I_4 e I_6 apuntan a la parte real de los coeficientes e I_5 e I_7 apuntan a los coeficientes imaginarios; I_4 e I_5 son punteros circulares. Las sumas y restas de la mariposa se realizan en dos etapas para evitar repetir operaciones en las sumas de 4 elementos como lo indica la figura 5 del capítulo 1. Las figuras 64 y 65 muestran los diagramas de flujo de la FFT en base 4 y de la subrutina que invierte los dígitos de las direcciones del vector de resultados respectivamente.

FIGURA 64. Diagrama de flujo de la FFT en base 4.



Fuente: El autor.

FIGURA 65. Diagrama de flujo de la subrutina que invierte la dirección de los elementos del vector de señal en un algoritmo FFT en base 4.



Fuente: El autor.

Las figuras 66 y 67 contienen el código de la FFT en base 4 e inversión de dígitos para el lenguaje ensamblador de la familia 21xx de los DSP de ANALOG DEVICES y la figura 68 el código en lenguaje ANSI C de estas dos subrutinas.

FIGURA 66. Código de la FFT en base 4 en lenguaje ensamblador.

```
#define N 1024
#define BASE 5
#define NBR 16
#define MASKE 5461
#define MASKO 10922

section/data data1;
.VAR REAL[N]="real.dat";
.VAR IMAG[N]="imag.dat";
.VAR REALREV[N];
.VAR IMAGREV[N];

section/pm pm_da;
.VAR COEFFR[N]="coeffr.dat";
.VAR COEFFI[N]="coeffi.dat";

section/pm interrupts;
__reset: JUMP start; NOP; NOP; NOP;
     RTI; NOP; NOP; NOP;
     RTI; NOP; NOP; NOP;
     RTI; NOP; NOP; NOP;
     RTI; NOP; NOP; NOP;
     RTI; NOP; NOP; NOP;
     RTI; NOP; NOP; NOP;
     RTI; NOP; NOP; NOP;
     RTI; NOP; NOP; NOP;
     RTI; NOP; NOP; NOP;
     RTI; NOP; NOP; NOP;
     RTI; NOP; NOP; NOP;

section/pm program;
start: L0=0;L1=0;L2=0;L3=0;
       M0=0;M1=1;M2=N/4;M3=-(N/4);
       I0=REAL;I1=IMAG;I4=N/4;I5=N/4;
       L6=0;L7=0;M4=1;M5=0;
       I4=COEFFR;I5=COEFFI;
       CNTR=BASE;
DO ETAPAS UNTIL CE;
       I0=REAL;I1=IMAG;
       I2=I0;
       I3=I1;
       MODIFY(I2,M2);MODIFY(I2,M2);
       MODIFY(I3,M2);MODIFY(I3,M2);
       CNTR=M4;
DO GRUPOS UNTIL CE;
    CNTR=M2;
DO MARIPOSAS UNTIL CE;
    SI=DM(I0,M0);SR=ASHIFT SI BY -2 (HI);
    AX0=SR1;SI=DM(I2,M2);
    SR=ASHIFT SI BY -2 (HI);AY0=SR1;
    SI=DM(I1,M0);SR=ASHIFT SI BY -2 (HI);
    AX1=SR1;AR=AX0+AY0;
    SI=DM(I3,M2);SR=ASHIFT SI BY -2 (HI);
    AY1=SR1;
    DM(I0,M2)=AR;AR=AX1+AY1;
    DM(I1,M2)=AR;AR=AX0-AY0;
    SI=DM(I0,M0);SR=ASHIFT SI BY -2 (HI);
    AX0=SR1;SI=DM(I2,M0);
    SR=ASHIFT SI BY -2 (HI);AY0=SR1;
    MR0=AR;AR=AX1-AY1;
    MR1=AR;AF=AX0-AY0;
    SI=DM(I1,M0);AR=MR1-AF;
    SR=ASHIFT SI BY -2 (HI);
    AX1=SR1;SI=DM(I3,M0);
    SR=ASHIFT SI BY -2 (HI);AY1=SR1;
    DM(I1,M3)=AR;AR=MR1+AF;
    DM(I3,M3)=AR;AF=AX1-AY1;
    AR=MR0+AF;
    DM(I0,M3)=AR;AR=MR0-AF;
    DM(I2,M3)=AR;AF=AX0+AY0;
    MR0=DM(I0,M0);AR=MR0+AF;
    DM(I0,M0)=AR;AR=MR0-AF;
    DM(I2,M0)=AR;AF=AX1+AY1;
    MR0=DM(I1,M0);AR=MR0+AF;
    DM(I1,M0)=AR;AR=MR0-AF;
    DM(I3,M0)=AR;AY0=COEFFR;AX0=I4;AR=AX0-AY0;
    IF NE JUMP MULTIPLICAR;
    MODIFY(I4,M4);MODIFY(I5,M4);
    JUMP UNO;
MULTIPLICAR: M7=AR; I6=I4;I7=I5;
    MODIFY(I6,M7);MODIFY(I7,M7);
    MODIFY(I0,M2);MODIFY(I1,M2);
    MX0=DM(I0,M0),MY0=PM(I4,M4);
    MX1=DM(I1,M0),MY1=PM(I5,M4);
    MR=MX0*MY0 (SS); MR=MR-MX1*MY1 (RND);
    DM(I0,M3)=MR1,MR=MX0*MY1 (SS);
    MR=MR+MX1*MY0 (RND);DM(I1,M3)=MR1;
    MX0=DM(I2,M0),MY0=PM(I6,M7);
    MX1=DM(I3,M0),MY1=PM(I7,M7);
    MR=MX0*MY0 (SS);MR=MR-MX1*MY1 (RND);
    DM(I2,M2)=MR1,MR=MX0*MY1 (SS);
    MR=MR+MX1*MY0 (RND);DM(I3,M2)=MR1;
    MX0=DM(I2,M0),MY0=PM(I6,M7);
    MX1=DM(I3,M0),MY1=PM(I7,M7);
    MR=MX0*MY0 (SS);MR=MR-MX1*MY1 (RND);
    DM(I2,M3)=MR1,MR=MX0*MY1 (SS);
     MR=MR+MX1*MY0 (RND);DM(I3,M3)=MR1;
UNO: MODIFY(I0,M1);MODIFY(I2,M1);MODIFY(I1,M1);
MARIPOSAS:  MODIFY(I3,M1);
AR=CNTR;AR=AR-1;
IF EQ JUMP GRUPOS;
MX0=3,MY0=M2;MR=MX0*MY0 (SS);
SR=ASHIFT MR0 BY -1 (HI);M0=SR1;
MODIFY(I0,M0);MODIFY(I1,M0);
MODIFY(I2,M0);MODIFY(I3,M0);
GRUPOS: M0=0;AR=CNTR;AR=AR-1;
 IF EQ JUMP ETAPAS;
 SI=M4;SR=ASHIFT SI BY 2 (HI);
 M4=SR1;SI=M2;SR=ASHIFT SI BY -2 (HI);
 M2=SR1;AR=-SR1;
ETAPAS: M3=AR;
```

Fuente: El autor.

FIGURA 67. Código de la subrutina para invertir los dígitos de la dirección de los elementos del vector de señal.

```
IBIT4R: MR0=0;MR1=0;L0=0;L1=0;L2=0;
M1=1;M0=0;M2=0;AY1=0;I0=REAL;I1=IMAG;
CNTR=N-1;AX1=N-1;
DO BIT1 UNTIL CE;
AY0=CNTR;AR=AX1-AY0;AF=AR-AY1;
IF GE JUMP BIT2;
I2=REAL;M2=AY1;MODIFY(I2,M2);
MX0=DM(I0,M0);MY0=DM(I2,M0);
DM(I2,M0)=MX0;DM(I0,M0)=MY0;
I2=IMAG;MODIFY(I2,M2);MX0=DM(I1,M0);
MY0=DM(I2,M0);DM(I1,M0)=MY0;
DM(I2,M0)=MX0;
BIT2:  MODIFY(I0,M1);MODIFY(I1,M1);
MR1=NA;MR0=NBITR;
BIT4:  AF=MR0-AY1;
IF GT JUMP BIT3;
AR=MR0-AY1;AR=-AR;AY1=AR;
SR=LSHIFT MR0 BY -2 (LO);MR0=SR0;
SR=LSHIFT MR1 BY -2 (LO);MR1=SR0;
JUMP BIT4;
BIT3:  AR=MR1+AY1;AY1=AR;
BIT1:  NOP;
       RTS;
```

Fuente: El autor.

FIGURA 68. Código de la FFT en base 4 e inversión de dígitos en lenguaje ANSI C.

```
#define N 1024
#define NBITS 5
int dm ESPECTRO[N];
int dm real[N] = {
        #include "real.dat"
        };
int dm imag[N] = {
        #include "imag.dat"
        };
int pm coeffr[N] = {
        #include "coeffr.dat"
        };
int pm coeffi[N] = {
        #include "coeffi.dat"
        };
void bitreverse(void);
void fft4r(void);
main()
{
 fft4r(); return 0;
}
void fft4r(void)
{long ax,bx,cx,dx,ex,fx,gx,hx;
  int g,i,j,k,m,s1,s2,s3,g2,g3;
  g=1;k=N>>2;m=k;
  while(m){i=0;
  while(i<N) {j=0;
   while(j<k){s1=i+m,s2=i+2*m;
   s3=i+3*m;real[i]>>=2;imag[i]>>=2;
real[s1]>>=2,imag[s1]>>=2,real[s2]>>=2,imag[s2]>>=2,real[s3]>>=2;
 imag[s3]>>=2; ax=real[i]+real[s2]; bx=imag[i]+imag[s2];
 cx=real[s1]+real[s3],dx=imag[s1]+imag[s3],ex=real[i]-real[s2];
 fx=imag[s1]-imag[s3],gx=imag[i]-imag[s2],hx=real[s1]-real[s3];
 real[s3]=ex-fx; imag[s3]=gx+hx,real[s2]=ax-cx,imag[s2]=bx-dx;
 real[s1]=ex+fx,imag[s1]=gx-hx,real[i]=ax+cx,imag[i]=bx+dx;
 if (j)
 {g2=2*j,g3=3*j,ax=real[s1],bx=imag[s1];
 cx=(coeffr[j]*ax-coeffi[j]*bx)>>15,real[s1]=cx;
 cx=(coeffr[j]*bx+coeffi[j]*ax)>>15,imag[s1]=cx;
 ax=real[s2],bx=imag[s2];
 cx=(coeffr[g2]*ax-coeffi[g2]*bx)>>15,real[s2]=cx;
 cx=(coeffr[g2]*bx+coeffi[g2]*ax)>>15,imag[s2]=cx,ax=real[s3];
 bx=imag[s3],cx=(coeffr[g3]*ax-coeffi[g3]*bx)>>15;
 real[s3]=cx,cx=(coeffr[g3]*bx+coeffi[g3]*ax)>>15,imag[s3]=cx;}
 i++;j+=g;}
 i+=3*m;}
 m>>=2;g<<=2;}
  bitreverse();}
void bitreverse(void)
{
 int l,i,j=0,k=0;
 for (i=0;i<N;i++)
 {if (i<j) {l=real[i],real[i]=real[j],real[j]=l;
   l=imag[i],imag[i]=imag[j],imag[j]=l;}
  j+=K2;if (j>=N) {j=j-K1;if (l>=K2)(k++;j=k;}
   else j=1;}}
```

Fuente: El autor.

Los dos programas pueden compararse computacionalmente teniendo en cuenta los siguientes parámetros: tamaño en *bytes* del archivo ejecutable generado por el compilador, número de ciclos de máquina empleados y el tiempo empleado por el procesador para ejecutar la subrutina. La comparación se realiza con un procesador de 16 bits y una frecuencia de reloj de 32 MHz. La figura 69 muestra la comparación computacional.

FIGURA 69. Comparación computacional entre el lenguaje C y el ensamblador de la familia ADSP 21xx.

	ENSAMBLADOR TAMAÑO: 2187 "bytes"		LENGUAJE C TAMAÑO: 8763 "bytes"	
MUESTRAS (N)	CICLOS DE MÁQUINA	TIEMPO (µS)	CICLOS DE MÁQUINA	TIEMPO (mS)
16	926	28,93	6166	0,1926
32				
64	4886	152,0	36856	1,1517
128				
256	24628	769,6	208252	6,5078
512				
1024	110680	6917,50	1090708	34,084

Fuente: El autor.

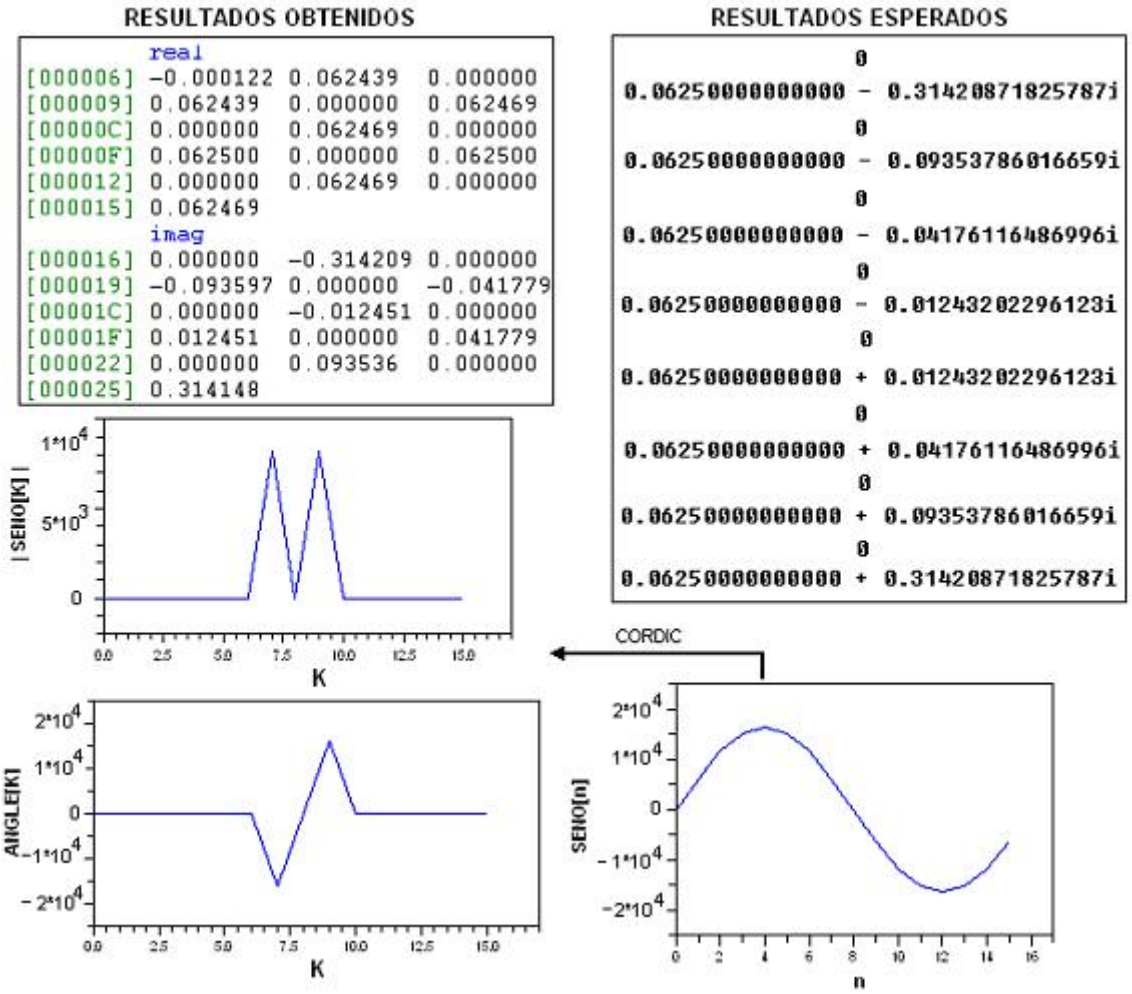
De los resultados computacionales de la FFT en base 4 de la figura 69, se puede concluir que tanto en lenguaje C como en lenguaje ensamblador los resultados obtenidos superan por un margen amplio a los resultados obtenidos en el algoritmo de la FFT en base 2 (Ver Figura 60). Con los resultados de las figuras 60 y 69 se puede concluir lo siguiente con respecto a la decisión de usar uno u otro algoritmo de acuerdo al número de muestras de la señal de entrada (N) y al número de ciclos que le toma al DSP calcular el espectro de la señal de entrada:

- Para valores de N menores o iguales a 16 muestras, se debe utilizar el algoritmo FFT en base 4.
- Para valores de N mayores a 16 muestras y menores que 32, se debe utilizar el algoritmo FFT en base 2.
- Para valores de N mayores a 32 muestras y menores que 64, se debe utilizar el algoritmo FFT en base 4.
- Para valores de N mayores a 64 muestras y menores que 128, se debe utilizar el algoritmo FFT en base 2.
- Para valores de N mayores a 128 muestras y menores que 256, se debe utilizar el algoritmo FFT en base 4.
- Para valores de N mayores a 256 muestras y menores que 512, se debe utilizar el algoritmo FFT en base 2.
- Para valores de N mayores a 512 muestras y menores que 1024, se debe utilizar el algoritmo FFT en base 2.

En la figura 70 se observan los resultados computacionales del algoritmo FFT en base 4 obtenidos al calcular el espectro de una señal seno con N igual a 16 muestras. Al igual que en el algoritmo FFT en base 2 se aprecian desviaciones de los resultados

obtenidos con respecto a los resultados esperados principalmente por el efecto que tiene sobre el algoritmo el diezmado de las muestras en cada una de las etapas del algoritmo. En el algoritmo FFT en base 4 el error es mayor que en el algoritmo FFT en base 2 ya que cada elemento se diezma por un valor mayor (4 en lugar de 2). Una manera de corregir esto es aumentar la resolución del procesador utilizado para el cálculo del espectro de la señal (ver Proakis & Manolakis capítulo 7).

FIGURA 70. Señal seno[n] y la magnitud de su espectro | SENO[k] |.



Fuente: El autor.

4.3. ALGORITMO Y PROGRAMAS PARA LA TRANSFORMADA RÁPIDA DE FOURIER DE BASE COMBINADA O PARTIDA.

Este algoritmo inicialmente debe dividir la cantidad de elementos de la señal según la expresión dada en la ecuación (74) donde r_1, r_2….r_n son valores enteros.

$$N = r_1 r_2 .....r_m \tag{74}$$

A continuación se expresan las variables del tiempo (n) y de la frecuencia (k) de acuerdo a las siguientes relaciones (75) y (76)

$$n = n_{m-1}(r_1 r_2 ... r_{m-1}) + n_{m-2}(r_1 r_2 ... r_{m-2}) + ... + n_1 r_1 + n_0 \quad (75)$$

$$k = k_{m-1}(r_2 r_3 ... r_m) + k_{m-2}(r_3 r_4 ... r_m) + ... + k_1 r_m + k_0 \quad (76)$$

El rango de valores de los términos n_{m-1}, n_{m-2}, n_1, n_0, k_{m-1}, k_{m-2}, k_1 y k_0 se obtienen así:

$$n_{i-1} = 0,1,2,.....,r_i - 1 \quad 1 \le i \le m \quad (77)$$

$$k_i = 0,1,2,....,r_{m-i} - 1 \;\; 0 \le i \le m-1 \quad (78)$$

Como ejemplo ilustrativo se tomará una señal con un número de muestras N = 24. Este número puede dividirse de la siguiente forma:

$$\mathrm{N} = 2 \times 3 \times 4 \;;\; \mathrm{m=3} \quad (79)$$

Esta forma de representar el número 24 lleva a las siguientes relaciones para las variables n y k:

$$n = 6n_2 + 2n_1 + n_0 \;;\; k = 12k_2 + 4k_1 + k_0 \quad (80)$$

El rango de valores que se obtiene en las variables n_2, n_1, n_0, k_2, k_1 y k_0 para este ejemplo se muestra en la tabla 13.

TABLA 13. Rango de valores para las variables que componen a n y k.

n_0	0,1
n_1	0,1,2
n_2	0,1,2,3
k_0	0,1,2,3
k_1	0,1,2
k_2	0,1

Al definir las variables n y k de esta manera, la ecuación de la Transformada Discreta de Fourier se convierte en:

$$X(n_2, n_1, n_0) = \sum_{k_0=0}^{3} \sum_{k_1=0}^{2} \sum_{k_2=0}^{1} x(k_2, k_1, k_0) W^{nk} \quad (81)$$

Siendo:

$$W^{nk} = W^{[6n_2+2n_1+n_0][12k_2+4k_1+k_0]} \quad (82)$$

El factor W^{nk} se puede reducir si se tiene en cuenta que $W^{72n_2k_2} = W^{24n_1k_1} = W^{24n_2k_1} = 1$. A partir de esta reducción se puede factorizar de dos formas la ecuación (82), lo cual da como resultado dos formas algorítmicas diferentes del mismo procedimiento. La primera de ellas es nombrada diezmado en frecuencia y la segunda diezmado en el tiempo. La ecuación (83) corresponde al diezmado en frecuencia y la ecuación (84) al diezmado en el tiempo.

$$W^{6n_2[k_0]+2n_1[4k_1+k_0]+n_0[12k_2+4k_1+k_0]} \quad (83)$$

$$W^{12k_2[n_0]+4k_1[2n_1+n_0]+k_0[6n_2+2n_1+n_0]} \quad (84)$$

La factorización por (83) o por (84) de la ecuación (82) da como resultado la división del algoritmo en tres etapas:

$$X_1(n_0,k_1,k_0)=\sum_{k_2=0}^{1}x(k_2,k_1,k_0)W^{n_0(12k_2+4k_1+k_0)} \quad (85)$$

$$X_2(n_0,n_1,k_0)=\sum_{k_1=0}^{2}X_1(n_0,k_1,k_0)W^{2n_1(4k_1+k_0)} \quad (86)$$

$$X_3(n_0,n_1,n_2)=\sum_{k_0=0}^{3}X_2(n_0,n_1,k_0)W^{6n2(k_0)} \quad (87)$$

En la ecuación (87) se observa la inversión de las cifras que componen la dirección del vector X_3. Para reposicionar los elementos del vector en la dirección adecuada, se debe tener en cuenta la siguiente ecuación:

$$k=12k_2+4k_1+k_0; n=6n_2+2n_1+n_0 \Rightarrow n_2=k_0, n_1=k_1, n_0=k_2 \quad (88)$$

Para programar este algoritmo se deben tener en cuenta los siguientes aspectos: número de etapas, grupos de mariposas por etapa, mariposas por grupo y el número de elementos que procesa cada mariposa. El primer factor está determinado por la variable m (ver la ecuación 79), el segundo y el tercero son determinados por las constantes que acompañan a cada una de las variables que conforman a n y k y el último factor es determinado a través de la descomposición del tamaño de la muestra N realizada en (79). En la tabla 14 se muestran los valores de las constantes para el ejemplo con N igual a 24.

TABLA 14. Distribución de grupos y mariposas por grupo para N igual a 24 muestras.

Etapa	1	2	3
Grupos	1	2	6
Mariposas	12	4	1
Elementos por mariposa	2	3	4

Otro factor importante en la elaboración del algoritmo es tener en cuenta que el valor por el cual se debe dividir el vector de señal antes de procesar una etapa ya no es una constante como lo es en los casos de los algoritmos de base 2 y base 4. En este caso se dividen las muestras de la señal antes de procesarlas por el valor del tamaño de la mariposa; esto con el fin de evitar el desbordamiento aritmético. Para el ejemplo de N igual a 24, en la figura 73 se muestra por cual constante hay que dividir en cada una de las etapas.

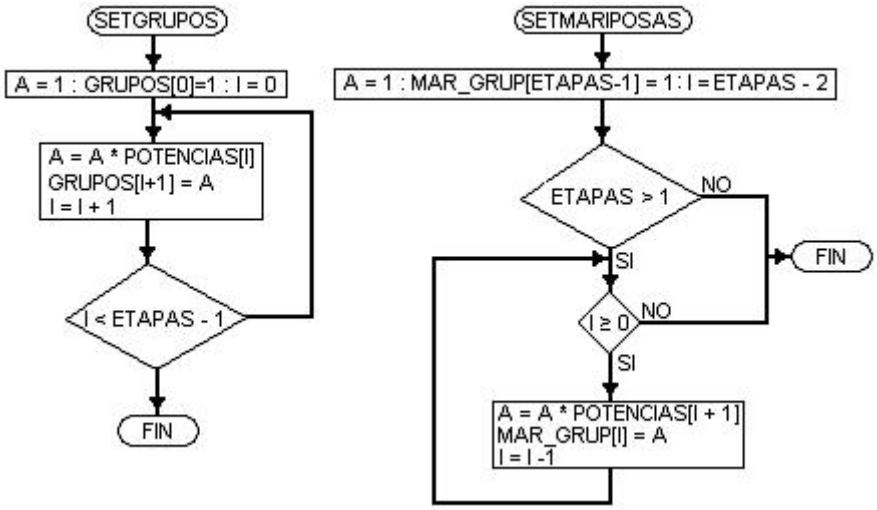
FIGURA 71. Distribución del diezmado para evitar el desbordamiento aritmético entre etapas, para N igual a 24.

Fuente: El autor.

A continuación se describirán las subrutinas que se utilizan para calcular la FFT de base mixta con un tamaño de muestra N factorizable en factores de 2, 3, 4, 5, 6 y 8.

Las dos primeras subrutinas se llaman SetGrupos y SetMariposas. SetGrupos calcula la cantidad de grupos que existe en cada una de las etapas del algoritmo mientras que SetMariposas calcula la cantidad de mariposas en cada uno de los grupos; las dos subrutinas reciben como entrada la serie de factores que componen a N en el vector potencias. En la figura 72 se muestran los diagramas de flujo de las dos subrutinas.

FIGURA 72. Diagramas de flujo de las subrutinas SetGrupos y SetMariposas.

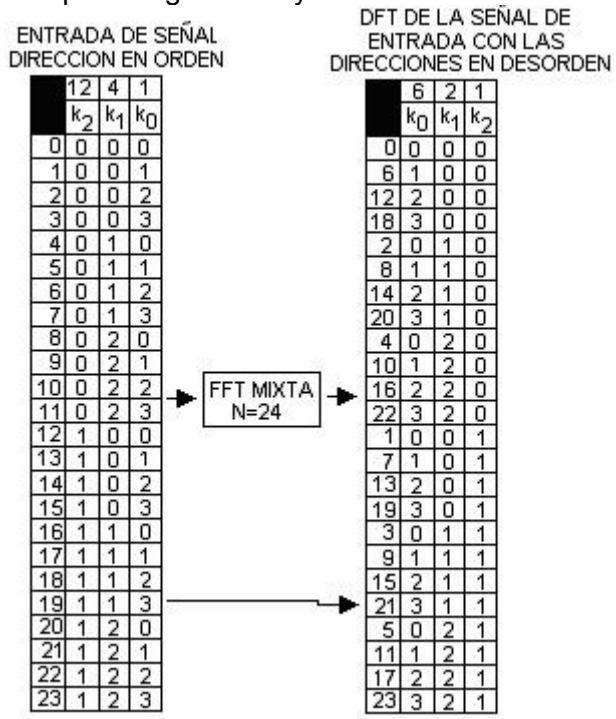


Fuente: El autor.

La subrutina que permite invertir la dirección de los elementos del vector de salida se llama ibitr_gen. En esta subrutina se realiza el cálculo de los dígitos que componen un número. Si un número cualquiera se puede representar como se especifica en las ecuaciones 79 y 80; entonces también se pueden obtener los dígitos que lo componen en cada uno de dichos sistemas de numeración. A manera de ejemplo se tomará el muestra número 19 de una señal con N igual a 24 muestras y se obtendrán los dígitos que lo conforman en el sistema numérico que representa las muestras de frecuencia k y con estos obtener la representación de este número después de la inversión de

dígitos provocada por la realización del algoritmo sobre las muestras de la señal de entrada. Lo anterior se observa en la figura 73.

FIGURA 73. Inversión de dígitos provocada por el algoritmo FFT en base mixta de diezmado en frecuencia para N igual a 24 y N = 2*3*4.



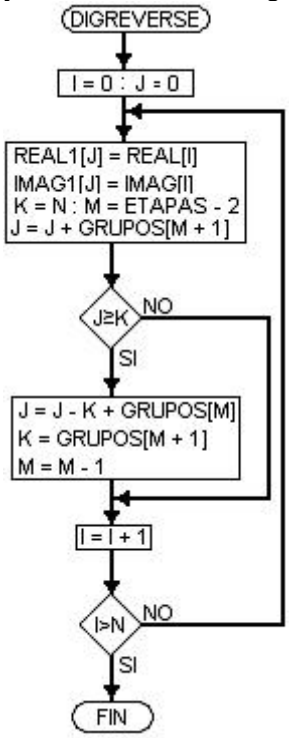
	12	4	1
	k_2	k_1	k_0
0	0	0	0
1	0	0	1
2	0	0	2
3	0	0	3
4	0	1	0
5	0	1	1
6	0	1	2
7	0	1	3
8	0	2	0
9	0	2	1
10	0	2	2
11	0	2	3
12	1	0	0
13	1	0	1
14	1	0	2
15	1	0	3
16	1	1	0
17	1	1	1
18	1	1	2
19	1	1	3
20	1	2	0
21	1	2	1
22	1	2	2
23	1	2	3

	6	2	1
	k_0	k_1	k_2
0	0	0	0
6	1	0	0
12	2	0	0
18	3	0	0
2	0	1	0
8	1	1	0
14	2	1	0
20	3	1	0
4	0	2	0
10	1	2	0
16	2	2	0
22	3	2	0
1	0	0	1
7	1	0	1
13	2	0	1
19	3	0	1
3	0	1	1
9	1	1	1
15	2	1	1
21	3	1	1
5	0	2	1
11	1	2	1
17	2	2	1
23	3	2	1

Fuente: El autor.

La subrutina que se requiere para llevar a cabo la inversión de la dirección es un procedimiento universal de conversión numérica el cual fue realizado mediante la subrutina DigReverse que se puede observar en la figura 74.

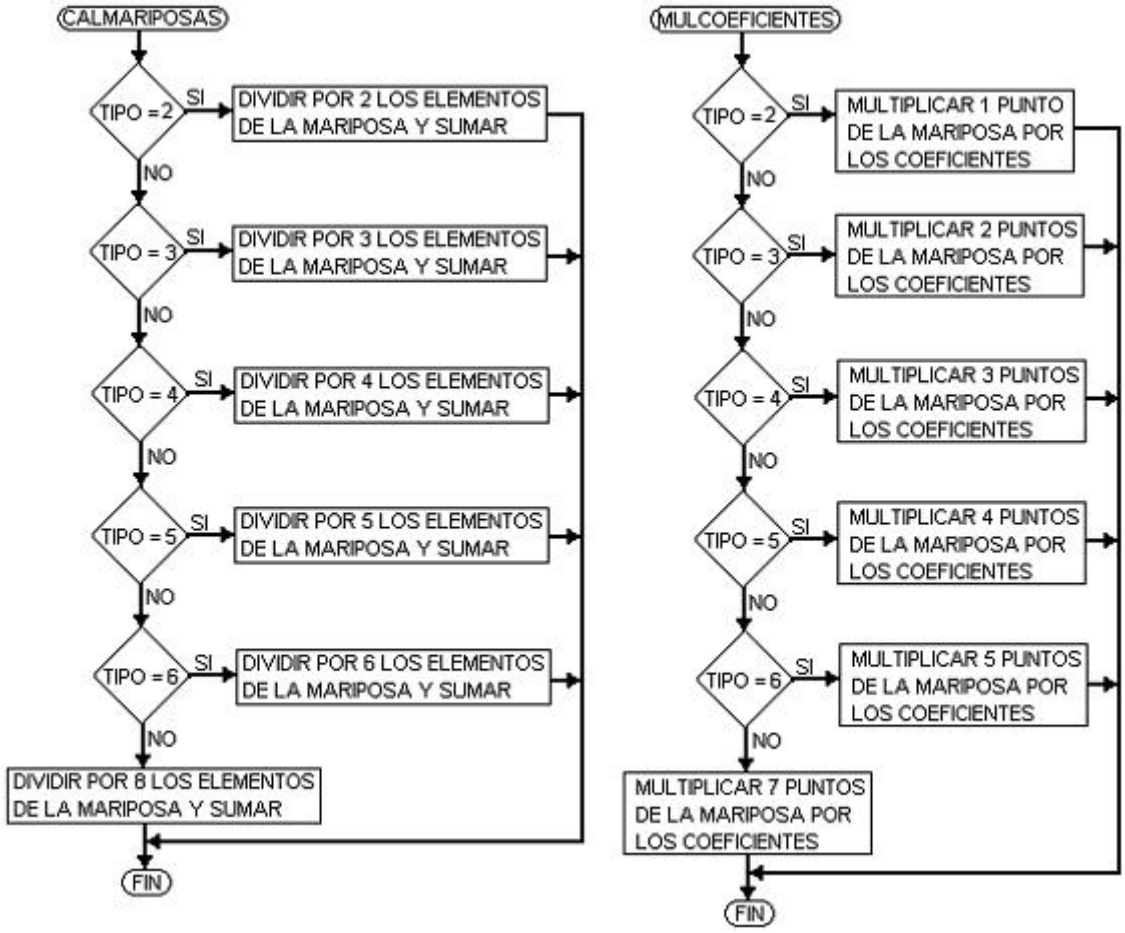
FIGURA 74. Diagrama de flujo de la subrutina DigReverse.



Fuente: El autor.

Las subrutinas que realizan las sumas, las multiplicaciones y evitan el desbordamiento aritmético de las mariposas se llaman CalMariposas y MulCoeficientes. Estas subrutinas tienen tres entradas: el primer elemento de la mariposa, el ancho de la mariposa y el número de elementos que procesa la mariposa. En la figura 75 se muestran los diagramas de flujo de estas dos subrutinas.

FIGURA 75. Diagramas de flujo de las subrutinas MulCoeficientes y CalMariposas.



Fuente: El autor.

La subrutina central encargada de llevar la lógica principal del algoritmo se llama fftmix y su estructura se muestra en la figura 76. Las figuras 77 y 78 muestran el código en lenguaje C de las subrutinas utilizadas para programar la FFT de base mixta y la inversión de los dígitos de la dirección del vector de salida. En la tabla 15 se realiza una comparación de rendimiento computacional con los algoritmos FFT en base 2 y 4 escritos en lenguaje C.

En la tabla de rendimiento computacional se pueden concluir los siguientes aspectos con respecto a cual algoritmo, (FFT base 2, FFT base 4 o FFT base mixta), utilizar dado el número de muestras de la señal de entrada:

- Para valores de N menores o iguales que 8, se debe utilizar el algoritmo FFT en base mixta.
- Para valores N mayores que 8 y menores o iguales que 16, se debe utilizar el algoritmo FFT en base 4.
- Para valores N mayores que 16 y menores o iguales que 24, se debe utilizar el algoritmo FFT en base mixta.

- Para valores de N mayores que 24 y menores o iguales que 32, se debe utilizar la FFT en base 2.
- Para valores de N mayores que 32 y menores o iguales que 48, se debe utilizar la FFT en base mixta.
- Para valores de N mayores que 48 y menores o iguales que 64, se debe utilizar la FFT en base 4.
- Para valores de N mayores que 64 y menores o iguales que 90, se debe utilizar la FFT en base mixta.
- Para valores de N mayores que 90 y menores o iguales que 128, se debe utilizar la FFT en base 2.
- Para valores de N mayores que 128 y menores o iguales que 160, se debe utilizar la FFT en base mixta.
- Para valores de N mayores que 160 y menores o iguales que 256, se debe utilizar la FFT en base 4.
- Para valores de N mayores que 256 y menores o iguales que 400, se debe utilizar la FFT en base mixta.
- Para valores de N mayores que 400 y menores o iguales que 512, se debe utilizar la FFT en base 2.

Otro factor que afecta el rendimiento de la FFT en base mixta es la estructura computacional que en cada etapa debe escoger el tipo de mariposa que se va a emplear; esto agrega ciclos computacionales que en realidad no hacen parte del algoritmo como tal pero que son necesarios para poder desarrollar el ciclo computacional necesario para llevar a cabo el algoritmo de FFT en base mixta.

Cuando el número de muestras N tiene factores 2 y 3 en su factorización es preferible elegir el factor de factorización 6; lo mismo se puede hacer si el número tiene factores 4 y 2 se debe implementar el factor 8. Utilizando los cambios anteriores se pueden lograr ahorros de un 25% en cuanto al número de multiplicaciones que se deben realizar.

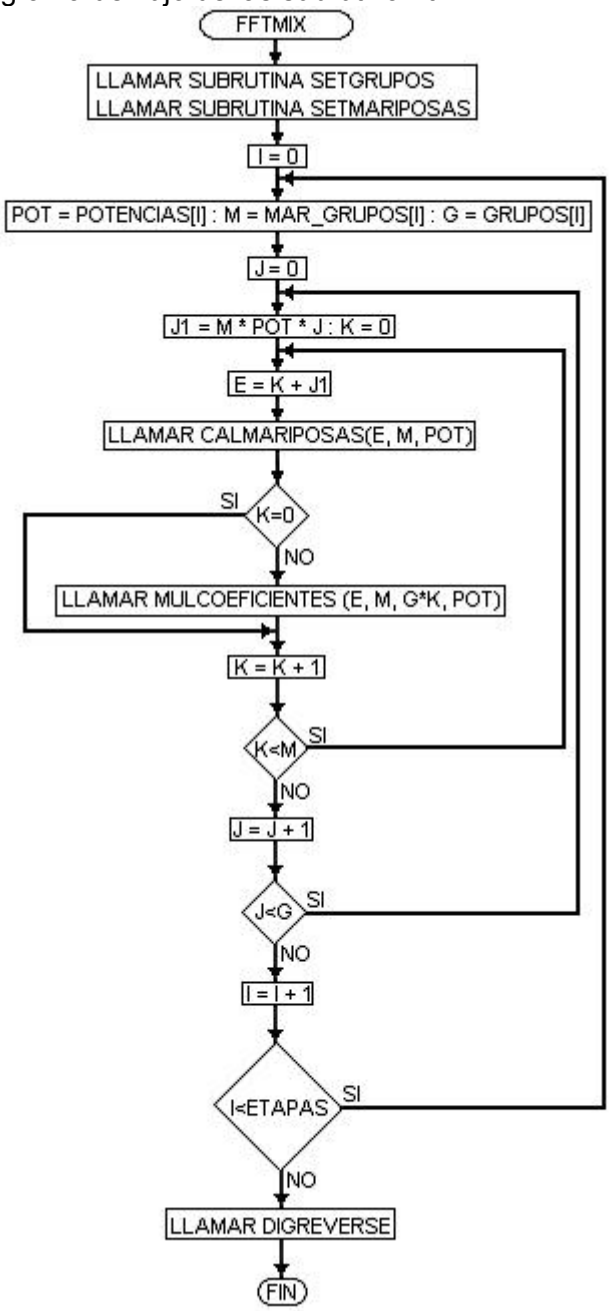
Factores como el 15 y 12 se pueden llevar a cabo teniendo en cuenta que se componen con factores primos, 15 = 3 * 5 y 12 = 4 * 3, se pueden realizar sin multiplicaciones entre etapas si se escogen adecuadamente las ecuaciones 79 y 80 que componen a n y k. El inconveniente de esta técnica radica en que tanto n como k entran y salen con las direcciones invertidas; lo anterior necesita dos algoritmos

adicionales que ingresen los datos en desorden y otro para ordenar la salida de la etapa. Los factores 12 y 15 pueden ser ejecutados de esta manera en el algoritmo de la FFT de base mixta como otro par de factores que componen un número cualquiera N.

TABLA 15. Comparación de rendimiento computacional entre los algoritmos FFT en base 2, FFT en base 4 y FFT en base mixta elaborados en lenguaje C.

	FFT2C			FFT4C			FFT MIXTAC	
N	CICLOS	TIEMPO ms	N	CICLOS	TIEMPO ms	N	CICLOS	TIEMPO ms
8	3223	0,100				8: 8	2570	0,080
						9: 3×3	6244	0,195
						10: 2×5	6571	0,205
						12: 3×4	6556	0,204
16	8785	0,274	16	6166	0,192	15: 3×5	12892	0,402
						20: 4×5	14440	0,451
						24: 4×6	15319	0,478
32	22316	0,697				30: 5×6	28772	0,899
						36: 6×6	31663	0,989
						40: 5×8	32933	1,029
						48: 6×8	38522	1,203
						50: 2×5×5	60596	1,893
64	54601	1,706	64	36856	1,151	60: 3×4×5	66183	2,068
						80: 4×4×5	77765	2,430
						90: 3×5×6	121141	3,785
						100: 4×5×5	129525	4,047
128	128922	4,028				120: 4×5×6	145725	4,553
						150: 5×5×6	230858	7,214
						160: 4×5×8	181604	5,675
						180: 3×3×4×5	268319	8,384
256	298038	9,313	256	208252	6,507	200:5×5×8	292050	9,126
						300:3×4×5×5	505066	15,78
						400:4×4×5×5	621012	19,40
512	660437	20,63				500:4×5×5×5	939011	29,34

FIGURA 76. Diagrama de flujo de las subrutina fftmix.



Fuente: El autor.

FIGURA 77.

. Código en lenguaje C de la FFT de base mixta. Parte A.

```
#include <math.h>
#include <fftmix.h>
#define N 24
#define ETAPAS 3
int dm real [N] = {#include"real.dat"};
int dm imag[N] = {#include "imag.dat"};
int pm coeffr [N] = {#include "coeffr.dat"};
int pm coeffi [N] = {#include "coeffi.dat"};
int dm ESPECTRO [N];
int dm potencias [ETAPAS]= {2, 3,4};
int dm grupos [ETAPAS];
int dm mar_grup [ETAPAS];
main () {fftmix ( );fft_shift ();return 0 ;}
void fftmix (void)
{int i, j, k, pot, e, m, g, j1;
SetGrupos ( ); SetMariposas ( );
for (i=0; i<ETAPAS ; i++)
{pot=potencias [i]; m=mar_grup[i]; g= grupos [i];
for (j=0; j<g; j++)
{j1=m*pot*j;
for (k=0;k<m; k++)
{e=k+j1; CallMariposas (e, m, pot);
if (k!=0) MulCoeficientes (e, m, g*k, pot); } }}
bitreverse ( ) ;}
 void CallMariposas(int pri,int sep,int tipo )
{
 int s1,s2,s3,s4;
 long a,b,a1,a2,b1,b2,c1,c2,d1,d2,e1,e2;
case 5:
  s1=pri+sep;s2=pri+2*sep;s3=pri+3*sep;s4=pri+4*sep;
  a=f4*real[pri];b=f4*imag[pri];
  a1=(a+f2*real[s1]-f3*real[s2]-f3*real[s3]+f2*real[s4])>>15;
  a2=(g2*imag[s1]+g3*imag[s2]-g3*imag[s3]-g2*imag[s4])>>15;
  b1=(b+f2*imag[s1]-f3*imag[s2]-f3*imag[s3]+f2*imag[s4])>>15;
  b2=(g3*real[s3]+g2*real[s4]-g2*real[s1]-g3*real[s2])>>15;
  c1=(a-f3*real[s1]+f2*real[s2]+f2*real[s3]-f3*real[s4])>>15;
  c2=(g3*imag[s1]-g2*imag[s2]+g2*imag[s3]-g3*imag[s4])>>15;
  d1=(b-f3*imag[s1]+f2*imag[s2]+f2*imag[s3]-f3*imag[s4])>>15;
  d2=(g2*real[s2]+g3*real[s4]-g3*real[s1]-g2*real[s3])>>15;
  e1=(a+f4*real[s1]-f4*real[s2]+f4*real[s3]+f4*real[s4])>>15;
  e2=(b+f4*imag[s1]+f4*imag[s2]+f4*imag[s3]+f4*imag[s4])>>15;
  real[pri]=e1;imag[pri]=e2;real[s1]=a1+a2;real[s4]=a1-a2;
  imag[s1]=b1+b2;imag[s4]=b1-b2;real[s2]=c1+c2;real[s3]=c1-c2;
  imag[s2]=d1+d2;imag[s3]=d1-d2;
break;
  case 4:
  s1=pri+sep;s2=pri+2*sep;s3=pri+3*sep;
  real[pri]>>=2;imag[pri]>>=2;real[s1]>>=2;imag[s1]>>=2;
  real[s2]>>=2;imag[s2]>>=2;real[s3]>>=2;imag[s3]>>=2;
  a1=real[pri]+real[s2];a2=imag[pri]+imag[s2];
  b1=real[s1]+real[s3];b2=imag[s1]+imag[s3];
  c1=real[pri]-real[s2];c2=imag[s1]-imag[s3];
  d1=imag[pri]-imag[s2];d2=real[s1]-real[s3];
  real[pri]=a1+b1;imag[pri]=a2+b2;real[s1]=c1+c2;imag[s1]=d1-d2;
  real[s2]=a1-b1;imag[s2]=a2-b2;real[s3]=c1-c2;imag[s3]=d1+d2;
  break;
case 3:s1=pri+sep;s2=pri+2*sep;
a=g4*real[pri];b=g4*imag[pri];
a1=(a-f1*real[s1]-f1*real[s2])>>15;
a2=(g1*imag[s1]-g1*imag[s2])>>15;
case 3:s1=pri+sep;s2=pri+2*sep;
a=g4*real[pri];b=g4*imag[pri];
a1=(a-f1*real[s1]-f1*real[s2])>>15;
a2=(g1*imag[s1]-g1*imag[s2])>>15;
b1=(b-f1*imag[s1]-f1*imag[s2])>>15;
b2=(g1*real[s2]-g1*real[s1])>>15;
c1=(a+g4*real[s1]+g4*real[s2])>>15;
c2=(b+g4*imag[s1]+g4*imag[s2])>>15;
real[pri]=c1;imag[pri]=c2;
real[s1]=a1+a2;imag[s1]=b1+b2;
real[s2]=a1-a2;imag[s2]=b1-b2; break;
case 2:s1=pri+sep;real[pri]>>=1;imag[pri]>>=1;
real[s1]>>=1;imag[s1]>>=1;
a=real[pri]-real[s1];b=imag[pri]-imag[s1];
real[pri]+=+real[s1];imag[pri]+=imag[s1];
real[s1]=a;imag[s1]=b;break;}}

void MulCoeficientes(int pri,int sep,int gru,int tipo)
{ long re1,im1,ar,br;
 int s1,g1;
 switch(tipo)
 { case 5:s1=pri+sep;re1=real[s1];im1=imag[s1];
ar=(re1*coeffr[gru]-im1*coeffi[gru])>>15;
br=(re1*coeffi[gru]+im1*coeffr[gru])>>15;
real[s1]=ar;imag[s1]=br;s1=pri+2*sep;
g1=2*gru;re1=real[s1];im1=imag[s1];
ar=(re1*coeffr[g1]-im1*coeffi[g1])>>15;
br=(re1*coeffi[g1]+im1*coeffr[g1])>>15;
real[s1]=ar;imag[s1]=br;s1=pri+3*sep;
g1=3*gru;re1=real[s1];im1=imag[s1];
ar=(re1*coeffr[g1]-im1*coeffi[g1])>>15;
br=(re1*coeffi[g1]+im1*coeffr[g1])>>15;
real[s1]=ar;imag[s1]=br;s1=pri+4*sep;g1=4*gru;
re1=real[s1];im1=imag[s1];
ar=(re1*coeffr[g1]-im1*coeffi[g1])>>15;
br=(re1*coeffi[g1]+im1*coeffr[g1])>>15;
real[s1]=ar;imag[s1]=br;break;
case 3:s1=pri+sep;re1=real[s1];im1=imag[s1];
ar=(re1*coeffr[gru]-im1*coeffi[gru])>>15;
br=(re1*coeffi[gru]+im1*coeffr[gru])>>15;
real[s1]=ar;imag[s1]=br;s1=pri+sep;
g1=2*gru;re1=real[s1];im1=imag[s1];
ar=(re1*coeffr[g1]-im1*coeffi[g1])>>15;
br=(re1*coeffi[g1]+im1*coeffr[g1])>>15;
real[s1]=ar;imag[s1]=br;break;
case 4:s1=pri+sep;re1=real[s1];im1=imag[s1];
ar=(re1*coeffr[gru]-im1*coeffi[gru])>>15;
br=(re1*coeffi[gru]+im1*coeffr[gru])>>15;
real[s1]=ar;imag[s1]=br;s1=pri+2*sep;
g1=2*gru;re1=real[s1];im1=imag[s1];
ar=(re1*coeffr[g1]-im1*coeffi[g1])>>15;
br=(re1*coeffi[g1]+im1*coeffr[g1])>>15;
real[s1]=ar;imag[s1]=br;s1=pri+3*sep;
g1=3*gru;re1=real[s1];im1=imag[s1];
ar=(re1*coeffr[g1]-im1*coeffi[g1])>>15;
br=(re1*coeffi[g1]+im1*coeffr[g1])>>15;
real[s1]=ar;imag[s1]=br;break;
case 2:s1=pri+sep;re1=real[s1];im1=imag[s1];
ar=(re1*coeffr[gru]-im1*coeffi[gru])>>15;
br=(re1*coeffi[gru]+im1*coeffr[gru])>>15;
real[s1]=ar;imag[s1]=br;break; }}
```

Fuente: El autor.

FIGURA 78. Código en lenguaje C de la FFT de base mixta. Parte B.

```
void SetMariposas (void)
{int a=1, i; mar_grup [ETAPAS-1]=1;
 if (ETAPAS>1)
 {for (i=ETAPAS-2;i>=0;i--)
 {a*=potencias[i+1]; mar_grup[i]=a; }}}
long magnitud(long RE,long IM)
{return (sqrt(RE*RE+IM*IM));}
void fft_shift(void){
 int a,i;
 for (i=0;i<N/2;i++)
{a=ESPECTRO[i];ESPECTRO[i]=ESPEC
TRO[i+N/2];
 ESPECTRO[i+N/2]=a; }}
```

```
void SetGrupos(void){int a=1,i;grupos[0]=1;
for (i=0;i<ETAPAS-1;i++){ a*=potencias[i];
grupos[i+1]=a; }}
void digreverse(void)
{
 int k,j=0,im;
 for (i=0;i<N;i++)
 {real[j]=real[i];imag[j]=imag[i];
  k=N;m=ETAPAS-2;j+=grupos[m+1];
  while(j>=k)
  {j=j-k+grupos[m];k=grupos[m+1];
   m--;}}
}
```

Fuente: El autor.

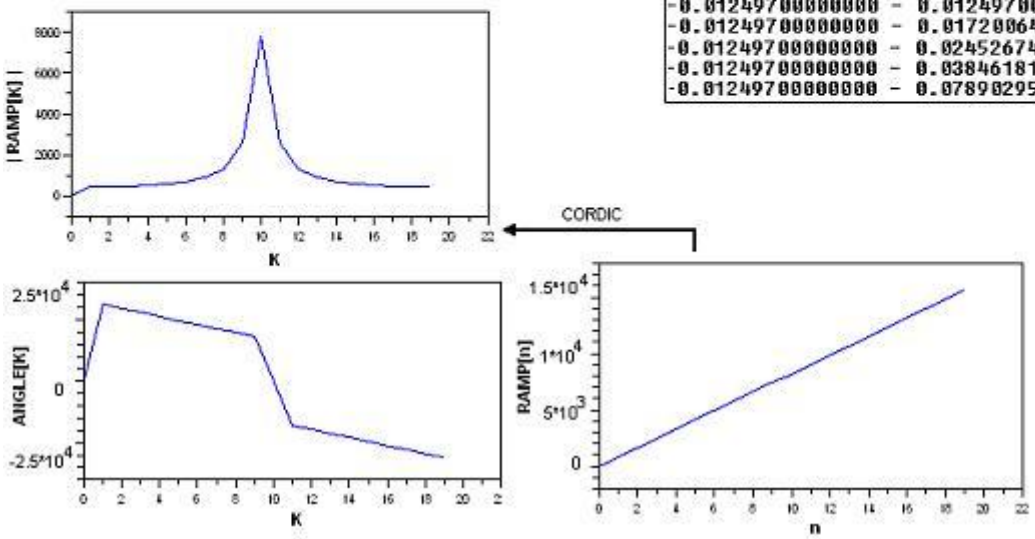
FIGURA 79. Señal PULSOS[n] y la magnitud de su espectro calculada con el algoritmo FFT de base mixta.

RESULTADOS OBTENIDOS

```
        real
[00007B] 0.237396  -0.012512 -0.012512 -0.012512
[00007F] -0.012543 -0.012512 -0.012512 -0.012512
[000083] -0.012543 -0.012512 -0.012512 -0.012512
[000087] -0.012543 -0.012512 -0.012512 -0.012512
[00008B] -0.012543 -0.012512 -0.012512 -0.012512
        imag
[00008F] 0.000000  0.078888  0.038422  0.024506
[000093] 0.017212  0.012451  0.009033  0.006348
[000097] 0.004059  0.001953  -0.000031 -0.002014
[00009B] -0.004059 -0.006348 -0.009064 -0.012512
[00009F] -0.017212 -0.024536 -0.038452 -0.078918
```

RESULTADOS ESPERADOS

```
 0.237443000000000
-0.012497000000000 + 0.078902952678891i
-0.012497000000000 + 0.038461811164081i
-0.012497000000000 + 0.024526743487301i
-0.012497000000000 + 0.017208644860131i
-0.012497000000000 + 0.012497000000001i
-0.012497000000000 + 0.009079601972481i
-0.012497000000000 + 0.006367539542331i
-0.012497000000000 + 0.004068521443821i
-0.012497000000000 + 0.001979330350741i
-0.012497000000000 - 0.000000000000001i
-0.012497000000000 - 0.001979330350741i
-0.012497000000000 - 0.004068521443821i
-0.012497000000000 - 0.006367539542331i
-0.012497000000000 - 0.009079601972481i
-0.012497000000000 - 0.012497000000001i
-0.012497000000000 - 0.017208644860131i
-0.012497000000000 - 0.024526743487301i
-0.012497000000000 - 0.038461811164081i
-0.012497000000000 - 0.078902952678891i
```



Fuente: El autor.

En la figura 79 se observan los resultados computacionales del algoritmo FFT en base mixta obtenidos al calcular el espectro de una señal rampa con N igual a 20 muestras, (N = 4*5). Al igual que en el algoritmo FFT en base 2 y FFT en base 4 se aprecian desviaciones de los resultados obtenidos con respecto a los resultados esperados principalmente por el efecto que tiene sobre el algoritmo el diezmado de las muestras en cada una de las etapas del algoritmo. En el algoritmo FFT en base mixta el error es mayor que en los algoritmos FFT en base 2 y FFT en base 4 ya que cada elemento se

diezma por un valor mayor a medida que los valores que componen N se hacen más grandes.

4.4. ALGORITMO Y PROGRAMA PARA EL CÁLCULO DE LA TRANSFORMADA DE FOURIER DE DOS SEÑALES UTILIZANDO LA PARTE IMAGINARIA DE LA SEÑAL DE ENTRADA.

Dado que las señales que se utilizan en el procesamiento digital son en su mayoría reales, se puede utilizar la parte imaginaria de la señal de entrada para procesar una nueva señal real en este espacio de memoria. El espacio de memoria requerido para este algoritmo es dos veces más grande que el de una sola señal. Para ejecutar esta subrutina primero se deben llenar los vectores REAL e IMAG con el par de señales que se quieren procesar; a continuación se ejecuta la subrutina de la FFT, (algoritmos de FFT analizados previamente), y finalmente se ejecuta la subrutina que separa los espectros de las dos señales originales almacenando el espectro de la señal 1 en los vectores REAL e IMAG y el de la señal 2 en los vectores REAL1 e IMAG1.

Para calcular el número de ciclos que se necesita para calcular el espectro de dos señales discretas con un algoritmo FFT de cualquier base sin aprovechar la parte imaginaria de la señal de entrada, se necesita ejecutar 2 veces la FFT y la subrutina que invierte ya sea a la entrada o a la salida el orden de las direcciones de los datos de entrada; lo anterior se puede expresar mediante la ecuación (89) llamada ciclos1.

$$ciclos1 = 2*[ciclos(FFT) + ciclos(BITREVERSE)] \tag{89}$$

Para calcular el número de ciclos que se necesita para calcular el espectro de dos señales discretas con un algoritmo FFT de cualquier base aprovechando la parte imaginaria de la señal de entrada, se necesita ejecutar una vez la FFT y la subrutina que invierte ya sea a la entrada o a la salida el orden de las direcciones de los datos de entrada; lo anterior se puede expresar mediante una ecuación (90) aritmética llamada ciclos2.

$$ciclos2 = ciclos(FFT) + ciclos(BITREVERSE) + ciclos(FFT2SENAL1) \tag{90}$$

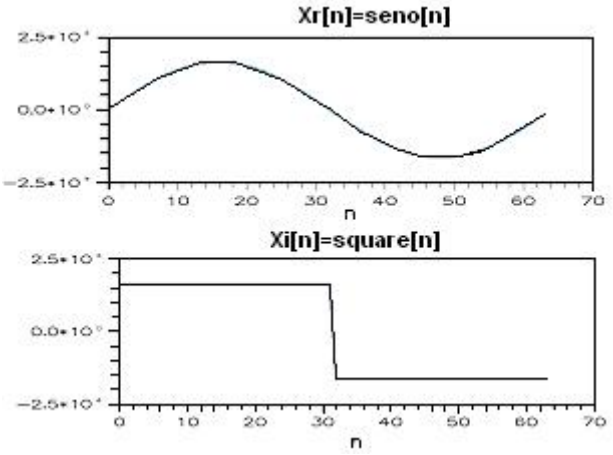
Para obtener el número de ciclos ahorrados mediante el uso del algoritmo FFT2SENAL1 solamente se debe realizar restar ciclos1 de ciclos2. En la tabla número 16 se observan los resultados obtenidos al utilizar el algoritmo FFT2SENAL1. Los datos de la columna ciclos1 se obtienen de la figura 60 mientras que los datos de la columna ciclos2 se obtienen aplicando la expresión (90).

TABLA 16. Comparación computacional entre algoritmos FFT equivalentes al algoritmo FFT2SENAL1.

N	Ciclos1	Ciclos2	Ahorros
8	6440	3680	2760
16	17570	9645	7925
32	44362	23481	20521
64	109202	57861	51341
128	257844	135382	122462
256	596076	310898	285178
512	1320874	686097	634777
1024	2865628	1484125	1381503

Para comprobar el funcionamiento del algoritmo FFT2SENAL1 se carga el vector REAL con una onda seno de 0,5V con N igual a 64 muestras y el vector IMAG con una onda cuadrada de 0,5V con N igual a 64 muestras. Se puede observar que el cálculo de la magnitud máxima de los 64 números complejos de los vectores REAL e IMAG no es mayor que la unidad: $\sqrt{0,5^2 + 0,5^2} = 0,707$. En la figura 80 se observa la disposición de las señales en los vectores REAL e IMAG.

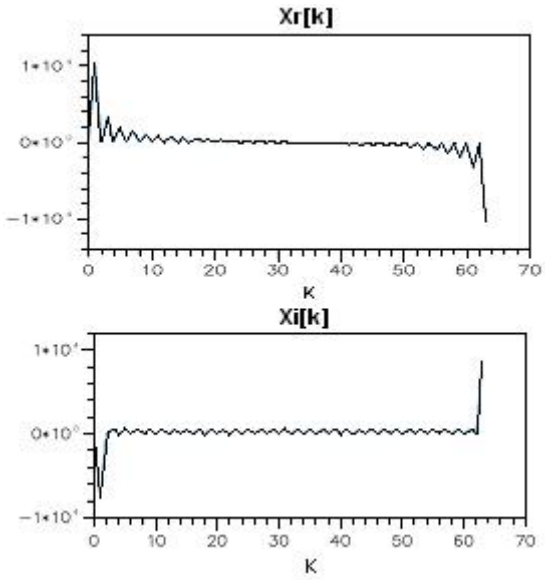
FIGURA 80. Señales iniciales en la parte imaginaria y en la parte real de la señal de entrada x[n].



Fuente: El autor.

En la figura 81 se observa el segundo paso para calcular el espectro de dos señales con una sola FFT; En esta parte se observa que los resultados tienen los mismos problemas que se presentan al calcular el espectro de una sola señal con un algoritmo FFT; el diezmado por etapas y la resolución del procesador las cuales inciden directamente sobre los resultados.

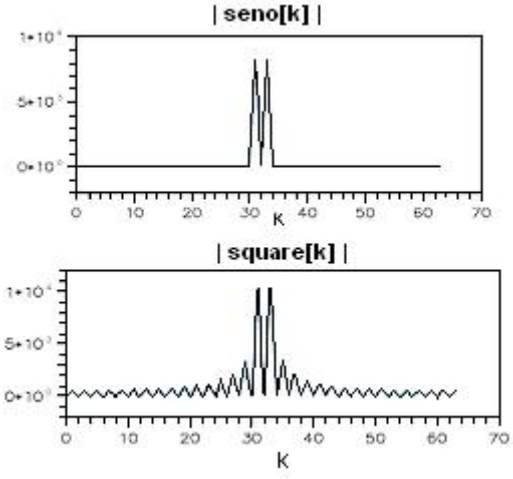
FIGURA 81. Parte real e imaginaria del espectro de la señal compleja de entrada a la FFT.



Fuente: El autor.

En la figura 82 se observa el tercer paso para calcular el espectro de dos señales con una sola FFT; en este punto del procedimiento se aplican las ecuaciones de la sección 1.2.4 del presente informe para separar el espectro de las dos señales originales.

FIGURA 82. Magnitud de los espectros de las dos señales originales después de correr la subrutina fft2senal1.

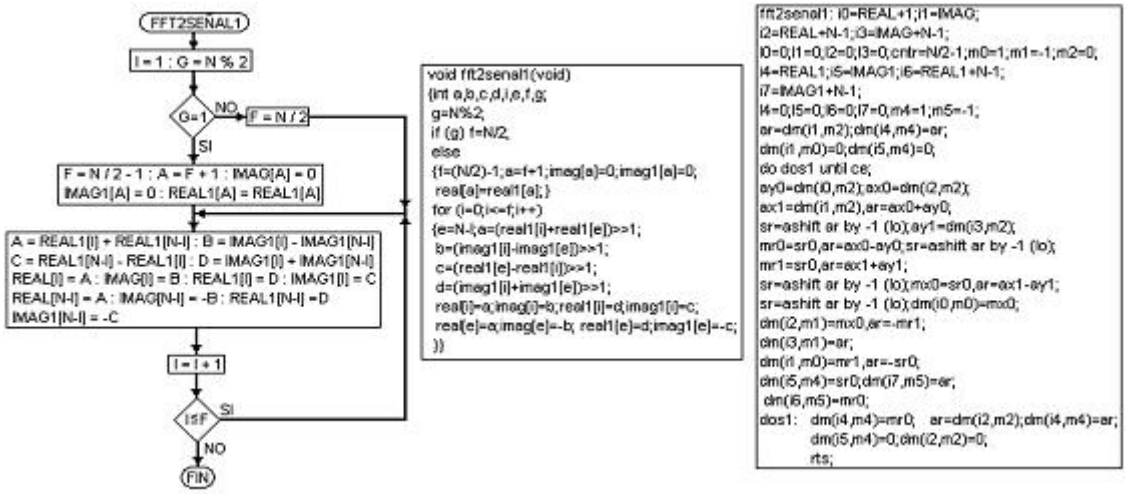


Fuente: El autor.

La figura 83 muestra el diagrama de flujo y el código en lenguaje ANSI C que permite extraer el espectro de dos señales reales a partir de una sola FFT. En este algoritmo se debe tener en cuenta para el diezmado entre etapas del algoritmo FFT, que las señales que se procesan son complejas; debido a esto se debe contemplar no el valor máximo que tiene la señal de entrada real sino la amplitud máxima que tenga la señal

compleja que se obtiene entre la señal de entrada real y la señal de entrada imaginaria; todo lo anterior para evitar el desbordamiento aritmético.

FIGURA 83. Diagrama de flujo y código en lenguaje C y Ensamblador del cálculo del espectro de dos señales reales utilizando la parte imaginaria de la señal de entrada.



Fuente: El autor.

4.5. ALGORITMO Y PROGRAMA PARA EL CÁLCULO DE LA TRANSFORMADA DE FOURIER DE UNA SEÑAL UTILIZANDO LA PARTE IMAGINARIA PARA ALMACENAR LA MITAD DE LA MISMA.

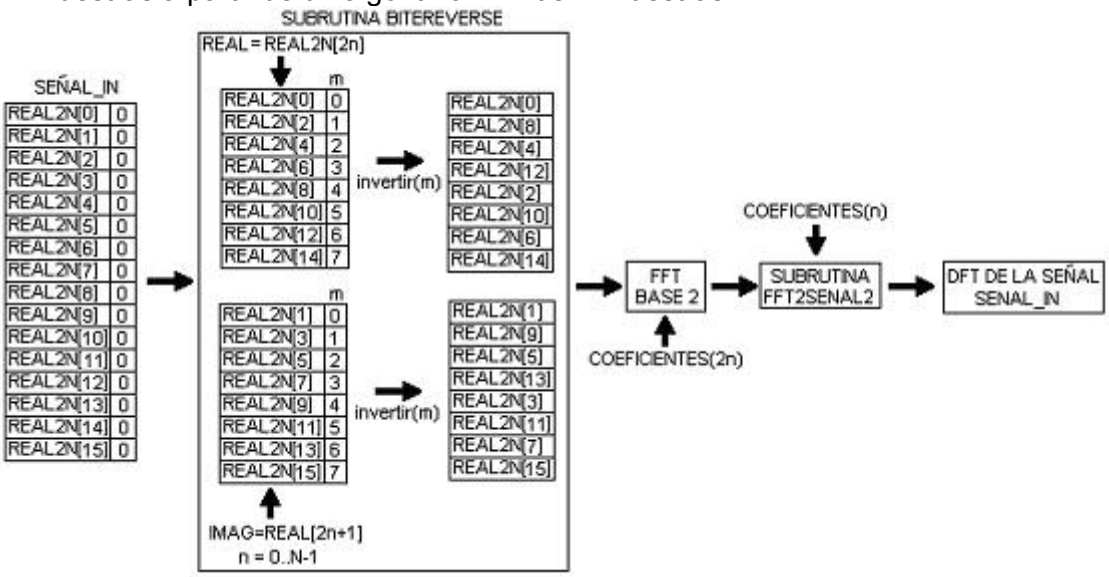
Otro algoritmo que se utiliza para aprovechar la parte imaginaria del vector de señal de entrada se implementa ubicando las muestras pares de una señal de entrada de 2N muestras en la parte real del vector de entrada de la FFT y las muestras impares en la parte imaginaria de la misma. Esta ubicación de los datos se puede implementar de manera conjunta con la inversión de *bits* inicial que se requiere en un algoritmo FFT de diezmado en el tiempo. Lo anterior evita tener que utilizar una subrutina adicional que calcule la inversión de *bits* al final del proceso lo cual aumentaría el número de ciclos empleados por el procesador para evaluar la DFT.

Otro aspecto que se debe tener en cuenta cuando se calcula la DFT mediante este algoritmo es la forma en que se leen los coeficientes cuando se calcula la DFT con el algoritmo FFT. Para esto se deben obtener los coeficientes de la señal de 2N muestras ya que en este vector están incluidos los coeficientes de una señal de N muestras; cuando se realiza la FFT de N muestras, los coeficientes que se deben tomar para evaluarla se toman del vector de 2N muestras pero de 2 en 2 coeficientes; lo anterior se logra en el algoritmo de la FFT mediante un desplazamiento a la izquierda de un *bit* en el puntero que accede a los elementos del vector de coeficientes. Una vez culminado este proceso se calcula la DFT de la señal de N

muestras mediante un algoritmo FFT de diezmado en el tiempo el cual obtiene los resultados de forma ordenada. Posteriormente se calcula la DFT de 2N muestras con la subrutina FFT2SENAL2 utilizando los elementos del vector de coeficientes de la señal de 2N muestras de uno en uno.

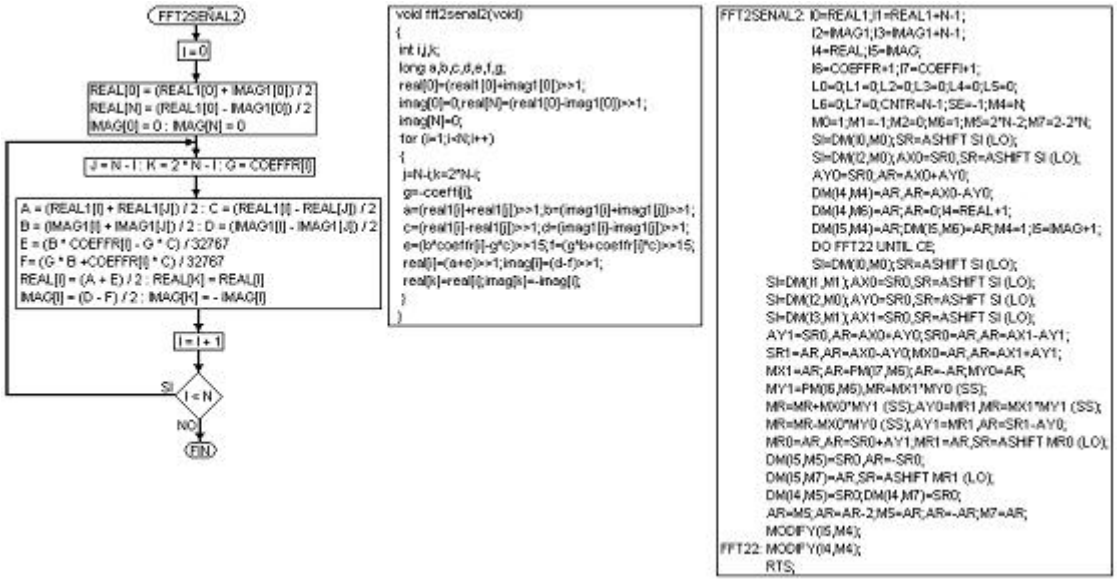
En la figura 84 se muestra el procedimiento computacional que se utiliza para calcular la DFT de una señal de 2N muestras. En la figura 85 se muestran el diagrama de flujo y el código en lenguaje C y lenguaje Ensamblador de la subrutina FFT2SEÑAL2.

FIGURA 84. Procedimiento computacional para obtener la DFT de una señal de 2N muestras a partir de un algoritmo FFT de N muestras.



Fuente: El autor.

FIGURA 85. Diagrama de flujo y programas para calcular la DFT de una señal de 2N muestras.



Fuente: El autor.

Para realizar una comparación de rendimiento computacional con la FFT en base 2 y la subrutina FFT2SEÑAL2 se deben tener en cuenta los siguientes aspectos: el número de ciclos de la FFT comparable con la subrutina FFT2SEÑAL2 se calculan sumando el número de ciclos de la FFT con el número de ciclos de la subrutina de inversión de *bits* de la dirección de los elementos del vector de entrada para una señal de 2N muestras; el número de ciclos que le toma al DSP calcular la DFT de una señal de 2N muestras con el algoritmo FFT2SEÑAL2, se calcula sumando el número de ciclos de la FFT de N muestras con el número de ciclos de la subrutina que invierte los *bits* y almacena la parte par e impar de la señal de entrada en los vectores real e imaginario respectivamente con el número de ciclos de la subrutina FFT2SEÑAL2. La tabla 17 muestra la comparación computacional para varios tamaños de muestra de la señal de entrada.

La expresión ciclos1 (ecuación 91) representa el número de ciclos que se emplean para calcular el espectro de una señal de 2N muestras, que requiere ejecutar una vez el algoritmo FFT de 2N muestras y una vez la subrutina que reordena los datos ya sea del vector de salida o del vector de entrada según el tipo de FFT que se ejecute. La expresión ciclos2 (ecuación 92) muestra el número de ciclos que se emplean al calcular el espectro de una señal de 2N muestras pero con el algoritmo FFT2SENAL2; en este caso los ciclos de la FFT y la subrutina de reordenamiento de los elementos del vector de salida de entrada se suman con los ciclos consumidos por FFT2SENAL2. Los ahorros computacionales al igual que en la subrutina FFT2SENAL1 se calculan restando ciclos1 de ciclos2.

$$ciclos_1 = ciclos(FFT_{2N}) + ciclos(BITREVERSE_{2N}) \quad (91)$$

$$ciclos_2 = ciclos(FFT_N) + ciclos(BITREVERSE_N) + ciclos(FFT2SEÑAL2) \quad (92)$$

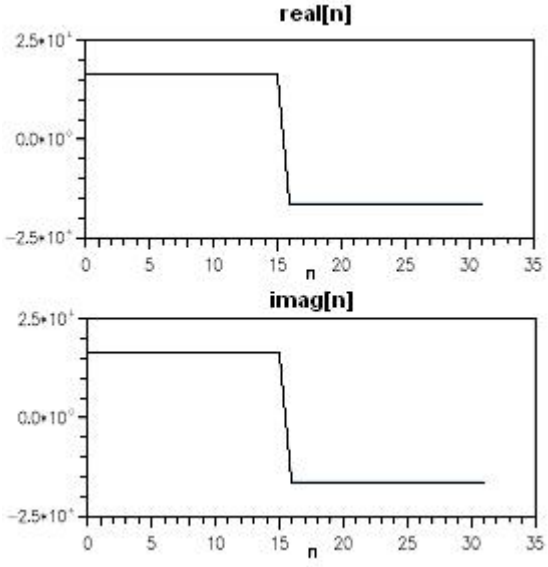
En la tabla 17 se muestran los resultados de la subrutina FFT2SENAL2 utilizando la subrutina FFT en base 2.

TABLA 17. Comparación computacional entre algoritmos FFT para señales de 2N muestras y la subrutina FFT2SEÑAL2.

2N	N	Ciclos1	Ciclos2	ahorros
32	16	22316	13795	8521
64	32	54601	32622	21979
128	64	128922	75499	53423
256	128	298038	171165	126873
512	256	660437	382810	277627
1024	512	1432814	830428	602386

La figura 86 muestra el inicio del algoritmo FFT2SENAL2 el cual consiste en separar los elementos con dirección par de la señal de entrada cuadrada de 64 muestras en el vector REAL de 32 muestras y los elementos con dirección impar en el vector IMAG también de 32 muestras.

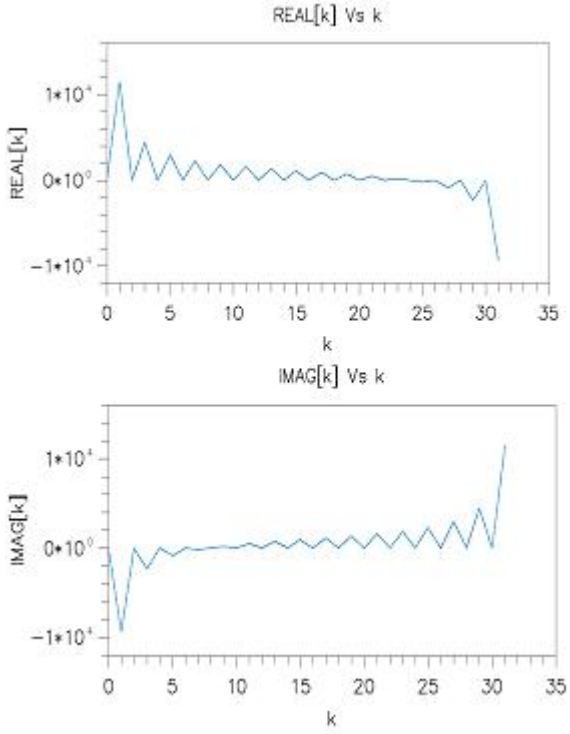
FIGURA 86. Señal original dividida en dos. Muestras pares en la señal real e impares en la señal imaginaria.



Fuente: El autor.

La figura 87 muestra el espectro de la señal compleja REAL+jIMAG calculado con el algoritmo FFT de diezmado en tiempo con N = 32 muestras, antes de llevar a cabo el computo anterior se desordenan los vectores REAL e IMAG con la subrutina de inversión de los elementos de los vectores de entrada.

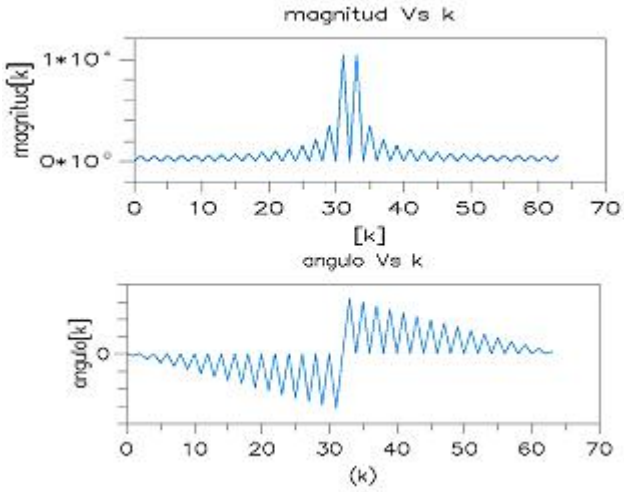
FIGURA 87. Espectro de la señal de entrada compleja N = 32.



Fuente: El autor.

La figura 88 muestra el espectro y la fase de la señal entrada de 64 muestras calculado con el algoritmo FFT2SENAL2.

FIGURA 88. Espectro de la señal original con N = 64 muestras.



Fuente: El autor.

4.6. REALIZACIÓN DEL ALGORITMO GOERTZEL PARA EVALUAR PARCIALMENTE EL ESPECTRO DE UNA SEÑAL.

Para realizar el algoritmo Goertzel, la ecuación 50 debe iterarse para valores de n=0, 1,…, N y la ecuación 51 se calcula una sola vez para cuando n=N. Cada repetición requiere de una multiplicación real y dos sumas. Este algoritmo se utiliza cuando M, parte del espectro que se desea calcular, es menor o igual a $\log_2 N$ si no se cumple esta relación es mejor utilizar el algoritmo de la FFT. Por este motivo el algoritmo Goertzel es útil para la decodificación de tonos de multifrecuencia (DTMF) de las llamadas telefónicas. A manera de ejemplo si N es igual a 8 muestras, entonces $\log_2 8$ es igual a 3 y por ende M ≤ 3.

Para comprobar el funcionamiento de la subrutina Goertzel el mismo se utiliza para detectar dos tonos transmitidos por una línea telefónica; la salida del detector de tonos son dos señales digitales de 0 a 5 V. La señal de salida 1 se pone en alto si el tono transmitido corresponde al primer armónico; se pondrá en bajo si el tono equivale al segundo armónico. La señal de salida 2 funciona de la misma forma pero se pone en alto cuando detecta el segundo armónico. En la figura 89 se observan las señales obtenidas en los dos canales de salida digital.

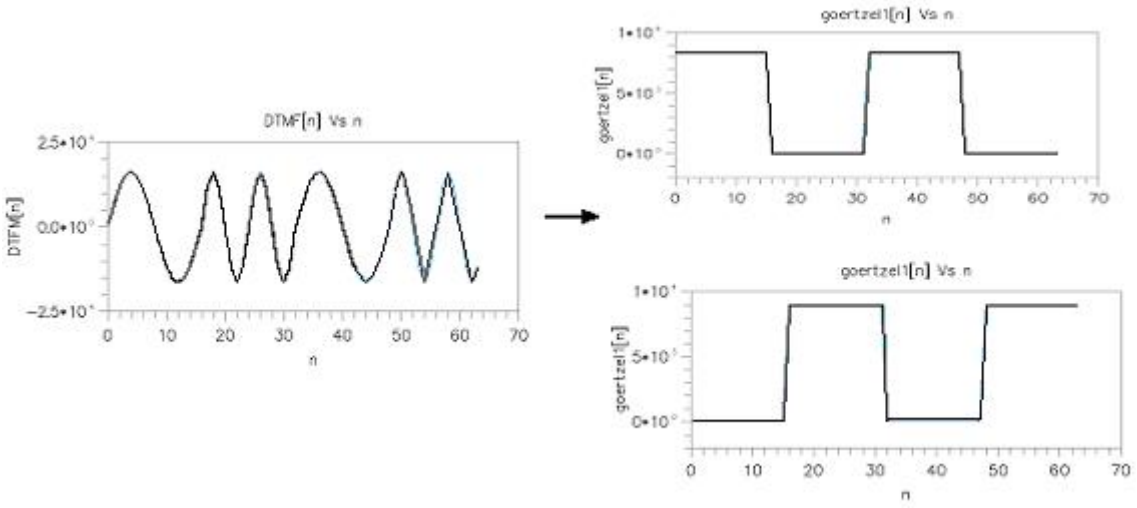
La tabla 18 se utiliza para comparar el rendimiento computacional del algoritmo Goertzel contra el algoritmo FFT en base 4 para 16 muestras (los programas bajo análisis fueron elaborados en lenguaje ensamblador). Como se puede observar en la tabla 18, el algoritmo Goertzel se puede utilizar para calcular hasta la sexta componente del espectro de la señal de entrada, después de este armónico es preferible utilizar el algoritmo FFT en base 4.

En la figura 90 se muestra el diagrama de flujo y el código en lenguaje C y Ensamblador del algoritmo Goertzel.

TABLA 18. Comparación computacional del algoritmo de Goertzel con la FFT en base 4 para un tamaño de muestras igual a 16.

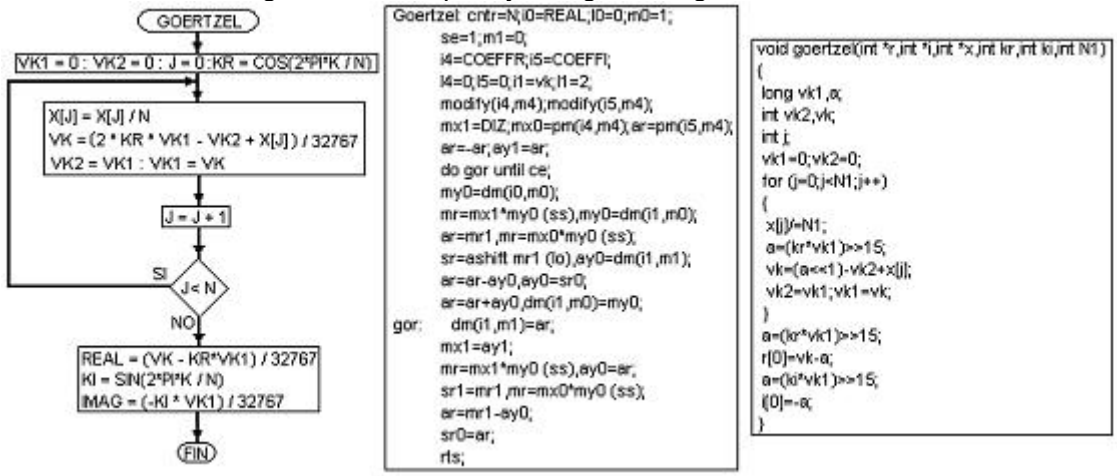
FFT en base 4 + DigReverse	Armónicos calculados con el algoritmo Goertzel	Ciclos con el algoritmo de Goertzel
	1	141
	2	282
	3	423
926	4	564
	5	705
	6	846
	7	**987**
	8	**1128**
	9	**1269**

FIGURA 89. Figura 89. Señal DTMF y salida de dos filtros detectores de los dos primeros armónicos.



Fuente: El autor.

FIGURA 90. Diagrama de bloques y código del algoritmo de Goertzel.



Fuente: El autor.

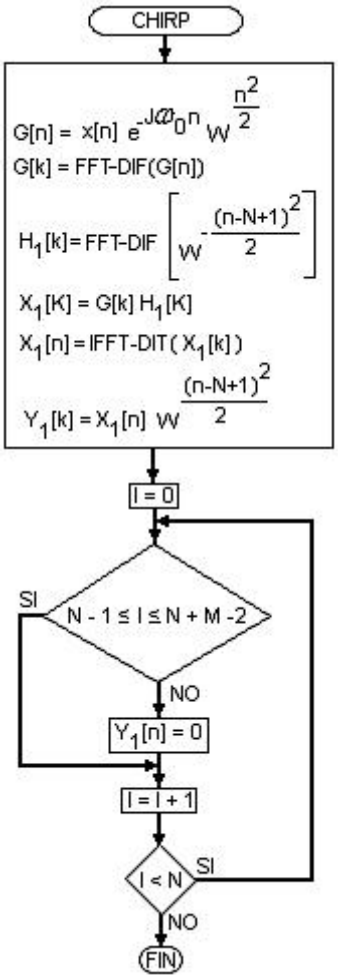
4.7. ALGORITMO CHIRP PARA EL CÁLCULO PARCIAL DEL ESPECTRO.

Para calcular el fragmento de espectro deseado con el algoritmo de la transformada CHIRP se deben seguir los siguientes pasos:

1. Multiplicar la señal de entrada por la señal $e^{-j\omega_0 n}W^{\frac{n^2}{2}}$ donde W es igual a $e^{-\frac{j2\pi}{\Delta\omega}}$, $\Delta\omega$ es la resolución en frecuencia deseada y ω_0 la frecuencia inicial del intervalo de frecuencias deseado. El resultado es la señal g[n].
2. Obtener la FFT de la señal g[n].
3. Calcular la FFT de la función de transferencia $h_1[n] = W^{-\frac{(n-N+1)^2}{2}}$ donde N es el número de muestras de la señal de entrada. El resultado es el espectro $H_1[k]$
4. Multiplicar G[k] con $H_1[k]$ para obtener el espectro $X_1[k]$.
5. Calcular la transformada inversa de Fourier del espectro $X_1[k]$ con lo cual se obtiene la señal $x_1[n]$.
6. Multiplicar la señal $x_1[n]$ con $W^{\frac{(n-N+1)^2}{2}}$ para obtener la señal $y_1[n]$. Esta señal equivale a las muestras del espectro desde la muestra N hasta la muestra N+M-2 donde M equivale al número de frecuencias que se evalúan.

La figura 91 muestra el procedimiento computacional para el cálculo de la transformada CHIRP.

FIGURA 91. Algoritmo de la transformada CHIRP.



Fuente: El autor.

El cálculo de los espectros se debe realizar con un algoritmo FFT de diezmado en frecuencia y el cálculo de la transformada inversa IFFT con un algoritmo de diezmado en tiempo para evitar las subrutinas de reordenamiento de datos. El primero recibe datos en orden normal y obtiene resultados en orden inverso, el segundo los recibe en orden inverso y obtiene una salida con los datos ordenados; además la transformada IFFT no necesita calcular los coeficientes multiplicativos ya que son los mismos valores, conjugados, del algoritmo FFT utilizado para calcular el espectro. Para el cálculo de la IFFT no se necesita diezmar la señal a medida que van transcurriendo las etapas del mismo. Lo anterior se puede ver en las ecuaciones 92 y 93.

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \quad \text{k = 0, 1,2,..., N-1} \tag{92}$$

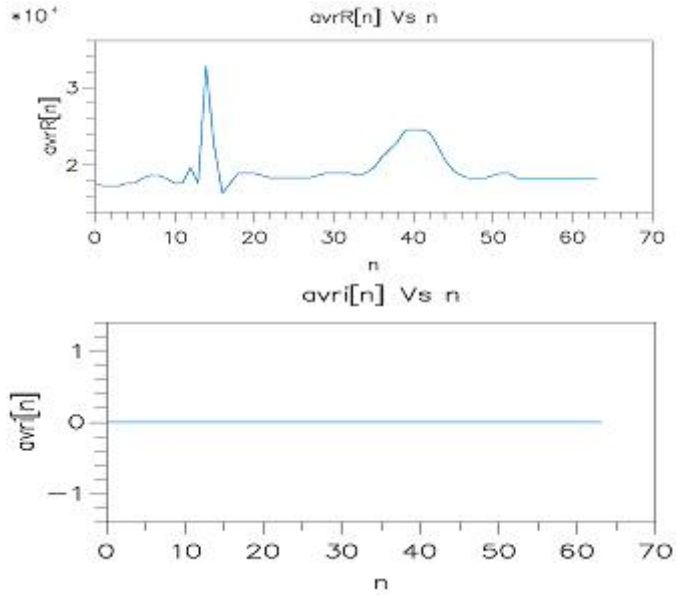
$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn} \quad \text{n = 0, 1, 2,..., N-1} \tag{93}$$

Para calcular la transformada CHIRP se requieren tres FFTs, una modulación y una demodulación; lo anterior limita su uso para situaciones en las cuales la resolución en frecuencia $\Delta\omega$ es pequeña, ya que para lograr este valor se necesitaría una FFT de gran tamaño. Como aclaración se mostrará a continuación un ejemplo: x[n] es una señal de longitud finita distinta de cero en el intervalo desde n=0 hasta n=25.

Se desea calcular 16 muestras de la transformada discreta de Fourier $X(\omega)$ en las frecuencias $\omega_k = \frac{2\pi}{27} + \frac{2\pi k}{1024}$ para k = 0, 1, 2,…., 15. De los datos del ejemplo se observa que $\Delta\omega = \frac{2\pi}{1024}$; para lograr esta resolución con la FFT de base 2 se debería utilizar un ancho de señal N igual a 1024 lo cual necesitaría $(\frac{1024}{2}) \times \log_2(1024)$ multiplicaciones y $1024 \times \log_2(1024)$ sumas lo cual daría un total de 15360 operaciones aritméticas. La transformada CHIRP tiene su dificultad principal en el cálculo de la convolución. Para llevar a cabo este procedimiento se necesita calcular la FFT de la señal de entrada g[n] y la FFT de la función de transferencia $h_1[n]$. La longitud de la función de transferencia $h_1[n]$ necesaria para evaluar adecuadamente la convolución utilizando la multiplicación de los espectros es igual a M+N-2 donde M es la longitud de las muestras en frecuencia que se desean obtener y N es la longitud de la señal de entrada. En el caso del ejemplo M es igual a 16 N es igual a 26 por lo tanto las FFTs necesitarían una longitud de 40 muestras. Si las FFTs se calculan con un algoritmo de FFT en base 2 se debe adoptar una longitud de 64 muestras; el espacio que sobra se rellena con ceros. Las tres FFTs requeridas de 64 muestras para el cálculo de la transformada CHIRP necesitarían $3 \times (\frac{64}{2}\log_2(64) + 64\log_2(64))$ lo cual da como resultado 1728 operaciones aritméticas para el cálculo de la convolución. Ahora para la modulación de la señal de entrada se requieren 64×4 multiplicaciones y 64×2 sumas lo cual da como resultado 384 operaciones. La demodulación de la señal de salida de la convolución necesita la misma cantidad de operaciones; los resultados anteriores se suman y dan como resultado el coste computacional de la transformada CHIRP. Al comparar la cantidad de operaciones de esta, 2112, con las requeridas por la FFT de 1024 muestras 15360 se obtiene una disminución del 86,25% en cuanto a operaciones aritméticas.

La figura 92 muestra la señal electrocardiográfica extractada de la derivación aVR con una longitud de 64 muestras como entrada al algoritmo CHIRP. De esta señal se desea calcular la magnitud de su espectro desde la frecuencia $\frac{2\pi}{27}$ hasta la frecuencia $\frac{50\pi}{1024}$.

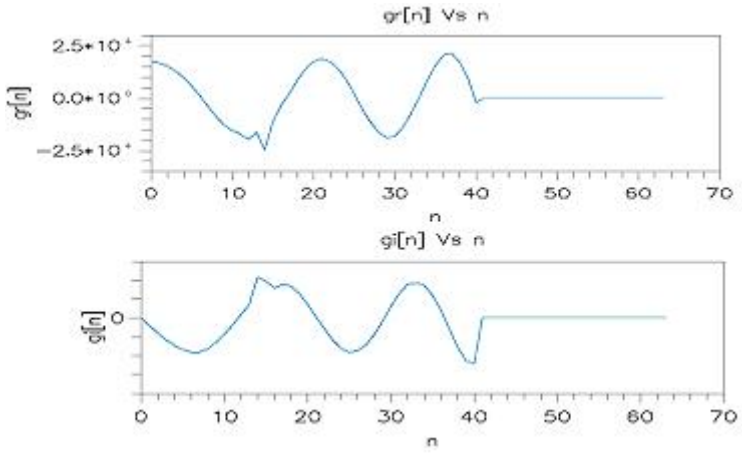
FIGURA 92. Señal de entrada avr[n].



Fuente: El autor.

La figura 93 muestra la modulación de la señal de entrada avr + j avi por la señal $e^{-j\omega_0 n}W^{\frac{n^2}{2}}$ con lo cual se obtiene la señal g[n].

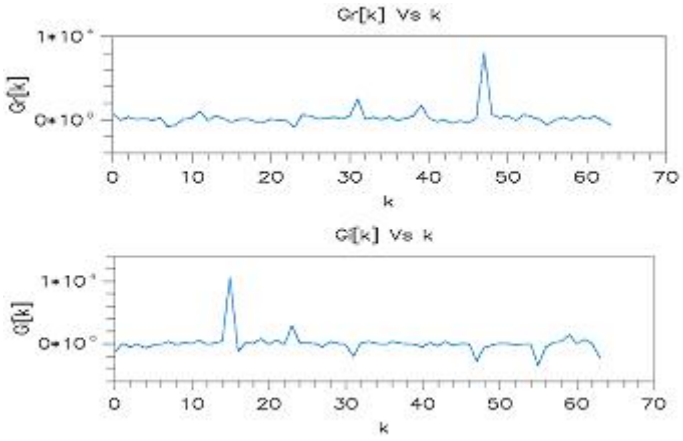
FIGURA 93. Señal $g[n] = avr[n] \times e^{-jw_0 n}W^{\frac{n^2}{2}}$.



Fuente: El autor.

La figura 94 muestra el espectro G[k] de la señal g[n], ya que, la señal g[n] tiene 40 muestras se necesita para el cálculo del mismo una FFT de 64 muestras. El algoritmo FFT utilizado es de diezmado en frecuencia en base 2. el resultado del espectro no se debe reordenar porque en una etapa posterior del algoritmo se necesita ejecutar la IFFT con un algoritmo de diezmado en tiempo que necesita los datos de entrada en forma inversa.

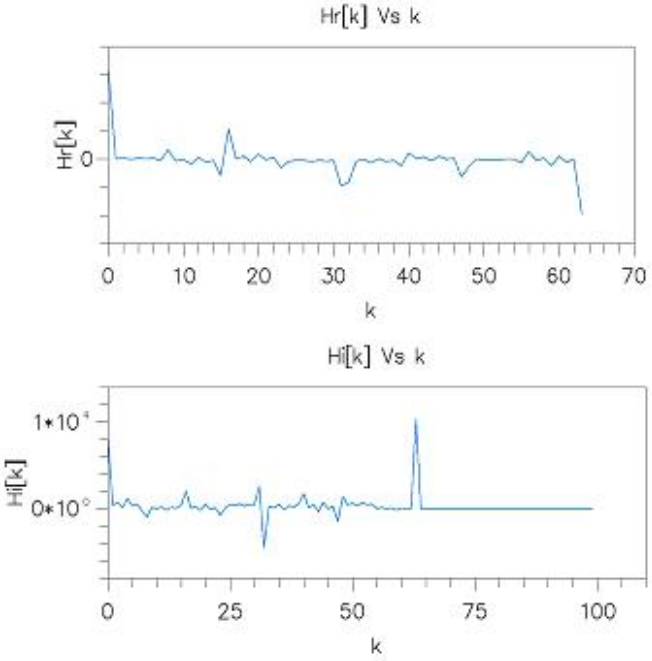
FIGURA 94. Espectro G[k] de la señal g[n].



Fuente: El autor.

La figura 95 muestra la DFT $H_1[k]$ de la función de transferencia $h_1[n]$. El cálculo se lleva a cabo con una FFT de 64 muestras de diezmado en frecuencia.

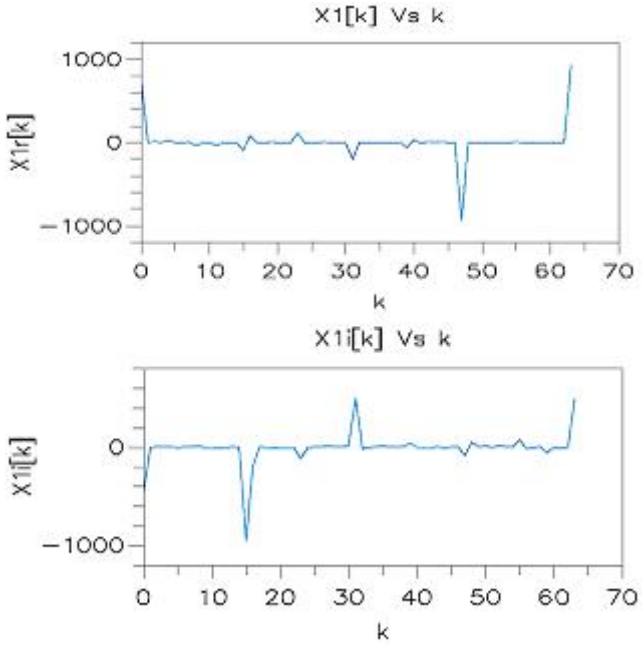
FIGURA 95. Espectro $H_1[k]$ de la función de transferencia $h_1[n]$.



Fuente: El autor.

Las DFTs $H_1[k]$ y $G[k]$ se multiplican para obtener la DFT $X_1[k]$ que se observa en la figura 96.

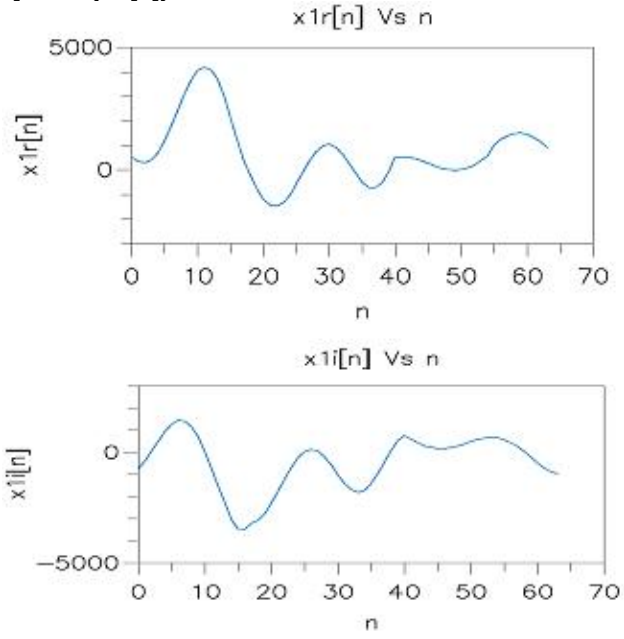
FIGURA 96. $X_1[k] = H_1[k] \times G[k]$.



Fuente: El autor.

La figura 97 muestra la transformada inversa de la DFT $X_1[k]$. Este cálculo se realiza con un algoritmo FFT de diezmado en el tiempo. Lo anterior se debe a que $X_1[k]$ se encuentra con los elementos invertidos por el cálculo previo con FFT de diezmado en frecuencia.

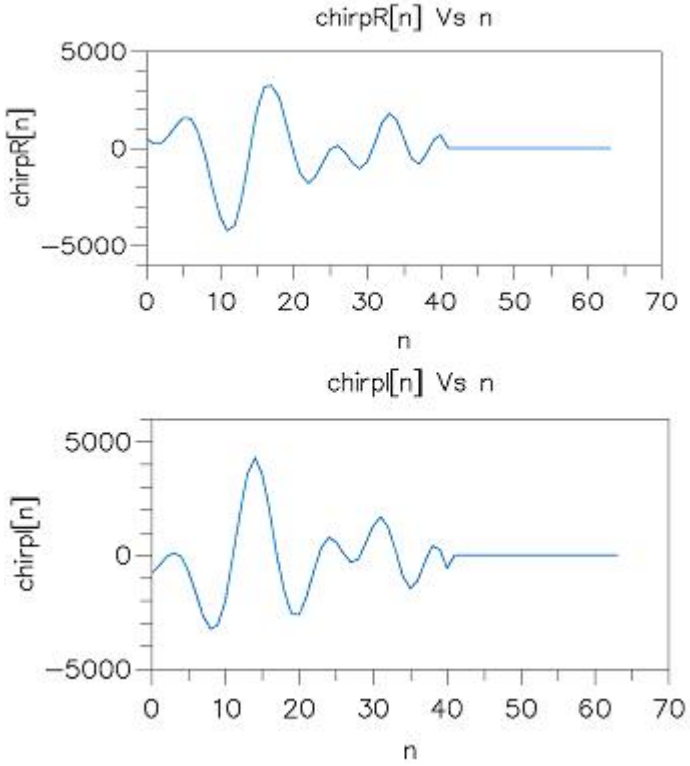
FIGURA 97. $x_1[n] = F^{-1}\{X_1[k]\}$.



Fuente: El autor.

En la figura 98 se obtiene la señal CHIRP[n] como la demodulación de la señal $x_1[n]$ con la señal compleja $W^{\frac{(n-N+1)^2}{2}}$.

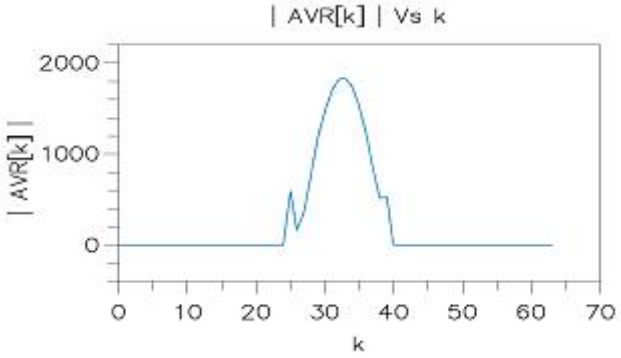
FIGURA 98. Señal $chirp[n] = x_1[n] \times W^{\frac{(n-N+1)^2}{2}}$.



Fuente: El autor.

Finalmente en la figura 99 se muestra la magnitud del espectro en el rango de frecuencias deseados.

FIGURA 99. Magnitud del espectro de la señal avr[n] en el intervalo de frecuencias $\frac{2\pi}{27} \le \omega \le \frac{2\pi(1429)}{27648}$ con $\Delta\omega = \frac{2\pi}{1024}$.



Fuente: El autor.

El código en lenguaje C de la transformada CHIRP se observa en las figuras 100 y 101.

FIGURA 100. Código de la transformada CHIRP en lenguaje C. Parte A.

```
#include <math.h>
#define N 64
#define ETAPAS 6
#define N1 26
#define M 16
int dm real[N] = {
          #include "real.dat"
          };
int dm imag[N] = {
           #include "imag.dat"
          };
int pm coeffr[N] = {
           #include "coeffr.dat"

           };
int pm coeffi[N] = {
           #include "coeffi.dat"
           };
int dm er[N] = {
          #include "ereal.dat"
          };
int dm ei[N] = {
           #include "eimag.dat"
          };
int dm hr[N] = {
          #include "hreal.dat"
          };
int dm hi[N] = {
           #include "himag.dat"
          };
int ESPECTRO[N];
void IFFT(int *,int *);
void FFT(int *,int *);
long magnitud(long,long);
void calc_espectro(int *,int *);
void multiplicar(int *,int *,int *,int *,int *);
void chirp(void);
main(void)
{
 FFT(hr,hi);
 multiplicar(real,imag,er,ei,N);
 FFT(real,imag);
 multiplicar(real,imag,hr,hi,N);
 IFFT(real,imag);
 multiplicar(real,imag,wr,wi,N);
 chirp ();
 calc_espectro(real,imag);
 return 0;
}
long magnitud(long RE,long IM)
{ return (sqrt(RE*RE+IM*IM));}
```

```
void calc_espectro(int *r1,int *i1)
{ int i;
 for(i=0;i<N;i++)
 ESPECTRO[i]=magnitud(r1[i],i1[i]);}
void multiplicar(int *r1,int *i1,int *r2,int *i2,int N11)
{
 int i;
 long a,b,c,d;
 for (i=0;i<N11;i++)
 {a=r1[i]; b=i1[i];c=(a*r2[i]-b*i2[i])>>15;
  d=(a*i2[i]+b*r2[i])>>15;r1[i]=c;i1[i]=d;}}
void chirp(void)
{ int i;
 for (i=0;i<N;i++)
 { if (i<N1-1||i>=N1+M-2) {real[i]=0;imag[i]=0;}}}
void IFFT(int *r1,int *i1)
{
 int i,j,k,grupos,mariposas,s1,j1,e,gr;
 long a,b,ar,br;
 mariposas=1; grupos=N>>1;
 for (i=0;i<ETAPAS;i++)
 {
  for (j=0;j<grupos;j++)
  {j1=2*j*mariposas;
   for (k=0;k<mariposas;k++)
   {
    e=k+j1;s1=e+mariposas;
              if (k)
    {
     gr=(N-grupos*k);a=r1[s1];b=i1[s1];
     ar=(a*coeffr[gr]-b*coeffi[gr])>>15;
     br=(a*coeffi[gr]+b*coeffr[gr])>>15;
     r1[s1]=ar;i1[s1]=br;
    }
    a=r1[e]-r1[s1];b=i1[e]-i1[s1];r1[e]+=r1[s1];
    i1[e]+=i1[s1];r1[s1]=a;i1[s1]=b;
   }}
  mariposas<<=1;grupos>>=1;
 }}
void FFT(int *r1,int *i1)
{
 int i,j,k,grupos,mariposas,s1,j1,e,gr;
 long a,b,ar,br;
 mariposas=N>>1;grupos=1;
 for (i=0;i<ETAPAS;i++)
 {for (j=0;j<grupos;j++){
 j1=2*j*mariposas;for (k=0;k<mariposas;k++){
 e=k+j1;s1=e+mariposas;
r1[e]>>=1;i1[e]>>=1;r1[s1]>>=1;i1[s1]>>=1;
 a=r1[e]-r1[s1];b=i1[e]-i1[s1];r1[e]+=r1[s1];
i1[e]+=i1[s1];r1[s1]=a;i1[s1]=b;
if (k){ gr=grupos*k;a=r1[s1];b=i1[s1];
ar=(a*coeffr[gr]-b*coeffi[gr])>>15;
br=(a*coeffi[gr]+b*coeffr[gr])>>15;
r1[s1]=ar;i1[s1]=br;}}}
mariposas>>=1;grupos<<=1;}}
```

Fuente: El autor.

FIGURA 101. Código de la transformada CHIRP en lenguaje C. Parte B.

```
#include <math.h>
#define N 64
#define ETAPAS 6
#define N1 26
#define M 16
int dm real[N] = {
        #include "real.dat"
        };
int dm imag[N] = {
        #include "imag.dat"
        };
int pm coeffr[N] = {
        #include "coeffr.dat"
        };
int pm coeffi[N] = {
        #include "coeffi.dat"
        };
int dm er[N] = {
        #include "ereal.dat"
        };
int dm ei[N] = {
        #include "eimag.dat"
        };
int dm vr[N] = {
        #include "vreal.dat"
        };
int dm vi[N] = {
        #include "eimag.dat"
        };
int dm hr[N] = {
        #include "hreal.dat"
        };
int dm hi[N] = {
        #include "himag.dat"
        };
int ESPECTRO[N];
void IFFT(int *,int *);
void FFT(int *,int *);
long magnitud(long,long);
void calc_espectro(int *,int *);
void multiplicar(int *,int *,int *,int *,int);
void chirp(void);
main(void)
{
 FFT(hr,hi);
 multiplicar(real,imag,er,ei,N);
 FFT(real,imag);
 multiplicar(real,imag,hr,hi,N);
 IFFT(real,imag);
 multiplicar(real,imag,vr,vi,N);
 chirp();
 calc_espectro(real,imag);
 return 0;
}
long magnitud(long RE,long IM)
{ return (sqrt(RE*RE+IM*IM));}
```

```
void calc_espectro(int *r1,int *i1)
{ int i;
 for(i=0;i<N;i++)
ESPECTRO[i]=magnitud(r1[i],i1[i]);}
void multiplicar(int *r1,int *i1,int *r2,int *i2,int N11)
{
 int i;
 long a,b,c,d;
 for (i=0;i<N11;i++)
 {a=r1[i]; b=i1[i];c=(a*i2[i]-b*i2[i])>>15;
  d=(a*i2[i]+b*r2[i])>>15;r1[i]=c;i1[i]=d;}}
void chirp(void)
{ int i;
 for (i=0;i<N;i++)
 { if (i<N1-1||i>=N1+M-2) {real[i]=0;imag[i]=0;}}}
void IFFT(int *r1,int *i1)
{
 int i,j,k,grupos,mariposas,e,l,s1,gr;
 long a,b,ar,br;
 mariposas=1; grupos=N>>1;
 for (i=0;i<ETAPAS;i++)
 {
  for (j=0;j<grupos;j++)
  {l=2*j*mariposas;
   for (k=0;k<mariposas;k++)
   {
    e=k+l;s1=e+mariposas;
          if (k)
    {
     gr=(N-grupos*k);a=r1[s1];b=i1[s1];
     ar=(a*coeffr[gr]-b*coeffi[gr])>>15;
     br=(a*coeffi[gr]+b*coeffr[gr])>>15;
     r1[s1]=ar;i1[s1]=br;
    }
    a=r1[e]-r1[s1];b=i1[e]-i1[s1];r1[e]+=r1[s1];
    i1[e]+=i1[s1];r1[s1]=a;i1[s1]=b;
   }}
  mariposas<<=1;grupos>>=1;
 }}
void FFT(int *r1,int *i1)
{
 int i,j,k,grupos,mariposas,s1,l,e,gr;
 long a,b,ar,br;
 mariposas=N>>1;grupos=1;
 for (i=0;i<ETAPAS;i++)
 {for (j=0;j<grupos;j++){
 l=2*j*mariposas;for (k=0;k<mariposas;k++){
 e=k+l;s1=e+mariposas;
 r1[e]>>=1;i1[e]>>=1;r1[s1]>>=1;i1[s1]>>=1;
 a=r1[e]-r1[s1];b=i1[e]-i1[s1];r1[e]+=r1[s1];
i1[e]+=i1[s1];r1[s1]=a;i1[s1]=b;
if (k){ gr=grupos*k;a=r1[s1];b=i1[s1];
ar=(a*coeffr[gr]+b*coeffi[gr])>>15;
br=(a*coeffi[gr]-b*coeffr[gr])>>15;
r1[s1]=ar;i1[s1]=br;}}}
mariposas>>=1;grupos<<=1;}}
```

Fuente: El autor.

La tabla 19 muestra los ciclos obtenidos para diferentes tamaños de señal y diferente cantidad de muestras del espectro en frecuencia de la misma. Se puede inferir de estos resultados que la complejidad computacional esta muy ligada a la cantidad de ciclos que emplea el algoritmo FFT para obtener los espectros de la señal modulada y el filtro $h_1[n]$ y la IFFT para obtener la señal de salida que debe ser demodulada para obtener las muestras deseadas del espectro de la señal de entrada. A medida que se requiere una cantidad de muestras mayor en el rango del espectro que se desea obtener se torna ineficiente el uso de la transformada CHIRP para el cálculo parcial del espectro de una señal de entrada. A continuación se explica cada una de las columnas de la tabla 19: Tamaño de la FFT base 2 tiene que ver con la resolución que se

requiere en el rango de frecuencias que se desea calcular; ciclos FFT base 2 hace referencia al número de ciclos que se requieren para obtener la resolución deseada con un algoritmo FFT; muestras del espectro (M) es la diferencia entre la frecuencia mayor y la frecuencia menor del rango de frecuencia que se desea calcular; tamaño de la señal (N) tiene que ver con el número de muestras de la señal que se desea procesar; FFT necesaria para chirp hace referencia a el tamaño de las FFTs que se utilizaran para el cálculo de las DFTs y de las transformadas inversas del algoritmo CHIRP; ciclos CHIRP como se observo en el ejemplo gráfico se requieren tres FFTs para calcular la señal CHIRP, por lo tanto esta columna esta dominada por la expresión 3*ciclos(FFT necesaria para CHIRP).

TABLA 19. Ciclos computacionales empleados por el procedimiento algorítmico CHIRP para diferentes tamaños de muestra (N) y diferente tamaño de muestras del espectro (M).

Tamaño de FFT base 2	Ciclos FFT base 2	Muestras del espectro (M)	Tamaño de la señal (N)	FFT necesaria para CHIRP	Ciclos CHIRP
1024	151799	16	26	64	19814
1024	151799	80	100	256	100061
1024	151799	150	200	512	216253

5. LABORATORIOS DE PRUEBAS PARA LA TARJETA ADAPTABLE.

Este capítulo del trabajo de investigación tiene como objeto experimentar con la tarjeta adaptable y poner en práctica los conceptos fundamentales de la tecnología del procesador de señales digitales DSP que los distinguen de otros procesadores tales como los microprocesadores y microcontroladores; para lograr este propósito se diseñaron tres prácticas distribuidas de la siguiente forma:

- Ensayos preliminares. Experimenta con los componentes de la tarjeta adaptable: puertos digitales, filtro antisolapamiento ajustable, convertidores analógicos-digitales y digitales-analógicos.
- Filtros digitales tipo FIR e IIR.
- Prácticas con los algoritmos FFT, Goertzel y CHIRP.

5.1. ENSAYOS PRELIMINARES DE LA TARJETA ADAPTABLE.

5.1.1. Objetivos.

- Diseñar y construir un circuito electrónico para controlar un motor paso a paso unipolar que permita comprobar el funcionamiento de los puertos digitales de la tarjeta adaptable.
- Construir un generador de señales con base en los puertos digitales y los canales de salida analógica de la tarjeta adaptable.
- Comprobar el funcionamiento de los filtros antisolapamiento ajustables.
- Observar con un osciloscopio el funcionamiento de los canales de entrada analógica de la tarjeta adaptable.

5.1.2. Lecturas recomendadas.

- Consultar las principales características de los circuitos integrados 74HC14, AD7398, AD974, AD5204 y AD8534 (Fairchild y Analog Devices) .
- Describir el funcionamiento de un motor paso a paso, su característica de par contra velocidad y las clases de motores paso a paso que se pueden encontrar.
- Profundizar en el funcionamiento de los vectores circulares y las interrupciones provocadas por el temporizador y el pulsador INTERRUPT de la tarjeta EZKIT2181.
- Describir el comportamiento de las banderas configurables de la tarjeta EZKIT2181. (PF_0….PF_7).

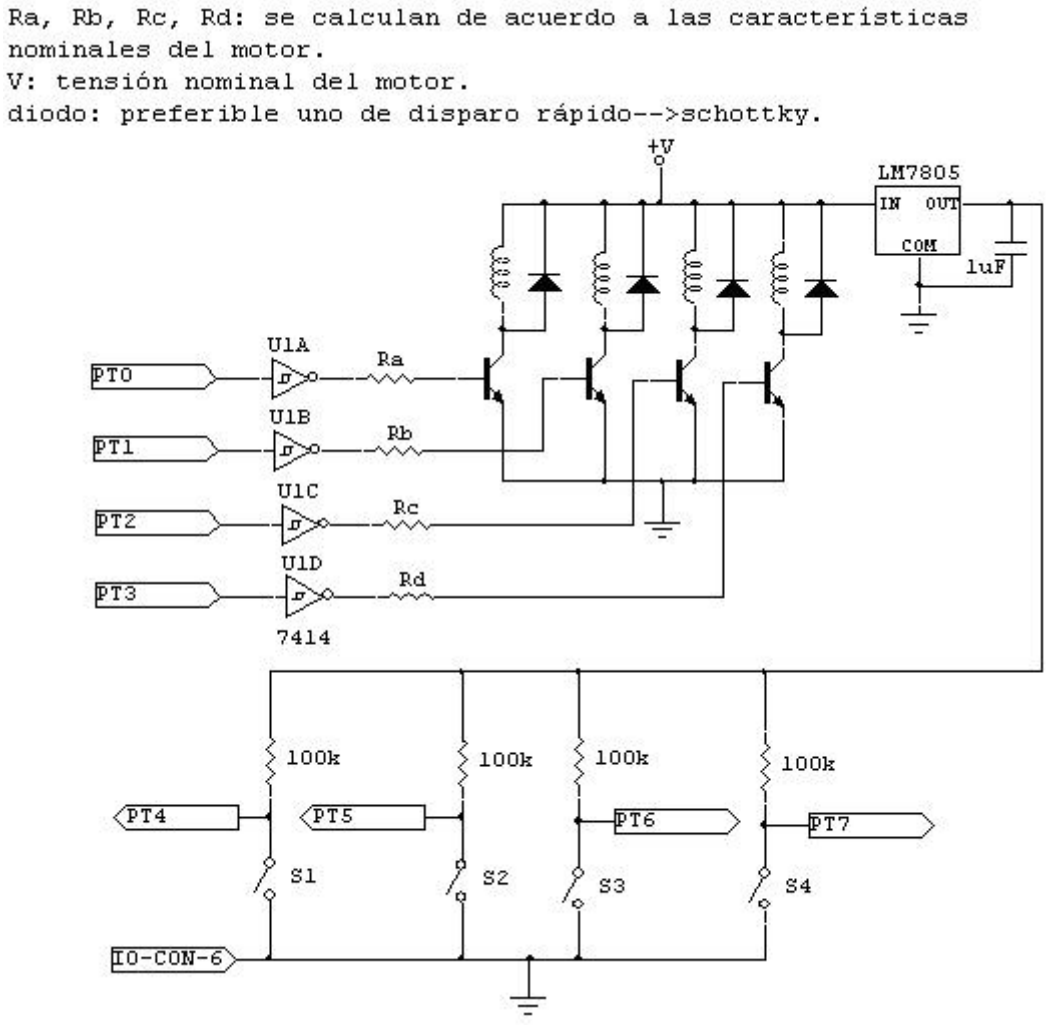
5.1.3. Elementos.

- Motor paso a paso unipolar.
- Circuito integrado 74HC14.
- Generador de señales.
- Osciloscopio.
- Interruptores.
- Resistencias.
- Voltímetro.

5.1.4. Procedimiento.

- Construir el circuito de la figura 102.

FIGURA 102. Circuito para controlar un motor paso a paso unipolar desde la tarjeta adaptable.



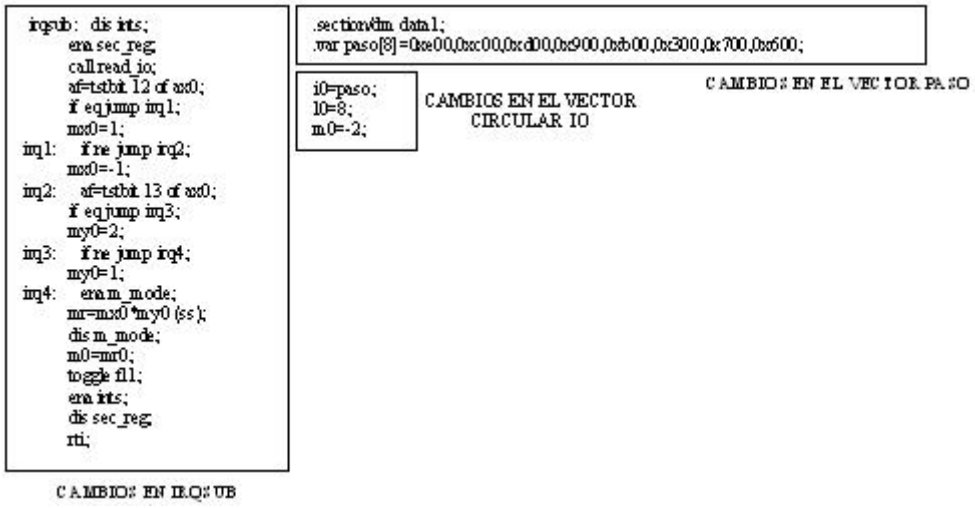
Fuente: El autor. Elaborado con el *software* CIRCUITMAKER.

- Ejecutar el programa VISUALDSP++ y en el menú PROJECT seleccionar la opción OPEN PROJECT y seleccionar la carpeta con el nombre lab_io1 que se encuentra en la dirección **c:\archivos de programa\analog devices\visualdsp\218x\examples\lab_io y seleccione lab_io1**.
- En el menú SESSION seleccionar la opción NEW SESSION y escoger la opción EZ-KIT 218x en la lista DEBUG TARGET y en PROCCESOR seleccione ADSP-2181.
- Si todo se ha realizado bien deberá aparecer una ventana donde se le pide al usuario oprimir el botón RESET de la tarjeta EZKIT2181. Despues de presionar el pulsador de RESET el diodo LED rojo dejará de parpadear quedando encendido indicando que la tarjeta se ha comunicado adecuadamente con el computador.
- Conectar el cable de comunicación serial que interconecta la tarjeta EZKIT2181 con la tarjeta adaptable a través del conector nombrado como DSP-CON.
- Comunicar la tarjeta adaptable con el circuito de la figura 102 a través del conector nombrado como IO-CON de la tarjeta adaptable.
- En el menú DEBUG seleccione la opción RUN. Observe y anote lo sucedido en el motor paso a paso.
- Cambie la posición del interruptor S_1 y presione el pulsador INTERRUPT de la tarjeta EZKIT2181. Observe y anote lo sucedido con el motor.
- Observe la subrutina CONFIG_TIM. ¿Cada cuantos milisegundos se produce una interrupción por temporizador, si el reloj que lo sincroniza tiene una frecuencia de 33 Mhz? **Rta: 1 milisegundo.**
- ¿Cada vez que es interrumpido el DSP a través del temporizador cual es la subrutina que se ejecuta? ¿Cual se ejecuta cuando se interrumpe con el pulsador INTERRUPT de la tarjeta EZKIT2181? **Rta: TIMSUB e IRQSUB.**
- Observe la subrutina CONFIG_PF. ¿Que función cumple esta subrutina? **Rta: configura como entradas o salidas los pines de banderas configurables de la tarjeta EZKIT2181.**
- ¿En la subrutina TIMSUB, que funciones cumplen las variables VELO y AUX? **Rta: AUX es una variable que cuenta el número de veces que el temporizador del ADSP2181 interrumpe la CPU. Cuando AUX es igual a VELO el programa ejecuta los cambios correspondientes en el motor paso a paso.**
- ¿En la subrutina IRQSUB que función cumple la variable m0? (clave: recuerde el funcionamiento de los vectores circulares del DSP 2181). **Rta: se utiliza**

para cambiar la dirección de avance del vector circular paso[4], el cual regula el funcionamiento del motor paso a paso.

- ¿Como funciona la subrutina motor? **Rta: si M_0 es igual a 1 el vector circular se incrementa y selecciona la siguiente posición del vector circular I_0, si M_0 es igual a -1 la siguiente posición será la anterior del vector circular I_0.**
- En la línea 68 del programa lab_io cambie la linea ar=0x32 por la línea de código ar=0x19. compare en un osciloscopio las dos formas de onda antes y despues que se producen en el pin PT0 del conector IO-CON. ¿De acuerdo a lo observado que puede concluir con respecto a la variable VELO? **Rta: controla la velocidad del motor paso a paso.**
- Rediseñar el programa para que el motor pueda avanzar con pasos medios y completos. **Rta: ver figura 103**.

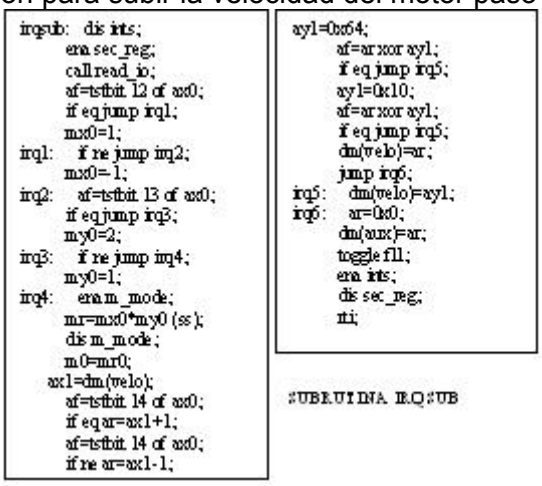
FIGURA 103. solución para pasos medios y completos.



Fuente: El autor.

- Rediseñar el programa, conservando las características anteriores, para que el motor pueda aumentar la velocidad en un rango de frecuencia de alimentación entre 1000 y 10 Hz. Rta: ver figura 104.

FIGURA 104. Solución para subir la velocidad del motor paso a paso.



```
irqsub:  dis ints;
         ena sec_reg;
         call read_io;
         af=tstbit 12 of ax0;
         if eq jump irq1;
         mx0=1;
irq1:     if ne jump irq2;
         mx0=-1;
irq2:     af=tstbit 13 of ax0;
         if eq jump irq3;
         my0=2;
irq3:     if ne jump irq4;
         my0=1;
irq4:     ena m_mode;
         mr=mx0*my0 (ss);
         dis m_mode;
         m0=mr0;
       ax1=dm(velo);
         af=tstbit 14 of ax0;
         if eq ar=ax1+1;
         af=tstbit 14 of ax0;
         if ne ar=ax1-1;
```

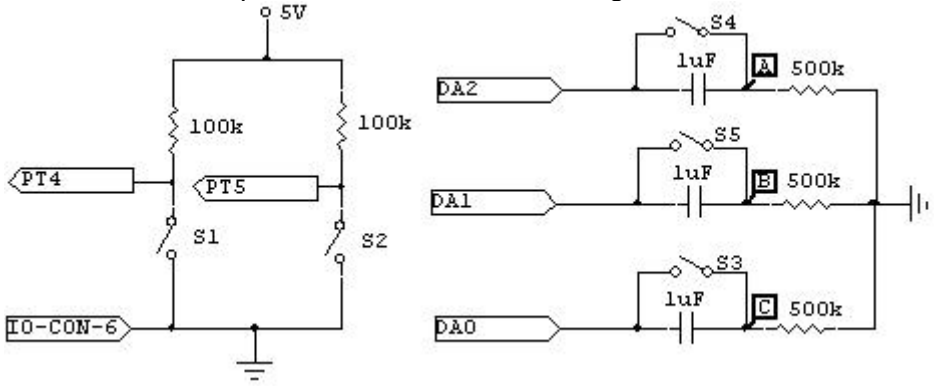
```
ay1=0x64;
         af=ar xor ay1;
         if eq jump irq5;
         ay1=0x10;
         af=ar xor ay1;
         if eq jump irq5;
         dm(velo)=ar;
         jump irq6;
irq5:     dm(velo)=ay1;
irq6:     ar=0x0;
         dm(aux)=ar;
         toggle fl1;
         ena ints;
         dis sec_reg;
         rti;
```

SUBRUTINA IRQSUB

Fuente: El autor.

- Construir el circuito de la figura 105.

FIGURA 105. Circuito que evalúa los canales analógicos de salida.

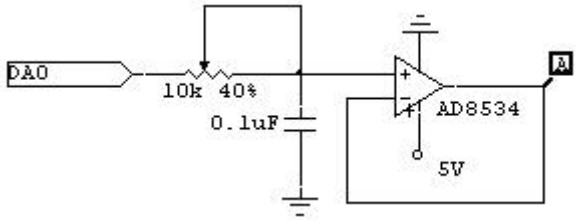


Fuente: El autor. Elaborado con el *software* CIRCUITMAKER.

- Abrir el programa VISUALDSP++ y en el menú PROJECT seleccione la opción OPEN PROJECT y escoja la carpeta con el nombre lab_da1 que se encuentra en la dirección **c:\archivos de programa\analog devices\visualdsp\218x\examples\lab_io y seleccione lab_da1**.
- En el menú SESSION seleccione la opción NEW SESSION y seleccione la opción EZ-KIT 218x en la lista DEBUG TARGET y en PROCCESOR seleccione ADSP-2181.
- Si todo se ha realizado bien deberá aparecer una ventana donde se le solicita al usuario oprimir el botón RESET de la tarjeta EZKIT2181. Después de presionar el pulsador de RESET el diodo LED rojo dejará de parpadear quedando encendido indicando que la tarjeta se ha comunicado adecuadamente con el computador.

- Conectar el cable de comunicación serial que interconecta la tarjeta EZKIT2181 con la tarjeta adaptable a través del conector nombrado como DSP-CON.
- Comunicar la tarjeta adaptable con el circuito de la figura 105 a través de los conectores nombrados como DA-CON e IO-CON de la tarjeta adaptable.
- En el menú DEBUG seleccione la opción RUN. Coloque la punta del osciloscopio en el terminal C de la figura 105. Describa la señal observada en el mismo.
- Cierre el interruptor S_3 y describa la señal observada en el osciloscopio. ¿Que función cumple la red R-C? **Rta: en la configuración dada actua como un filtro pasaaltas el cual elimina la componente DC de la señal cuando el interruptor esta abierto; si se encuentra cerrado deja pasar las componentes de alterna y de continua provenientes del canal analógico de salida seleccionado.**
- Deduzca una expresión matemática para la señal observada en la pantalla del osciloscopio. **Rta:** 2,5 + $\sin(\frac{\pi n}{8})$, donde $0 \le n \le 15$.
- En el programa, subrutina config_tim, obtenga el valor de la frecuencia de muestreo y comparela con la obtenida en el osciloscopio. Construya un circuito que elimine el rizado inherente a la conversion digital-analógica. **Rta: ver circuito de la figura 106. Frecuencia de muestreo se obtiene en la subrutina config_tim; F_s es igual 960 Hz, la señal tiene una frecuencia de 60 Hz y un pico de 5 V.**

FIGURA 106. Circuito que elimina el rizado de la conversión digital-analógica.

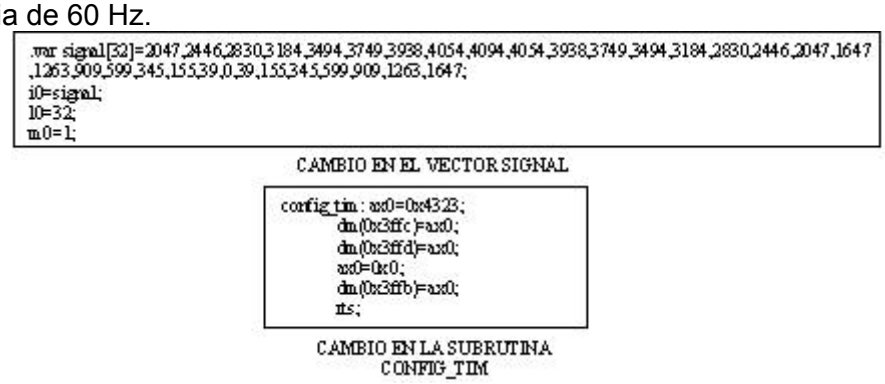


Fuente: El autor. Elaborado con el *software* CIRCUITMAKER.

- Cambiar el número de elementos del vector **signal** de 16 a 32 muestras conservando la misma forma de onda (ver figura 107). ¿Cual debería ser la nueva frecuencia de muestreo para mantener el periodo de la onda anteriormente vista en el osciloscopio? ¿Cual sería la nueva expresión

matemática que representa la señal observada en el osciloscopio? **Rta: 1920 Hz.** $2{,}5 + \sin(\frac{\pi n}{16})$ donde $0 \le n \le 31$.

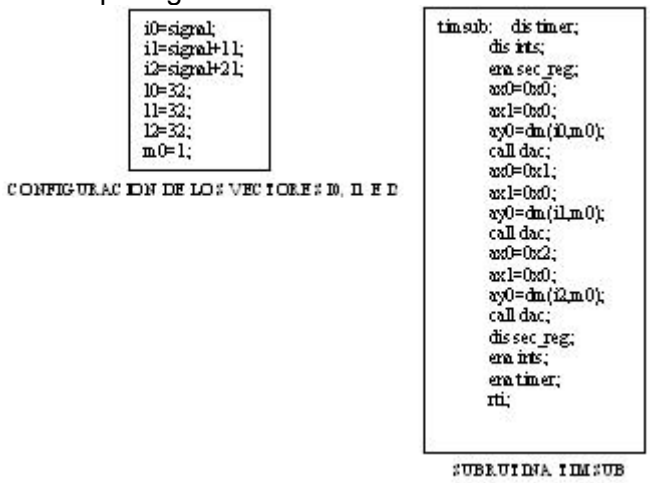
FIGURA 107. Solución para el aumento de muestras de 16 a 32 conservando la frecuencia de 60 Hz.



Fuente: El autor.

- Diseñar un programa que genere un sistema trifásico de 60 Hz utilizando tres canales de salidas analógicas de la tarjeta adaptable. (Ver figura 108)

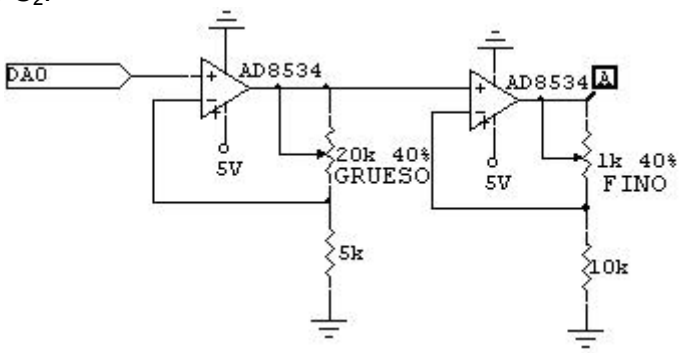
FIGURA 108. Solución para generar un sistema trifásico de 60 Hz.



Fuente: El autor.

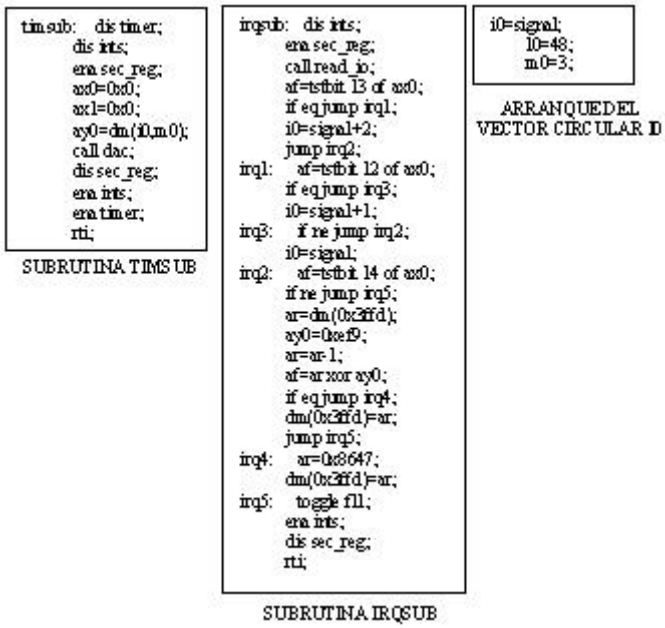
- Diseñar un generador de ondas seno, cuadrada y diente de sierra. La salida se debe realizar a través del pin DA0 del conector DA-CON. La frecuencia y la forma de onda se controla con el programa mientras que la magnitud de la señal se regula mediante la construcción de un circuito externo. La amplitud de la señal seleccionada deberá tener dos ajustes uno grueso y otro fino (ver figura 109). La subrutina con el programa del generador de señales se muestra en la figura 110.

FIGURA 109. Circuito para cambiar la amplitud de la señal escogida con los interruptores S_1 y S_2.



Fuente: El autor. Elaborado con el *software* CIRCUITMAKER.

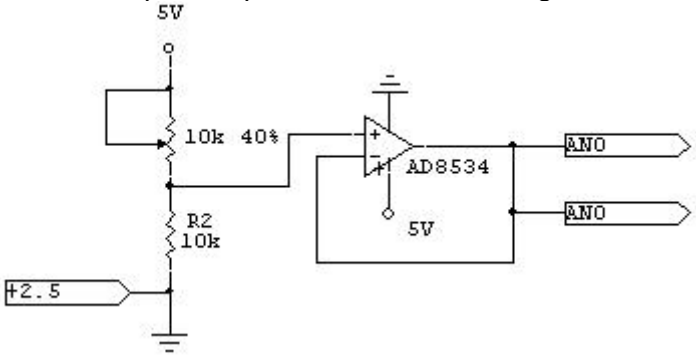
FIGURA 110. Solución de *software* para el problema propuesto.



Fuente: El autor.

- Construya el circuito de la figura 111.

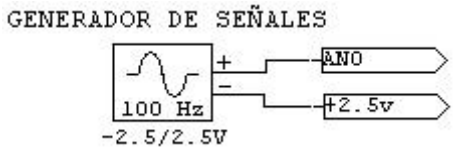
FIGURA 111. Circuito de prueba para los canales analógicos de entrada.



Fuente: El autor. Elaborado con el *software* CIRCUITMAKER.

- Abrir el programa VISUALDSP++ y en el menú PROJECT seleccione la opción OPEN PROJECT y la carpeta con el nombre lab_ad1 que se encuentra en la dirección **c:\archivos de programa\analog devices\visualdsp\218x\examples\lab_io y seleccione lab_ad1**.
- En el menú SESSION seleccione la opción NEW SESSION y escoja la opción EZ-KIT 218x en la lista DEBUG TARGET y en PROCCESOR seleccione ADSP-2181.
- Si todo se ha realizado bien deberá aparecer una ventana donde se le pide al usuario oprimir el botón RESET de la tarjeta EZKIT2181. Después de presionar el pulsador de RESET el diodo LED rojo dejará de parpadear quedando encendido indicando que la tarjeta se ha comunicado adecuadamente con el computador.
- Conectar el cable de comunicación serial que interconecta la tarjeta EZKIT2181 con la tarjeta adaptable a través del conector nombrado como DSP-CON.
- Comunicar la tarjeta adaptable con el circuito de la figura 105 a través de los conectores nombrados como DA-CON y AD-CON de la tarjeta adaptable.
- En el menú DEBUG seleccione la opción RUN. Presione el pulsador INTERRUPT de la tarjeta EZKIT2181 y mida con un voltímetro la señal en los terminales AN0 del circuito de la figura 111. Realice la misma medición en los canales DA0 y DA1 del conector DA-CON. ¿Los errores entre las dos mediciones, qué origen tienen? **Rta: la resolución de los canales de entrada y de salida es diferente. La salida analógica de tensión tiene 12 bits mientras que la entrada analógica tiene 16 bits de resolución.**
- Cambie la tensión de entrada moviendo el tornillo del potenciometro y repita el numeral anterior.
- Cambie la entrada analógica de los terminales AN0 a los terminales AN1 y repita los dos numerales anteriores. Haga lo anterior hasta barrer todo el conector AD-CON.
- Construya el circuito de la figura 112.

FIGURA 112. Circuito para analizar los filtros antisolapamiento de la tarjeta adaptable.



Fuente: El autor. Elaborado con el *software* CIRCUITMAKER.

- Comunicar la tarjeta adaptable con el circuito de la figura 105 a través del conector nombrado como AD-CON de la tarjeta adaptable.
- Presionar una vez el pulsador INTERRUPT de la tarjeta EZKIT2181 para fijar la frecuencia de corte de los filtros antisolapamiento de los canales analógicos de entrada de la tarjeta adaptable.
- En el generador de ondas seleccione la onda seno con una frecuencia de 100 Hz y una amplitud de ±2,5 V. Coloque la punta del osciloscopio en una de las salidas de los amplificadores operacionales de los canales de entrada analógica. Aumente la frecuencia hasta que el valor pico de la tensión sea de 1,7776 V. Esta frecuencia corresponde a la frecuencia de corte del filtro cuando la resistencia del potenciometro se encuentra en el valor máximo.
- Regrese el generador de ondas a sus valores iniciales y presione el pulsador INTERRUPT de la tarjeta EZKIT2181. Repita el procedimiento anterior y analice lo sucedido. ¿Aumenta o disminuye el valor de la frecuencia de corte del filtro antisolapamiento? ¿Porque? **Rta: la frecuencia de corte aumenta debido a la disminución de la resistencia del potenciometro variable.**

$$\omega_c = \frac{1}{R_{pot} \times C} \rightarrow F_c = \frac{1}{2 \times \pi \times R_{pot} \times C}$$

- Trace una curva que relacione la frecuencia de corte contra la resistencia del potenciometro variable.

5.2. FILTROS DIGITALES FIR E IIR.

5.2.1. Objetivos.

- Comprobar de manera práctica el funcionamiento de los filtros digitales de respuesta impulsional infinita (IIR) y finita (FIR), utilizando el DSP y los canales de entrada y salida analógica de la tarjeta adaptable.
- Profundizar en el funcionamiento de los vectores circulares mediante el empleo de los mismos en el vector de datos y coeficientes de un filtro FIR.

5.2.2. Lecturas recomendadas.

- Leer las funciones básicas disponibles de la tarjeta adaptable, tales como: adc, dac, poten.
- Funcionamiento de un vector circular en los filtros FIR e IIR. Ver el documento Mixedsignal_sect6.pdf Anexo de este documento.

- Revisar el diseño de filtros FIR e IIR en el libro Tratamiento digital de señales, Principios, algoritmos y aplicaciones de Proakis y manolakis. Editorial Prentice-hall, tercera edición. Capítulo 8.
- Estudiar el funcionamiento de las subrutinas del programa MATLAB que realizan el diseño de filtros tales como: CREMEZ, REMEZ, FIRLS, FIR1, FIR2, BUTTER, CHEBY1, CHEBY2, ELLIP. Documento anexo: signal_tb.pdf.

5.2.3. Elementos.

- Osciloscopio.
- Generador de ondas.

5.2.4. Procedimiento.

- Abrir el programa VISUALDSP++ y en el menú PROJECT seleccione la opción OPEN PROJECT y la carpeta con el nombre lab_fir que se encuentra en la dirección **c:\archivos de programa\analog devices\visualdsp\218x\examples\lab_fir y seleccione lab_fir**.
- En el menú SESSION seleccione la opción NEW SESSION y la opción EZ-KIT 218x en la lista DEBUG TARGET y en PROCCESOR seleccione ADSP-2181.
- Si todo se ha realizado bien deberá aparecer una ventana donde se le pide al usuario oprimir el botón RESET de la tarjeta EZKIT2181. Después de presionar el pulsador de RESET el diodo LED rojo dejará de parpadear quedando encendido indicando que la tarjeta se ha comunicado adecuadamente con el computador.
- Conectar el cable de comunicación serial que interconecta la tarjeta EZKIT2181 con la tarjeta adaptable a través del conector nombrado como DSP-CON.
- Comunicar la tarjeta adaptable con el circuito de la figura 109 a través del conector nombrado como AD-CON de la tarjeta adaptable. La salida del filtro FIR se puede observar en el pin DA0 del conector DA-CON.
- Diseñar un filtro FIR de fase lineal con 15 coeficientes con respuesta impulsional simétrica y respuesta en frecuencia que satisfaga las siguientes condiciones:

$$H_r(\frac{2\pi k}{15}) = [1,\ 1,\ 1,\ 1,\ 0.4,\ 0,\ 0,\ 0]\ \ k = 0,1,2,.....,7$$

- calcular el vector $G[k] = (-1)^k Hr[k]$

$$G[k] = [1,\ -1,\ 1,\ 1,\ 0.4,\ 0,\ 0,\ 0]\ \ k = 0,1,....,7$$

- Si $\alpha = 0$ y M impar (M: coeficientes del filtro) se puede calcular la respuesta impulsional del filtro así:

n	h[n]
0,14	-14,11e-3
1,13	-1,945e-3
2,12	40e-3
3,11	12,23e-3
4,10	-91,38e-3
5,9	-18,08e-3
6,8	313,3e-3
7	520,0e-3

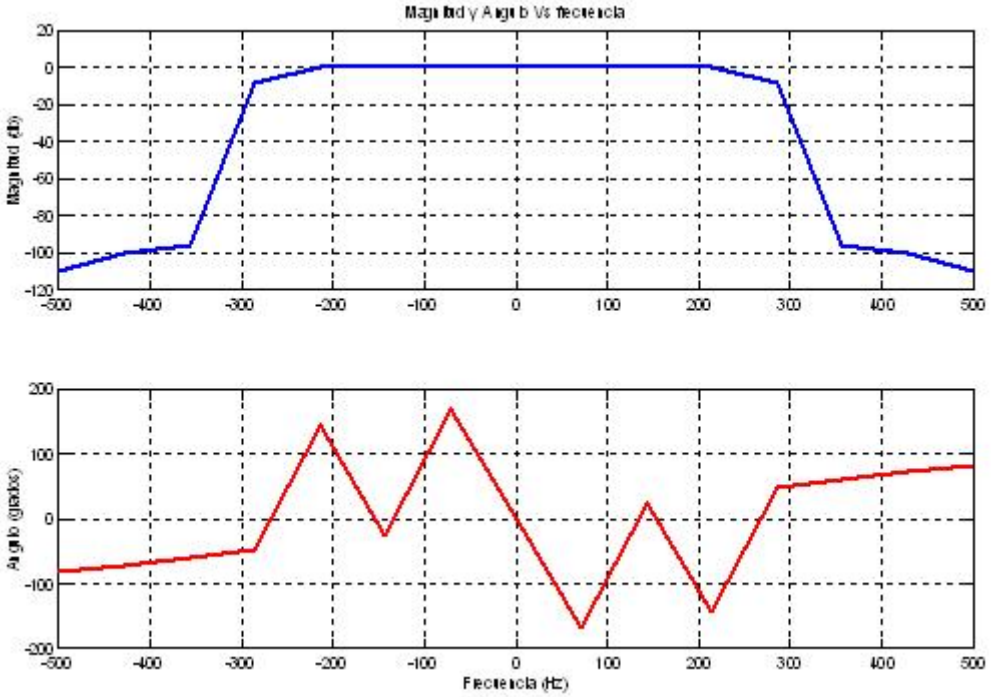
$$h[n] = \frac{1}{15}\left\{G[0] + 2\sum_{k=1}^{7} G[k]\cos\left(\frac{2\pi k(n+0.5)}{15}\right)\right\}$$

- Esta respuesta impulsional se encuentra en un formato diferente al formato fraccional 1.15 (1 bit de signo y 15 bits de fracción) que acepta el DSP, por lo tanto se debe realizar una transformación de la siguiente forma:

$$h[n] = \text{int}(32767 \times h[n])$$ donde int: parte entera.

- La característica de magnitud y ángulo de fase para una frecuencia de muestreo de 1000 Hz se puede observar en la figura 113.

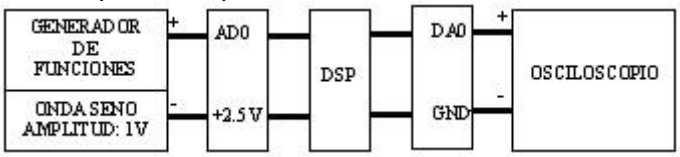
FIGURA 113. Magnitud y ángulo contra frecuencia para el filtro diseñado.



Fuente: El autor. Elaborado con el *software* MATLAB.

- Realizar el montaje de la figura 114.

FIGURA 114. Circuito para comprobar el filtro FIR diseñado en el DSP.



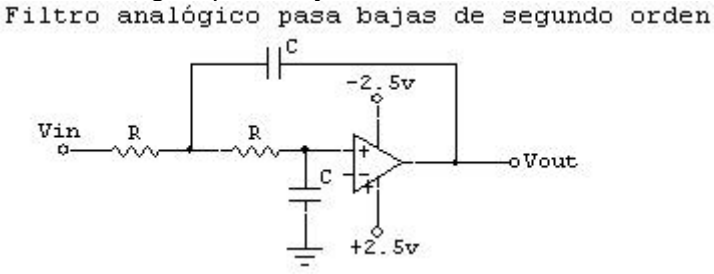
Fuente: El autor.

- Abrir el menú DEBUG y seleccionar la opción RUN del programa lab_fir.
- Presionar una vez el botón INTERRUPT de la tarjeta EZKIT2181 para fijar el valor de la frecuencia de corte de los filtros antisolapamiento de los canales analógicos de entrada de la tarjeta adaptable.
- Obtenga la característica real de magnitud contra frecuencia realizando un barrido de frecuencia desde 10 hasta 600 Hz con incrementos de 10 Hz.
- Diseñar y comprobar en el canal de procesamiento de señal de la figura 111 un filtro FIR de respuesta impulsional simétrica con la respuesta frecuencial que se pide a continuación:

$$H_r\left(\frac{2\pi k}{32}\right) = [1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 0{,}3789795,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0]$$

- Ejecutar el programa VISUALDSP++ y en el menú PROJECT seleccione la opción OPEN PROJECT y la carpeta con el nombre lab_iir que se encuentra en la dirección **c:\archivos de programa\analog devices\visualdsp\218x\examples\lab_iir y seleccione lab_iir**.
- En el menú SESSION seleccione la opción NEW SESSION y la opción EZ-KIT 218x en la lista DEBUG TARGET y en PROCCESOR seleccione ADSP-2181.
- Si todo se ha realizado bien deberá aparecer una ventana donde se le pide al usuario oprimir el botón RESET de la tarjeta EZKIT2181. Después de presionar el pulsador de RESET el diodo LED rojo dejará de parpadear quedando encendido indicando que la tarjeta se ha comunicado adecuadamente con el computador.
- Conectar el cable de comunicación serial que interconecta la tarjeta EZKIT2181 con la tarjeta adaptable a través del conector nombrado como DSP-CON.
- Comunicar la tarjeta adaptable con el circuito de la figura 109 a través del conector nombrado como AD-CON de la tarjeta adaptable. La salida del filtro FIR se puede observar en el pin DA0 del conector DA-CON.
- Diseñar un filtro paso bajos digital IIR a partir del filtro analógico que se muestra en la figura 115. El filtro debe tener una atenuación de 3 dB en la frecuencia Ω_c igual a 10 Hz y una frecuencia de muestreo de 1000 Hz utilizando la transformación bilineal.

FIGURA 115. Filtro analógico pasa bajas.



Fuente: El autor. Elaborado con el *software* CIRCUITMAKER.

La función de transferencia del filtro analógico de la figura 115 es igual a:

$$H(S) = \frac{\frac{1}{(RC)^2}}{S^2 + \left(\frac{2}{RC}\right)S + \frac{1}{(RC)^2}}$$

$$\Omega_c = \frac{1}{RC}$$

$$H(S) = \frac{\Omega_c^2}{S^2 + \Omega_c S + \Omega_c^2}$$

$$\Omega_c = \frac{2}{T}\tan\left(\frac{10\pi}{F_s/2}\right) \quad F_s = 1000Hz$$

$$H(S) = \frac{\frac{0.1258^2}{T^2}}{S^2 + 2\left(\frac{0.1258}{T}\right)S + \left(\frac{0.1258}{T}\right)^2}$$

Teniendo en cuenta la transformación bilineal se obtiene la transformada Z como:

$$S = \frac{2}{T}\left(\frac{Z-1}{Z+1}\right) \Rightarrow H(z) = \frac{3.5\times10^{-3}(Z^2+2Z+1)}{Z^2 - 1.76329Z + 0.777729}$$

$$H[Z] = \frac{3.5\times10^{-3}\times Z^2\times\left(1+2Z^{-1}+Z^{-2}\right)}{Z^2\times\left(1-1.76329Z^{-1}+0.777729Z^{-2}\right)} = \frac{3.5\times10^{-3}\times(1+2Z^{-1}+Z^{-2})}{1-1.76329Z^{-1}+0.441066982Z^{-2}}$$

La ecuación en diferencias para implementar el filtro es:

$$y[n]\times(1-1.76329Z^{-1}+0.777729Z^{-2}) = x[n]\times(3.5\times10^{-3}\times(1+2Z^{-1}+Z^{-2}))$$

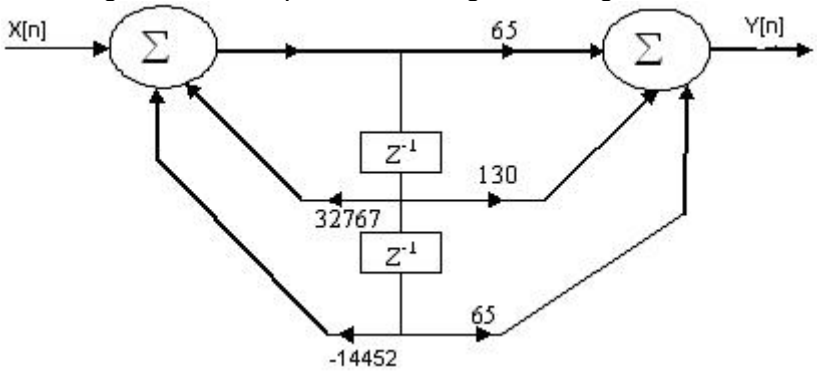
$$y[n] = 1.76329\,y[n-1] - 0.777729\,y[n-2] + 3.5\times10^{-3}\,x[n] + 7\times10^{-3}\,x[n-1] + 3.5\times10^{-3}\,x[n-2]$$

$$y_{ENTERA}[n] = \left(\frac{32767}{1.76329}\right)\times y[n]$$

$$y[n] = 32767\,y[n-1] - 14452\,y[n-2] + 65x[n] + 130x[n-1] + 65x[n-2]$$

El diagrama de bloques correspondiente al filtro digital se observa en la figura 116.

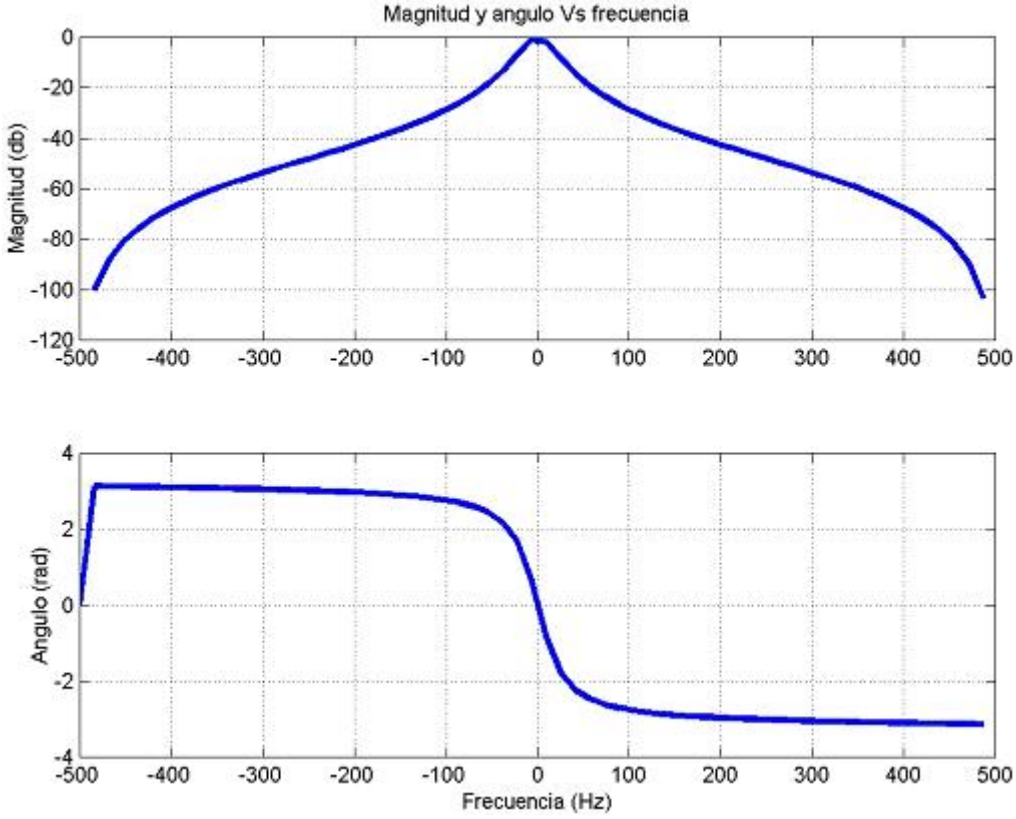
FIGURA 116. Diagrama de bloque del filtro digital de segundo orden.



Fuente: El autor.

En la figura 117 se muestra la respuesta en frecuencia del filtro diseñado.

FIGURA 117. Características de magnitud y ángulo contra frecuencia del filtro IIR.



Fuente: El autor. Elaborado con el *software* MATLAB.

- Utilizando el circuito de la figura 111 compruebe la curva de magnitud contra frecuencia de la figura 113. Utilice un barrido desde 3 Hz hasta 200 Hz. Verifique que en la frecuencia de 10 Hz el filtro tenga la atenuación deseada. ¿Porqué causas el filtro no alcanzaría esta especificación?

- ¿Qué diferencias hay en la fase del filtro IIR con respecto a la fase del filtro FIR?
- ¿Desde el punto de vista computacional que diferencias hay en el diseño de un filtro IIR y un filtro FIR?
- ¿El filtro IIR diseñado, podría tornarse inestable? ¿Porqué?
- Diseñar y comprobar en el DSP un filtro IIR digital a partir de la siguiente función de transferencia:

$$H(S) = \frac{A_0\left(\frac{\Omega_0}{Q}\right)S}{S^2 + \left(\frac{\Omega_0}{Q}\right)S + \Omega_0^2}$$

Ω_0: frecuencia central. A_0: ganancia en la frecuencia central. Q: factor de calidad.

5.3. EVALUACIÓN DE RENDIMIENTO DE LOS ALGORITMOS PARA EL CÁLCULO DE LA TRANSFORMADA DISCRETA DE FOURIER.

5.3.1. Objetivos.

- Comprobar mediante la utilización de un canal analógico de salida, el funcionamiento de los algoritmos de FFT, a través de señales discretas de espectros conocidos.
- Comparar la velocidad de ejecución del algoritmo FFT escrito en lenguaje ensamblador contra el mismo escrito en lenguaje C.
- Comparar los algoritmos de FFT en base 2, base 4 y base múltiple desde el punto de vista del tiempo de ejecución de cada uno.
- Verificar el funcionamiento de los algoritmos de FFT para señales reales; aquellos que aprovechan la parte imaginaria ya sea para procesar la mitad de la señal u otra señal diferente.
- Comprobar el funcionamiento del algoritmo de GOERTZEL, identificando tonos senoidales.
- Comprobar el algoritmo CHIRP para aumentar la resolución en frecuencia del espectro de una señal conocida.

5.3.2. Lecturas recomendadas.

- Capítulo VI del libro “Tratamiento digital de señales”, Proakis y Manolakis, editorial Prentice-Hall, tercera edición.

- Capítulo IX del libro *Discrete-Time signal processing*, Oppenheim y Schafer, editorial Prentice-Hall.
- Capítulo VIII del libro *Fast Fourier Transform and its applications*, E. Oran Brigham, editorial Prentice-Hall.

5.3.3. Elementos.

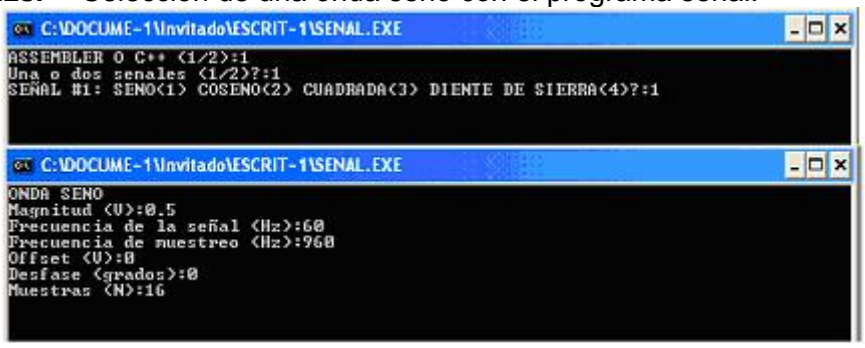
- Generador de señales.
- Osciloscopio.

5.3.4. Procedimiento.

- Abrir el programa VISUALDSP++ y en el menú PROJECT seleccione la opción OPEN PROJECT y la carpeta con el nombre lab_io1 que se encuentra en la dirección **c:\archivos de programa\analog devices\visualdsp\218x\examples\lab_fft_base_2_en_assembler y seleccione lab_fft2.**
- En el menú SESSION seleccione la opción NEW SESSION y la opción EZ-KIT 218x en la lista DEBUG TARGET y en PROCCESOR seleccione ADSP-2181.
- Si todo se ha realizado bien deberá aparecer una ventana donde se le pide al usuario oprimir el botón RESET de la tarjeta EZKIT2181. Despues de presionar el pulsador de RESET el diodo LED rojo dejará de parpadear quedando encendido indicando que la tarjeta se ha comunicado adecuadamente con el computador.
- Conectar el cable de comunicación serial que interconecta la tarjeta EZKIT2181 con la tarjeta adaptable a través del conector nombrado como DSP-CON.
- En la misma carpeta seleccionada previamente, abra el programa **señales.exe** y genere una onda seno de amplitud 0,5 V, 60 Hz, frecuencia de muestreo 960 Hz y 16 muestras**.** El primer mensaje que aparece en dicho programa le pregunta al usuario si la señal va a ser procesada en lenguaje Ensamblador o en lenguaje C; si se va a trabajar en Ensamblador seleccione 1, si va a trabajar en lenguaje C seleccione la opción 2. A continuacón aparecen 4 formas de onda diferentes seno (1), coseno (2), cuadrada (3) y rampa (4). Se debe colocar uno cualquiera de los cuatro números que identifican a cada señal. Despues de seleccionar la señal deseada el programa pregunta por las características fundamentales de dicha señal como: amplitud, frecuencia, frecuencia de muestreo, desfase, nivel de *offset* y número de muestras deseado. Cuando el programa concluye se obtienen dos archivos, uno llamado “real.dat” en el cual se almacena la parte real de la señal de entrada y otro

llamado “imag.dat” en el que se almacena la parte imaginaria de la misma. En la figura 118 se muestra el procedimiento paso a paso para seleccionar una onda seno con el programa señal.

FIGURA 118. Selección de una onda seno con el programa señal.



Fuente: El autor.

- En la carpeta seleccionada abra el programa “**coeffs.exe”**. Este programa permite almacenar en los archivos “coeffr.dat” y “coeffi.dat” los coeficientes reales e imaginarios necesarios para correr el algoritmo FFT en base 2. En la figura 119 se observa el procedimiento para generar los coeficientes para un algoritmo FFT en base 2 de 16 muestras.

FIGURA 119. Selección de los coeficientes para un algoritmo FFT en base 2 de 16 muestras.

Fuente: El autor.

- En el programa VISUALDSP++ seleccione en el menú PROJECT la opción BUILD ALL. A continuación si no hay errores en la compilación del proyecto abra el menú DEBUG y seleccione la opción RUN. En los canales analógicos 1 y 2 se obtiene el tiempo que le tomo al DSP ejecutar la subrutina FFT_B2 y en los canales 2 y 3 se puede obtener la medida del tiempo para la subrutina encargada de invertir el orden de los bits de la dirección del vector de salida. En los canales analógicos de salida 4 y 5 se pueden observar los espectros de magnitud y ángulo en forma periódica. Para calcular el tiempo de ejecución de las subrutinas en función de la tensión leida en los canales análogicos utilice la siguiente ecuación:

$$ciclos_{FFT\_B2} = (10 \times 32767 \times da_0) + \left(\frac{32767 \times da_1}{5} \right)$$

$$ciclos_{BITREVERSE} = (10 \times 32767 \times da_2) + \left(\frac{32767 \times da_3}{5} \right)$$

- Observando el programa principal, explique como se obtuvo la ecuación para relacionar el nivel de tensión en los canales analógicos de salida con el número de ciclos empleado por las subrutinas.
- Repetir el procedimiento anterior con una señal cuadrada de 32 muestras. Consignar en la hoja de resultados el tiempo empleado por las subrutinas y la forma de onda obtenida en los canales 4 y 5.
- Llene la tabla que sigue a continuación y discuta los resultados de la misma. Utilice las subrutinas en lenguaje Ensamblador. Las características de la señal cuadrada de entrada son: frecuencia 60 Hz, amplitud 0,5 voltios y 1 ciclo **Nota: la FFT_B2 en lenguaje C se encuentra en la carpeta: archivos de programa/analog devices/adsp2181/examples/lab_fft_2_C y la FFT_B4 se encuentra en la carpeta: archivos de programa/analog devices/ adsp2181/examples/lab_fft_4_C**.

N (muestras)	Ciclos FFT_B2	Ciclos BITREVERSE2	Ciclos FFT_B4	Ciclos BITEREVERSE4
16				
64				
256				

- Llene la tabla que sigue a continuación y discuta los resultados de la misma. Utilice las subrutinas en lenguaje C . Las características de la señal cuadrada de entrada son: frecuencia 60 Hz, amplitud 0,5 voltios y 1 ciclo. **Nota: la FFT_B2 en lenguaje C se encuentra en la carpeta: archivos de programa/analog devices/adsp2181/examples/lab_fft_2_C y la FFT_B4 se encuentra en la carpeta: archivos de programa/analog devices/ adsp2181/examples/lab_fft_4_C**.

N (muestras)	Ciclos FFT_B2	Ciclos BITREVERSE2	Ciclos FFT_B4	Ciclos BITEREVERSE4
16				
64				
256				

- Que diferencias observa en el número de ciclos empleado por las subrutinas en lenguaje C y en lenguaje Ensamblador? **Nota: Utilice la opción DISASSEMBLY del programa VISUALDSP++ en lenguaje C para analizar el proceso realizado por este programa para convertir el lenguaje C en lenguaje Ensamblador**.
- Utilice la subrutina FFT_B4 para obtener en los canales analógicos de salida 0 y 1 la magnitud y el ángulo contra la frecuencia para la siguiente señal:

$$x[n] = \frac{1}{4}\cos\left(\frac{32\pi n}{f_s}\right) + \frac{1}{4}\cos\left(\frac{64\pi n}{f_s}\right) + \frac{1}{4}\cos\left(\frac{128\pi n}{f_s}\right) + \frac{1}{4}\cos\left(\frac{256\pi n}{f_s}\right)$$

Nota: se debe seleccionar f_s de tal manera que durante 4 periodos de la señal se obtengan 256 muestras de la misma.

- Utilizando el procedimiento de diezmado en frecuencia genere un algoritmo FFT que pueda procesar señales de entrada con N igual a 3, 9, 27, 81,…Para este algoritmo, ¿cuál sería la base? ¿Cuántas etapas, grupos por etapa y mariposas por grupo se requieren para calcular la FFT de una señal cuadrada de 27 muestras? Genere el código y obtenga en los canales analógicos de salida 0 y 1 la magnitud y el ángulo contra la frecuencia para esta señal.
- Utilizando el algoritmo de fft en base mixta calcule el espectro de una señal cuadrada de 0,5 voltios de amplitud, frecuencia de 60 Hz, frecuencia de muestreo igual a 2880 Hz y N igual a 48 muestras. La carpeta que contiene esta subrutina es: archivos de programa/analog devices/adsp2181/examples/lab_fft_mixta. En este programa usted debe almacenar las constantes equivalentes a los múltiplos en el vector de entrada denominado factores y debe cambiar el número de etapas de acuerdo al número de coeficientes. Observe el siguiente procedimiento a manera de ejemplo:

$$N = 48 \rightarrow N = 2 \times 3 \times 8 \rightarrow coeficientes[3] = \{2,3,8\} \rightarrow etapas = 3$$

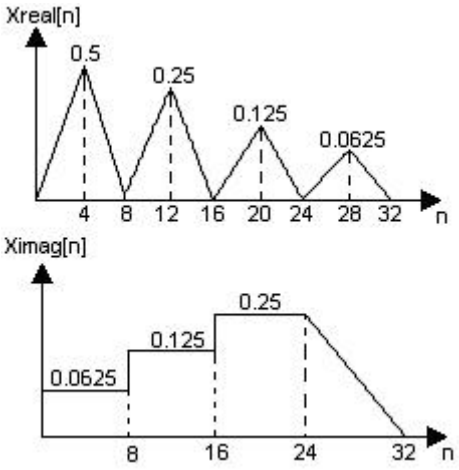
Llene la siguiente tabla utilizando la subrutina FFT en base mixta:

r_1*r_2*r_3*…….	Tensión canal da0	Tensión canal da1	Ciclos
2*3*8			
6*8			
2*2*3*4			

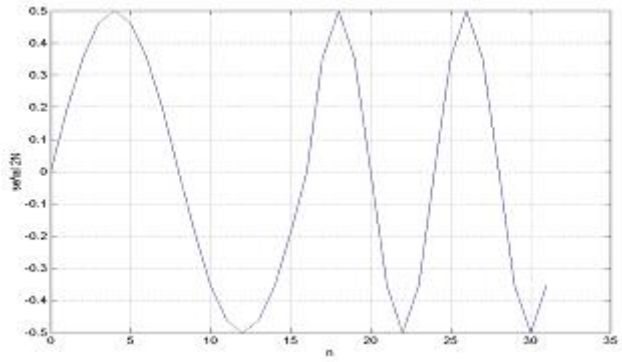
- Observe que las tres secuencias representan el mismo número; ¿Cuál tiene mejor rendimiento?, ¿Porque?
- A continuación se contrastará el algoritmo FFT en base mixta con el algoritmo FFT en base 4 para valores de N diferentes mediante la siguiente tabla:

Base 4	**Base mixta**	**Ciclos FFT_4**	**Ciclos FFT_mixta**
	2*4		
	3*3		
	4*3		
	2*5		
	2*6		
4*4	3*5		
	3*6		
	4*5		
	6*4		
	6*5		
	6*6		
	8*5		
	3*3*5		
	6*8		
	2*5*5		
4*4*4	3*4*5		

- Cual es la principal conclusión con respecto a la evaluación del algoritmo FFT en base mixta?
- Utilice el algoritmo que emplea el vector Imaginario de la señal de entrada para calcular el espectro, magnitud y ángulo, de las dos señales que se muestran a continuación:



- Multiplique la señal Xreal por 2 y repita el cálculo del espectro. ¿Se obtiene la misma forma de onda del espectro en magnitud y ángulo? Si la respuesta es negativa ¿Cuál es la causa de este efecto? **Ayuda: escalamiento de las señales de entrada.**
- Utilizando el algoritmo que emplea el vector Imaginario de la señal de entrada para almacenar los elementos impares de una señal con longitud 2N, calcule el espectro, magnitud y ángulo, de la siguiente señal:

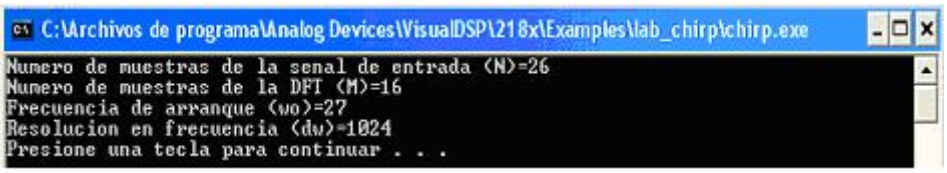


- La FFT no brinda información acerca del instante de tiempo donde ocurre cambios en la frecuencia de la señal. El hecho anterior se observa claramente en el ejemplo anterior en el cual la señal posee dos frecuencias para dos instantes de tiempo diferentes. Si se requiriera analizar efectos como cambios bruscos de frecuencia en el sistema de potencia de la red nacional la FFT no sería un buen candidato para este trabajo, ¿Qué tipo de algoritmo(s) se pueden implementar para llevar a cabo esta labor? Diséñelo y prográmelo en el DSP. **Pista: Transformada de Fourier de tiempo corto.**
- Abrir el programa lab_Goertzel en lenguaje Ensamblador que se encuentra en la carpeta: archivos de programa/analog devices/visual dsp/ads218x/examples/lab_Goertzel. Utilizando el programa **señales.exe** cree una señal senoidal de 0,5 V, 60 Hz, 16 muestras y frecuencia de muestreo de

960 Hz y llene la tabla que aparece a continuación. **Nota M4 es la variable que selecciona el coeficiente que se quiere evaluar con el algoritmo de Goertzel. Esta variable se encuentra antes de la instrucción call Goertzel.**

M4	magnitud (canal de salida analógica 2)	ángulo (canal de salida analógica 3)
1		
2		
3		
4		
5		
6		
7		
8		

- Repita el ejemplo anterior pero con una onda cuadrada con características de amplitud, frecuencia, número de muestras y frecuencia de muestreo iguales a la onda senoidal.
- Utilice el algoritmo de Goertzel para crear un programa que decodifique los tonos de un teclado telefónico de 12 teclas. Nota: investigar acerca de la decodificación de tonos de multi-frecuencia (DTMF).
- Abrir el programa lab_chirp en lenguaje Ensamblador que se encuentra en la carpeta: archivos de programa/analog devices/visual dsp/ads218x/examples/lab_Chirp. Utilizando el programa **señales.exe** cree una señal cuadrada de 0,5 V, 60 Hz, 64 muestras y frecuencia de muestreo de 3840 Hz. Abrir el programa chip.exe y siga la secuencia de instrucciones que se observan en la figura:



- Coloque la punta del osciloscopio en los canales de salida analógica 2 y 3, consigne en una hoja las formas de onda observadas.
- Utilice el programa **chirp.exe** y escoja una resolución de 512 y frecuencia de arranque de 100 y repita el numeral anterior.

6. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se recopilan los principales resultados obtenidos con la realización del trabajo de investigación, se establecen algunas conclusiones generales y se proponen algunos desarrollos para continuar con este trabajo.

6.1. CONCLUSIONES

Dentro de los objetivos propuestos en el plan de trabajo de la investigación se encontraban la construcción de la tarjeta adaptable, la programación de la DFT utilizando algoritmos de FFT conocidos como la base 2 y 4 en el procesador de señales digitales, DSP, y la elaboración de prácticas de laboratorio con la tarjeta adaptable y los programas previamente mencionados. Estos objetivos se cumplieron totalmente y se superaron ya que se utilizó el trabajo de investigación para llevar a cabo el siguiente trabajo extra: la comparación de rendimiento computacional entre el lenguaje Ensamblador y el lenguaje C utilizado por el compilador del programa VISUALDSP++; la obtención de algoritmos para procesar eficientemente las mariposas de tamaño 3, 5, 6 y 8 muy utilizadas en el procedimiento computacional de la FFT en base mixta; la programación de los algoritmos de FFT en el DSP que permiten reducir el número de ciclos empleando la parte imaginaria de la señal de entrada; la programación de algoritmos como Goertzel y CHIRP que permiten obtener de manera parcial y no total el espectro de una señal discreta; la programación del algoritmo CORDIC que permite obtener la magnitud y el ángulo de un número complejo mediante rotación vectorial.

El DSP mediante el uso de los punteros, DAGs, implementados en *hardware* ha permitido realizar labores que no eran fáciles de implementar en otro tipo de procesadores de propósito general como los microcontroladores. Particularmente útil resulta esta herramienta porque libera recursos de *software* que con anterioridad se utilizaban para el post-incremento, post-decremento y almacenamiento de la dirección resultante. El hecho anterior permite realizar programas compactos, algunas veces con poca claridad para el personal con experiencia en microcontroladores y no en los procesadores de señales digitales. Otro aspecto que cabe resaltar es la implementación de ciclos repetitivos en *hardware*; este hecho es un avance significativo que favorece el uso de los DSPs en aplicaciones de procesamiento de señales, ya que, libera recursos de *software* que antes se utilizaban en la variable que almacenaba el ciclo actual y en el control de la condición que teminaba el mismo, además permite con el mismo dispositivo en conjunto con una pila de almacenamiento

de direcciones crear ciclos anidados, hasta cuatro, los cuales son muy útiles cuando se programan algoritmos como la FFT de cualquier base.

Los incovenientes o problemas mas relevantes de los DSPs son: la memoria EEPROM interna es inexistente, el conjunto de instrucciones es muy diferente a los encontrados en microcontroladores y los periféricos son escasos. Estos hechos han logrado que algunos programadores de microcontroladores se muestren reacios al cambio, ya que, cada vez que se energice el DSP se debe cargar el programa al menos que se instale externamente el espacio para la memoria de programa; además no se realiza con facilidad una conexión entre un DSP y dispositivos externos. Las últimas familias de DSPs que han salido al mercado son dispositivos híbridos; cuentan con un núcleo de procesamiento digital de señales, memoria EEPROM de programa, una buena cantidad de periféricos y un conjunto de instrucciones amigable o compatible con los de un microcontrolador.

La cantidad de ciclos de máquina consumidos por los programas elaborados en lenguaje Ensamblador es mucho menor que los empleados por los programas elaborados en lenguaje C. El lenguaje C utilizado no permite explotar de manera directa algunas de las ventajas del procesador DSP como los vectores circulares y los apuntadores automáticos para el post-incremento y post-decremento de un vector de datos. Lo anterior incide notoriamente en el desempeño, porque el programa requiere de la utilización de variables auxiliares de realizar esta labor que el procesador DSP puede realizar en forma automática.

A medida que el tamaño de la señal de entrada (número de muestras) aumenta, la diferencia computacional es mayor debido a la cantidad de operaciones adicionales que se deben implementar para emular en lenguaje C los elementos que el DSP brinda de manera automática. Una gran cantidad de operaciones que puede realizar el DSP directamente con los registros internos de las unidades aritméticas que procesan los datos en lenguaje C, se implementan a través de la memoria RAM de datos; este factor influye de manera importante en la cantidad de ciclos empleados por el código escrito en lenguaje C; para realizar esta operación en lenguaje Ensamblador, se carga la variable directamente en el registro interno que se necesita.

El espacio en memoria ocupado por los programas en lenguaje C es mucho mayor que el espacio necesario para realizar la misma labor en lenguaje Ensamblador. El hecho anterior tiene como fundamento lo siguiente: el programa en lenguaje C no solo necesita para ejecutarse en el procesador el código de usuario sino que también requiere código adicional para que la traducción de lenguaje C a lenguaje Ensamblador sea lo mejor posible. Este código adicional se encuentra en las bibliotecas cabeceras con extensión .H provistas por el lenguaje C para realizar muchas de las labores que permite esta herramienta computacional.

El algoritmo de la FFT en base 2 es más eficiente cuando el número de muestras es igual a 2^N donde N es el tamaño del vector de entrada. El algoritmo de la FFT en base 4 es más eficiente cuando el número de muestras es igual a 4^N. Se debe utilizar la FFT en base mixta para valores de N que no se pueden obtener con 2^N o con 4^N. A pesar de esto existen valores de N que no se deben procesar con la FFT de base mixta debido a que el número de ciclos computacionales supera con creces los valores obtenidos con los algoritmos de FFT en base 2 o en base 4 más cercanos al valor de N. En conclusión solo se debe utilizar la FFT en base mixta cuando N no sea un múltiplo de 2 o de 4 y el número de ciclos de la misma sea menor que el número de ciclos empleados por las FFTs en base 2 y en base 4; de lo contrario se deben utilizar los algoritmos FFT en base 2 y en base 4 rellenando con ceros la señal de entrada hasta cumplir con el factor de 2 o de 4 mas cercano al valor de N.

El lenguaje C puede emplearse para solucionar problemas computacionales. Si la subrutina o programa que se desea hacer no tiene gran complejidad computacional no vale la pena realizarlo en lenguaje C. Lo anterior se fundamenta en el hecho de la gran cantidad de espacio que ocupa un programa en lenguaje C contra la escasa cantidad de memoria que se tiene disponible en la memoria interna de un DSP. Si el *hardware* que contiene el DSP es reducido en tamaño no sería valida la posibilidad de colocar una memoria adicional solamente para almacenar bibliotecas de un lenguaje que tal vez no se utilicen todas dentro de la aplicación.

En aplicaciones donde el espacio de memoria disponible sea una restricción importante y la velocidad del procesador no lo sea, se deben utilizar subrutinas que permitan calcular los coeficientes de manera progresiva. En este campo los algoritmos de rotación digital de coordenadas, CORDIC por su sigla en inglés, son de gran ayuda ya que permiten ahorrar el espacio requerido para implementar en una tabla los coeficientes que requieren los algoritmos de la FFT para ser ejecutados. Estos

algoritmos representan una gran ventaja debido a que no calculan de manera independiente la componente real e imaginaria sino que compactan este cálculo en una sola subrutina.

El algoritmo de Goertzel se debe emplear solamente cuando se requiera obtener una cantidad limitada del espectro en magnitud y ángulo de una señal de entrada. A medida que la cantidad de muestras del espectro aumenta, se torna altamente ineficiente haciendo preferible el cálculo de las muestras del espectro utilizando un algoritmo FFT.

La complejidad computacional del algoritmo CHIRP esta muy ligada a la cantidad de ciclos que emplea el algoritmo FFT para obtener los espectros de la señal modulada de entrada, del filtro intermedio y de la IFFT que se utiliza para obtener la señal de salida que debe ser demodulada para obtener las muestras deseadas del espectro de la señal de entrada. A medida que se requiere una resolución en frecuencia grande la utilidad del algoritmo CHIRP se pierde porque se necesita ejecutar 3 FFT en lugar de una FFT de pocas muestras con la misma resolución en frecuencia.

La tarjeta adaptable muestra que los prototipos de evaluación pueden ser expandidos e interconectados con otros dispositivos con la finalidad de ampliar sus posibilidades. En esta oportunidad se logró disponer de 8 canales de adquisición de señales analógicas distribuidos en dos conversores analógicos digitales que tienen comunicación directa con el microcontrolador permitiendo la adquisición de dos canales analógicos de entrada de forma simultánea. Además se cuenta con la posibilidad de controlar un puerto de entrada y salida digital de 8 canales independientes y 8 canales de salida analógica. Un inconveniente presentado por la tarjeta adaptable fue la comunicación serial entre el DSP y el microcontrolador que controla la tarjeta adaptable. El mismo se soluciono utilizando subrutinas escritas por el autor para este propósito utilizando los dos canales de los dispositivos como puertos de entrada y salida digital. La velocidad de muestreo máxima lograda con los conversores analógicos digitales, 4000 Hz, serviría para adquirir señales de baja frecuencia como algunas biológicas como el ECG y para aplicaciones de señales de energía eléctrica de potencia de 60 Hz. Este problema se presenta por la utilización del microcontrolador como intermediario entre el DSP y los conversores analógicos digitales. Lo anterior puede mejorarse si se conectan directamente los conversores con el DSP.

Las prácticas de laboratorio elaboradas en el presente trabajo de investigación permiten que el futuro estudiante de la maestría o pregrado pueda familiarizarse con la programación de los algoritmos de la FFT en base 2, FFT en base 4, FFT en base mixta, FFT2SENAL1, FFT2SENAL2, Goertzel, CHIRP y filtros FIR e IIR en un procesador de señales digitales además permiten experimentar con el *hardware* de la tarjeta adaptable.

6.2. TRABAJOS FUTUROS.

Dentro de las sugerencias o posibles proyectos futuros que se pueden implementar utilizando la tecnología DSP, es la implementación de algoritmos del tipo de las ondeletas (WAVELETS) para el análisis temporal del espectro de una señal. Lo anterior con la finalidad de analizar el comportamiento de señales no estacionarias, es decir, señales que no conservan la misma frecuencia para todo tiempo.

Otro proyecto podría ser la construcción de un equipo (monitor de calidad de la energía eléctrica) que permita analizar sistemas trifásicos de potencia utilizando un DSP por cada fase, lo cual permitiría analizar por separado la tensión y la corriente de las tres fases del sistema de potencia y además la posibilidad de programar algoritmos diferentes en cada una de ellas. Esto permitirá también las técnicas de procesamiento con esquemas de identificación, caracterización y clasificación automática de eventos en tiempo real en la red eléctrica de potencia.

Un tercer proyecto nace de la posibilidad de generar un procesador FFT de base 2, 4 u 8 en la tecnología FPGA. Lo anterior debido a la gran velocidad que se logra con estos dispositivos.

Implementar en un algoritmo de la transformada rápida de Fourier FFT el cálculo de los coeficientes a través de subrutinas de desplazamiento vectorial CORDIC; lo anterior con miras a disminuir el espacio en memoria necesario para el almacenamiento de dichos coeficientes.

Factores como los números 15 y 12 se pueden llevar a cabo teniendo en cuenta que se componen de factores primos, 15 = 3 * 5 y 12 = 4 * 3, se pueden realizar sin multiplicaciones entre etapas si se escogen adecuadamente que componen a las variables de entrada n y de salida k. El inconveniente de esta técnica radica en que tanto n como k entran y salen con las direcciones de los elementos invertidas; lo

anterior necesita dos algoritmos adicionales que ingresen los datos en desorden y otro para ordenar la salida de la etapa. Los factores 12 y 15 pueden ser ejecutados de esta manera en el algoritmo de la FFT de base mixta como otro par de factores que componen un número cualquiera N.

7. BIBLIOGRÁFIA

[Oppenheim & Shafer, 00] Oppenheim, Alan y Schafer, Ronald. "*Tratamiento de señales en tiempo discreto*". Segunda edición, Pp. 904, *Prentice Hall*, Madrid, 2000.

[Proakis & Manolakis, 98] Proakis, J. G. y Manolakis, D. G. "*Tratamiento digital de señales*". Tercera edición, Pp. 1048, *Prentice Hall*, Madrid, 1998.

[Franklin et al, 90] Franklin, Gene F.; Powell, J. David ; Workman, Michael L. "*Digital control of dynamics systems*". Segunda edición, Pp. 837, *Addison-Wesley*, 1990.

[Brigham , 88] Brigham, Oran E. "*The Fast Fourier Transform And Its Applications*". Primera edición, Pp. 446, *Prentice Hall*, 1988.

[Analog Devices , 95] Analog Devices. "*ADSP-2100 Family User´s Manual*". Tercera edición, Pp. 419, *Analog Devices*, 1995.

[Smith, 99] Smith, Steven W. "*The Scientist And Engineer´s Guide to Digital Signal Processing*". Segunda edición, Pp. 643, *California Technical Publishing*, 1999.

[Oppenheim et al, 98] Oppenheim, Alan V; Willsky, Alan S; Nawab, S hamid. "*Señales y sistemas*". Segunda edición, Pp. 941, *Prentice Hall*, 1998.

[Analog Devices, 01] Analog Devices. "*ADSP-218x DSP Hardware Reference*". Primera edición, Pp. 412, *Analog Devices*, 2001.

[Analog Devices, 95] Analog Devices. "*ADSP-2100 Family EZ-KIT Lite Reference Manual*". Primera edición, Pp. 258, *Analog Devices*, 1995.

[Analog Devices, 98] Analog Devices. "*VISUALDSP DEBUGGER TUTORIAL (FOR THE ADSP-21xx FAMILY DSPs*". Segunda edición, Pp. 28, *Analog Devices*, 1998.

[Analog Devices, 00] Analog Devices. *"Mixed-Signal And DSP Design Techniques"*. Segunda edición, Pp. 400, *Analog Devices*, 2000.

[Volder , 59] Volder J. E. *"The CORDIC trigonometric computing technique*" *IRE Trans. Electron.* vol. EC-8, no. 3, pp. 330-334, Sept. 1959

[Motorola, 02] Motorola. *"MC68HC908GP32 Technical Data*". Pp. 410, Available in: www.freescale.com, Motorola, 2002.

[Motorola, 96] Motorola. *"CPU08 Central Proccesor Unit. Referente Manual".* Pp. 274, Available in: www.freescale.com, Motorola, 1996.

[Analog Devices, 00] Analog Devices. *"AD8534, Low Cost, 250 mA Output, Single Suply Amplifier ".* Pp. 16, Available in: www.analog.com, Analog Devices, 2000.

[Analog Devices, 99] Analog Devices. "*REF192, Precision Micropower, Low Dropout, Voltage Reference ".* Pp. 23, Available in: www.analog.com, Analog Devices, 1999.

[Analog Devices, 99] Analog Devices. *"AD5204, 4 / 6 Digital potentiometer".* Pp. 11, Available in: www.analog.com, Analog Devices, 1999.

[Analog Devices, 99] Analog Devices. *"AD974, 4 channel, 16 bit, 200 Ksps, Data acquisition system ".* Pp. 20, Available in: www.analog.com, Analog Devices, 1999.

[Analog Devices, 99] Analog Devices. *"AD7398, Quad, Serial-Input, 12 bits, DAC".* Pp. 11, Available in: www.analog.com, Analog Devices, 1999.

[National Semiconductor, 99] National Semiconductor. *"LM7800, Series 3 Terminal Positive Terminal".* Pp. 15, Available in: www.natsemi.com, National Semiconductor, 1999.