Procedimiento de apoyo en hardware para un sistema de reconocimiento de voz

UNIVERSIDAD DEL VALLE DEGUATEMALA

Facultad de Ciencias y Humanidades

Procedimiento de apoyo en hardware para un sistema de reconocimiento de voz

Trabajo de investigación presentado por Pedro Miguel Viscovich Siitari para optar al grado de Ingeniero en Electrónica

Guatemala

2002

vo. BO.

Asesor: Dr. Ing. Manuel Antonio López Valdez

Tribunal:

Dr.Ing. Manuel Antonio López Valdez

Ing. Gonzalo Antonio Palarea Murga

Ing. Julio Ismael Vázquez Vargas

Fecha de Aprobación: 2 de julio del 2002

RESUMEN

Este trabajo de investigación se enfoca en la utilización de la transformada rápida de Fourier (FFT) integrada en un microcontrolador para un sistema de reconocimiento de voz por medio de patrones frecuenciales. El problema tratado en este trabajo consiste primordialmente en la integración de la transformada rápida de Fourier dentro del microcontrolador, con el objetivo de reducir el tiempo de procesamiento de dicha transformada; así como verificar el funcionamiento adecuado por medio de los resultados obtenidos al realizar la transformada.

Al integrar todos los pasos y cálculos necesarios para realizar la transformada en hardware, se presentan ciertos problemas como lo son la utilización del punto fijo y la falta de resolución, tanto en decimales como en enteros, para representar fielmente los valores numéricos necesarios para realizar la transformada con mayor precisión. Se observa también que, al realizar la normalización de la señal compuesta de entrada y su respectivo muestreo, se pierde la forma original de la señal debido que el muestreo de la misma fue realizado exactamente al doble de la frecuencia máxima de la función.

Se demuestra, por lo tanto, que realizar la transformada rápida de Fourier en el microcontrolador es posible, pero se sacrifican ciertos aspectos como el tiempo de operación y ciertos errores de aproximación que, dependiendo del caso de la aplicación que se le quiera dar, pueda incurrir en una cantidad de errores que conduzcan a un resultado desfavorable de dicha aplicación.

ÍNDICE

Página

LISTA DE CUADROS Y GRÁFICAS

Cuadro Página

Gráfica

LISTA DE FIGURAS

I. INTRODUCCIÓN

Para que la tecnología sea aceptada por un grupo cada vez mayor de usuarios, es necesario que la interacción con el humano se lleve a cabo de forma natural y accesible a las funciones innatas del hombre. La voz, es probablemente la herramienta más poderosa que el hombre conoce, y fue su desarrollo el que permitió intercambiar ideas entre humanos al evolucionar el lenguaje.

Siendo esta herramienta tan poderosa, es muy curioso que la misma encuentre tan poca implementación para la comunicación con sistemas técnicos. Por el contrario, se pueden observar tendencias con interfaces humanas poco naturales como son los teclados minúsculos para computadoras de bolsillo cada vez más compactas.

Las razones residen en la dificultad técnica de reconocer el lenguaje humano por la voz de forma inequívoca e independiente del individuo. Por lo visto, imitar la función humana de reconocimiento de voz, que parece tan natural, es un desafío técnico de grandes proporciones.

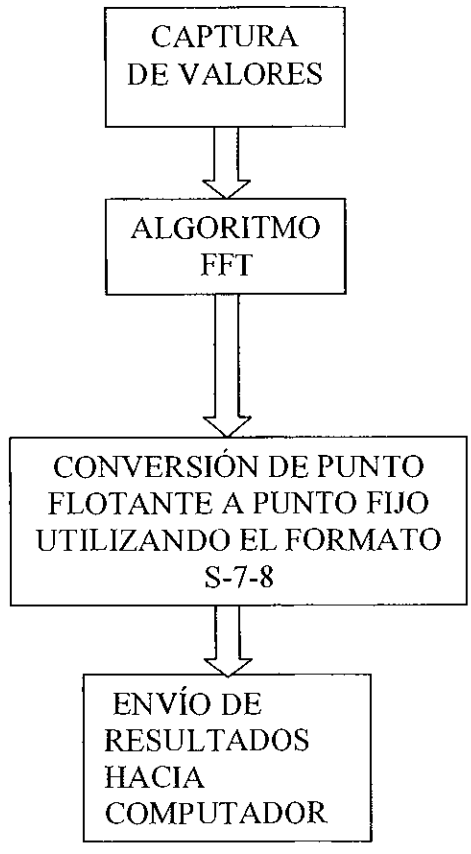
Las investigaciones en este campo se han concentrado tradicionalmente en el reconocimiento de patrones de voz en el régimen de tiempo. Es decir, se analizan las señales que se generan con el registro de ondas sonoras por micrófonos de acuerdo a la secuencia en que aparecen en el tiempo. Los métodos desarrollados son discrecionalmente complejos pero, por lo visto, no han encontrado amplia difusión en la tecnología de interacción.

Un método alterno de reconocimiento ofrece el régimen de frecuencia. El fundamento de estos métodos son las transformadas de señales de tiempo a señales de frecuencia (p.e. la transformada de Fourier). El método pretende reconocer los patrones de voz, no por la secuencia en que aparecen en el tiempo, sino por la configuración de frecuencias empleadas para reproducir el fonema a reconocer.

El presente trabajo, se encuentra anidado en un trabajo de investigación del departamento de electrónica de la Universidad del Valle de Guatemala, que busca un nuevo enfoque que reúna las siguientes propiedades:

- reconocimiento de voz por medio de frecuencias fonéticas,
- portabilidad de soluciones evitando tener que usar computadoras para el eficaz reconocimiento de la voz,
- reconocimiento de voz en tiempo real y
- solución de bajo costo.

Figura No. 1: Proceso requerido para realizar la transformada de Fourier



La primera propiedad no es objeto de investigación del presente trabajo. Sin embargo, la misma requiere del resultado práctico de este estudio para poder cumplir con los otros objetivos.

Para lograr un reconocimiento portable y en tiempo real, es necesario recurrir a la tecnología de microcontroladores. Para cumplir con el último requisito, se optó por investigar la posible realización de un sistema de reconocimiento de voz, utilizando microcontroladores de bajo costo.

Estos últimos tienen la desventaja de poseer muy pocas funciones matemáticas implementadas, las que tienen que ser emuladas por pequeños paquetes de software, que a su vez presentan problemas de consumo de tiempo.

El objetivo científico del presente trabajo es investigar la viabilidad de realización de la así llamada transformada rápida de Fourier en un microcontrolador de bajo costo, para ser usada en un sistema de reconocimiento de voz en el régimen frecuencial. Si bien los resultados obtenidos no realizan las funciones esperadas, el resultado práctico consiste en la argumentación de todas las limitaciones encontradas para la realización de tan exigente proyecto.

II. ANTECEDENTES

La transformada de Fourier especifica el contenido espectral de una señal contínua no periódica, proporcionando así una descripción en el dominio de la frecuencia de la misma. De especial interés resulta la transformación de señales discretas en el tiempo, es decir, señales que consisten en una sucesión de valores discretos provenientes del muestreo de una señal continua en tiempo. El procesamiento de este tipo de señales resulta importante dado su fácil manejo y representación.

Para el caso de señales discretas en el tiempo, se desarrolló lo que hoy se conoce como la transformada discreta de Fourier (DFT por sus siglas en inglés), cuyos resultados brindan una gran cantidad de información sobre el contenido frecuencial de la onda discreta que se está analizando. Sin embargo, el algoritmo de determinación de la transformada discreta de Fourier es exhaustivo, y requiere de una gran cantidad de cálculos. Aún y cuando es posible desarrollar el algoritmo correspondiente e implementarlo como una rutina de software, la naturaleza del algoritmo no permite que los cálculos se ejecuten eficientemente, aumentando de forma considerable el tiempo de procesamiento necesario para realizar la transformación. Lo anterior motivó al desarrollo de un algoritmo mucho más eficiente, comúnmente conocido como la transformada rápida de Fourier (FFT por sus siglas en inglés), que fue ideado como solución al problema computacional del algoritmo original de la DFT, aún y cuando los resultados obtenidos a través de este algoritmo coinciden exactamente con los que se obtendrían por medio de una DFT, con la ventaja de minimizar el tiempo de procesamiento. Para la correcta aplicación del algoritmo de FFT, la secuencia de datos de entrada contiene un número 2^n de datos. En caso se requiera del procesamiento de una secuencia que no cumpla con este requisito, deberán llenarse de ceros todas las posiciones que hagan falta para lograr que el número de elementos sea del tipo 2^n.

Actualmente, existe una necesidad marcada por minimizar aún más el tiempo de procesamiento de la transformada rápida de Fourier; dado que muchas aplicaciones de diversa índole necesitan cada vez más de los resultados que esta transformada proporciona.

Uno de los principales problemas al implementar este tipo de herramientas en rutinas de software es el tiempo que el microprocesador debe dedicar a la aplicación del algoritmo, dejando en segundo plano el procesamiento de señales que pueden ser cruciales para la aplicación que se esta desarrollando. Como solución a este problema, se propone la implementación del algoritmo de FFT en un microcontrolador, con una interfase hacia una computadora personal. De esta manera se logra tener un microcontrolador dedicado única y exclusivamente a la aplicación del algoritmo de FFT, descargando significativamente la tarea del microprocesador central, que puede dedicarse enteramente al manejo de alguna aplicación específica.

III. DETERMINACIÓN DEL PROBLEMA

Se pretende reducir el tiempo de procesamiento de un algoritmo de transformada rápida de Fourier, implementándolo en un microcontrolador dedicado exclusivamente a su cálculo. Existen varias formas para lograr una comunicación entre una computadora personal y un microcontrolador. A esto se le conoce como interfase. Existen muchas diferentes formas de interfases de las cuales las principales diferencias consisten en velocidades, medios de transmisión y número de conductores. El microcontrolador tendrá una interfase a una computadora personal de la cual recibe los datos a ser transformados y a la cual envía los resultados de la operación.

A. Objetivos Específicos:

1. Demostrar que es posible realizar una transformada rápida de Fourier (FFT) de 4096 muestras en un microcontrolador.
2. Comprobar la reducción de tiempo de cálculo al comparar el procedimiento implementado en el microcontrolador con una rutina de software ejecutada en una computadora personal.
3. Aplicar la herramienta desarrollada al procesamiento de señales de voz.

IV. MARCO TEÓRICO

A. Transformada de Fourier

La transformada de Fourier viene de un caso límite de la serie de Fourier en su forma exponencial:

$$f(t) = \sum_{n=-\infty}^{\infty} C_n e^{jnw_0 t}$$

Donde,

$$C_n = \frac{1}{T} \int_{-T/2}^{T/2} f(t) e^{-jnw_0 t} dt$$

Si se considera el caso en que el período fundamental T tiende al infinito, la función deja de ser periódica, y los coeficientes de la serie de Fourier (C_n) tienden a cero. De esta manera, considérese el producto C_nT, en el límite cuando T tiende al infinito:

$$C_n T \rightarrow \int_{-\infty}^{\infty} f(t) e^{-jwt} dt \text{ , cuando T tiende al infinito}$$

La integral descrita es la Transformada de Fourier, la cual se define formalmente como:

$$F(w) = \Im\{f(t)\} = \int_{-\infty}^{\infty} f(t) e^{-jwt} dt$$

B. Transformada discreta de Fourier

La DFT (Transformada Discreta de Fourier) es un procedimiento matemático utilizado para determinar el contenido frecuencial de una secuencia discreta, formada por el muestreo de una señal continua en el tiempo. Se origina en la definición de la transformada de Fourier. Se define la DFT como:

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nm/N}$$

donde x(n) es una secuencia discreta de valores muestreados en el dominio del tiempo de una función continua x(t). Para entender mejor la DFT, su definición suele expresarse en términos de senos y cosenos, a través de la identidad de Euler. Así:

$$e^{\pm j\phi} = \cos(\phi) \pm jsen(\phi)$$

Al sustituir la relación de Euler en la DFT, obtenemos lo siguiente:

$$X(m) = \sum_{n=0}^{N-1} x(n) \left[\cos\left(\frac{2n\pi m}{N}\right) - jsen\left(\frac{2n\pi m}{N}\right) \right]$$

Donde m es el índice de la salida de la DFT en el dominio de la frecuencia (m=0,1,2,...,N-1) y n es el índice del dominio del tiempo de las muestras de entrada (n=0,1,2,...,N-1).

Se logra así la separación del exponente complejo en sus partes real e imaginaria. El valor N es importante en este caso, ya que nos indica cuántas muestras de entrada son necesarias, la resolución de los resultados en el dominio de la frecuencia, y la cantidad de tiempo de procesamiento necesario para realizar la DFT.

Cada término X(m) es la suma del producto punto a punto entre los valores de una señal de entrada y un sinusoide complejo de la forma $\cos(\phi) \pm j sen(\phi)$. Las frecuencias exactas de los diferentes sinusoides dependen de la frecuencia de muestreo f_s y el número de muestras N. Por ejemplo, dada una frecuencia de muestreo de 5,500 muestras por segundo, al realizar una DFT de 4096 muestras, la frecuencia fundamental de los sinusoides está dada por:

$$\frac{f_s}{N}$$

Así, en el espectro de amplitudes se obtendrá un valor de amplitud para cada múltiplo entero de este valor. En el ejemplo anterior, la frecuencia fundamental es de 1.34 Hz. Los siguientes resultados de X(m) son múltiplos de esta frecuencia fundamental, es decir 0 Hz, 1.34 Hz, 2.68 Hz, 4.02 Hz, etc. La información obtenida en X(0) indica la magnitud de cualquier componente de 0 Hz (DC) contenida en la señal de entrada, X(1) especifica la magnitud de cualquier componente a 1.34 Hz en la señal de entrada, etc.

C. Propiedades de la DFT

1. La secuencia de tipo de entrada para la DFT que generalmente se presenta es de tipo real. Dado que la transformada discreta recibe como secuencia de entrada una secuencia de números complejos, la parte real de éstos coincidirá con la secuencia que se pretende transformar, mientras que la parte imaginaria de todos ellos será igual a cero.

En este caso, los datos obtenidos para m>=N/2 son redundantes. Esta característica puede apreciarse al analizar la forma exponencial de la DFT. Si se sustituye m por N-m, se obtiene la expresión de la DFT para el (N-m) componente frecuencial:

$$X(n-m)=\sum_{n=0}^{N-1}x(n)e^{-j2\pi n(N-m)/N}=\sum_{n=0}^{N-1}x(n)e^{-j2\pi nN/N}e^{-j2\pi(-m)/N}=\sum_{n=0}^{N-1}x(n)e^{-j2\pi n}e^{j2\pi nm/N}$$

Como $e^{-j2\pi n}=\cos(2\pi n)-jsen(2\pi n)=1$ para todos los valores enteros de n, se obtiene

$$X(N-m)=\sum_{n=0}^{N-1}x(n)e^{j2\pi nm/N}$$

Es evidente que X(N-m) es igual a X(m) con el signo revertido en el exponente de X(m). Esta es la definición del complejo conjugado.

Otra propiedad de simetría de la DFT es que si la función real de entrada x(n) es par, entonces X(m) será siempre real y par. Si la función real de entrada x(n) es impar, entonces X(m) será siempre imaginaria e impar.

2. Una de las propiedades más importantes de la DFT es su linealidad. Ésta establece que: La DFT de una suma de dos señales es equivalente a la suma de la DFT de ambas señales por separado. Sin esta propiedad, la DFT sería inútil debido a que sólo permitiría analizar señales de entrada que contengan una sola senoidal, restringiendo drásticamente su aplicabilidad.

D. Teorema de Muestreo y *Aliasing*

En las situaciones más comunes, la función h(t) está muestreada a intervalos iguales de tiempo.

Considerando a Δ como el intervalo de tiempo entre muestras consecutivas, el recíproco de Δ es llamado frecuencia de muestreo. Para cualquier intervalo de muestreo

Δ existe una frecuencia especial f_c, llamada la frecuencia crítica de Nyquist, la cual está descrita por:

$$f_c = \frac{1}{2\Delta}$$

El muestreo crítico de una función es dos puntos por ciclo. La frecuencia crítica de Nyquist es importante por dos razones principales. Si una función continua h(t) muestreada a un intervalo Δ está limitada en ancho de banda a frecuencias menores a la frecuencia crítica de Nyquist, entonces h(t) está descrita completamente por sus muestras. Si, por alguna razón, muestreamos una función contínua que no está limitada por banda, toda la energía que está fuera del rango descrito por:

$$-f_c < f < f_c$$

se mueve en forma de espurios dentro de este rango. A este efecto se le conoce como *Aliasing*. Una vez ya se haya discretizado una señal, ya no se puede hacer nada para remover esta información. La manera de evitarlo es saber el ancho de banda natural de la señal, o forzar un límite por medio de un filtro análogo aplicado sobre la señal continua, para luego muestrear a una velocidad lo suficientemente rápida para poder obtener por lo menos dos puntos por ciclo de la frecuencia más alta presente.

E. Transformada rápida de Fourier (FFT por sus siglas en inglés):

Aunque la DFT es el procedimiento matemático directo para determinar el contenido frecuencial de una secuencia en el dominio del tiempo, es demasiado ineficiente al considerar el número de operaciones matemáticas involucradas en el algoritmo.

Conforme la cantidad de puntos a ser analizados por la DFT aumentan a cientos o miles, la cantidad de manipulación de números se vuelve excesiva. La DFT requiere de N^2 multiplicaciones complejas para poder realizarse.

En 1965 Cooley y Tukey describieron la existencia de un algoritmo muy eficiente para implementar la DFT. Este algoritmo se conoce como la FFT (Transformada rápida de Fourier). Aunque varios algoritmos de la FFT han sidos desarrollados, el más popular de todos es el algoritmo *radix-2 FFT*. Este algoritmo es muy eficiente en realizar la DFT bajo la condición de que el tamaño de la DFT tiene que ser múltiplo de 2, es decir

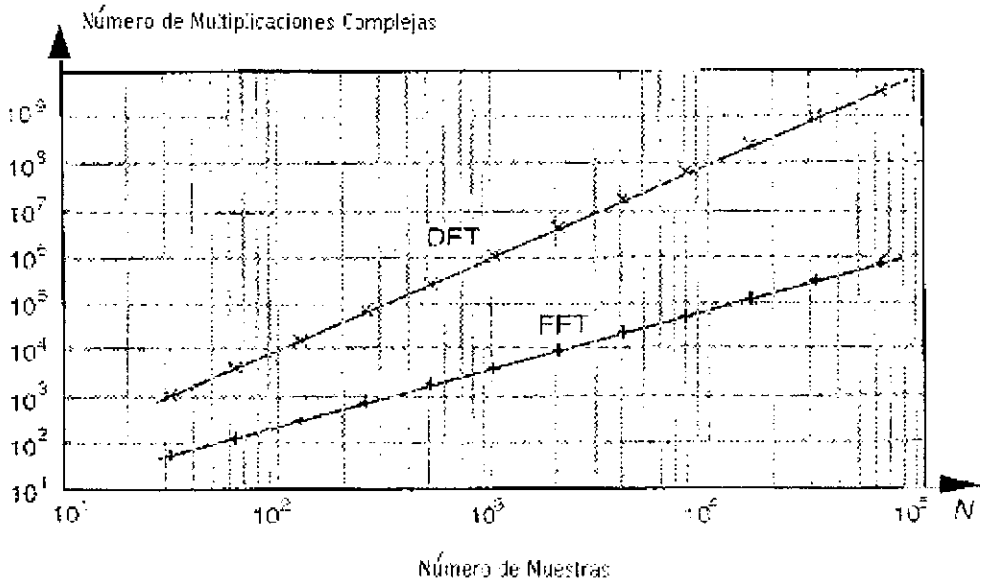
$$N = 2^k$$

donde k es un entero positivo. La cantidad de operaciones necesarias para realizar la FFT se calcula con la siguiente formula:

$$N \log_2 N$$

El cambio es sumamente significativo. Por ejemplo para $N = 10^6$, el tiempo de procesamiento en una computadora de microsegundo es más o menos 30 segundos para la FFT y 2 semanas para la DFT.

Figura No.2: Comparación de eficiencia entre DFT y FFT



(Lyons, 1997)

La figura anterior compara la eficiencia entre la determinación de la DFT utilizando su definición y utilizando el algoritmo de FFT. Nótese como mientras aumenta el número de muestras (N), el número de multiplicaciones complejas que se requieren para el cálculo de la DFT a través de su definición es considerablemente mayor (aproximadamente 10^3 veces para 10^5 muestras) que si se utilizara el algoritmo de FFT. Mientras mayor sea el número de muestras, más se evidencia la ventaja de utilizar los procedimientos de FFT.

En 1942 Danielson y Lanczos demostraron que una transformada discreta de Fourier de tamaño N puede ser rescrita como la suma de dos transformadas discretas de Fourier, cada una de largo N/2. Una de las dos esta formada por los puntos pares de N, mientras que la otra está formada por los puntos impares de N. Esto se conoce como el lema de Danielson-Lanczos.

$$F_k = \sum_{n=0}^{N-1} e^{2\pi njk/N} f_n = \sum_{n=0}^{N/2-1} e^{2\pi jk(2n)/N} f_{2n} + \sum_{n=0}^{N/2-1} e^{2\pi jk(2n+1)/N} f_{2n+1}$$

$$F_k = \sum_{n=0}^{N/2-1} e^{2\pi jkn/(N/2)} f_{2n} + e^{2\pi j/N} \sum_{n=0}^{N/2-1} e^{2\pi jkn/(N/2)} f_{2j+1}$$

donde k se extiende de 0 a N. La ventaja del teorema de Danielson-Lanczos es su recursividad. Esto quiere decir que es posible seguir subdividiendo la transformada en partes cada vez más pequeñas y por lo tanto más simples de operar. El patrón de este lema se puede mostrar en un diagrama, llamado diagrama de Mariposa.

Figura No.3 Diagrama de mariposa, FFT de ocho puntos.

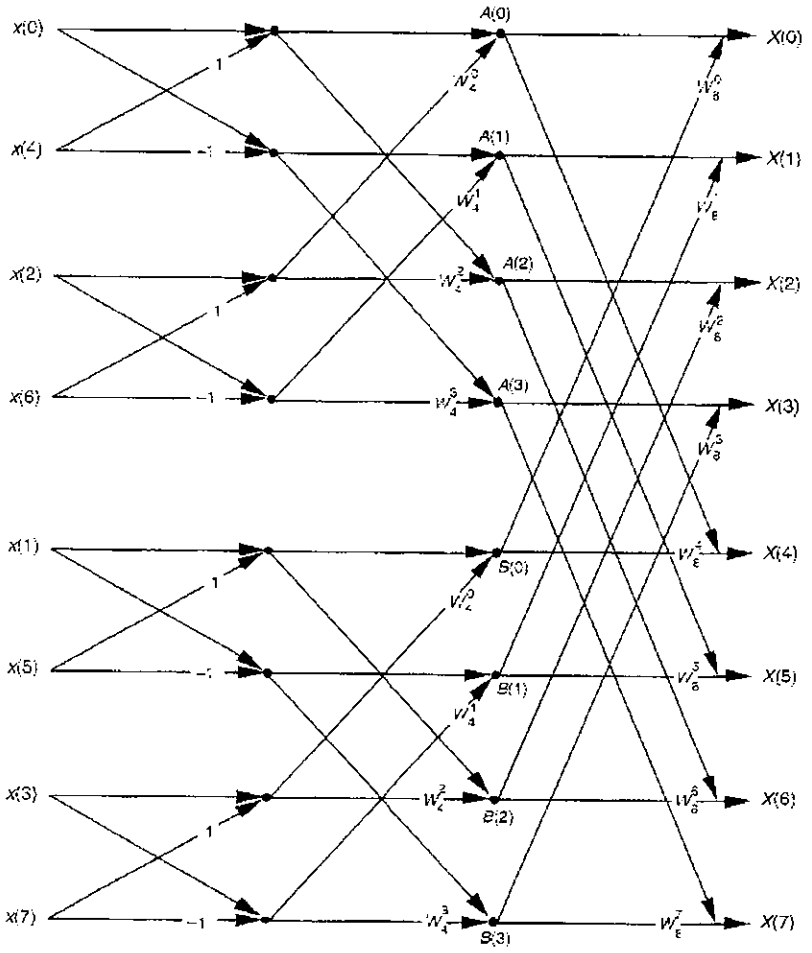


Diagrama de Mariposa para una FFT de 8 puntos

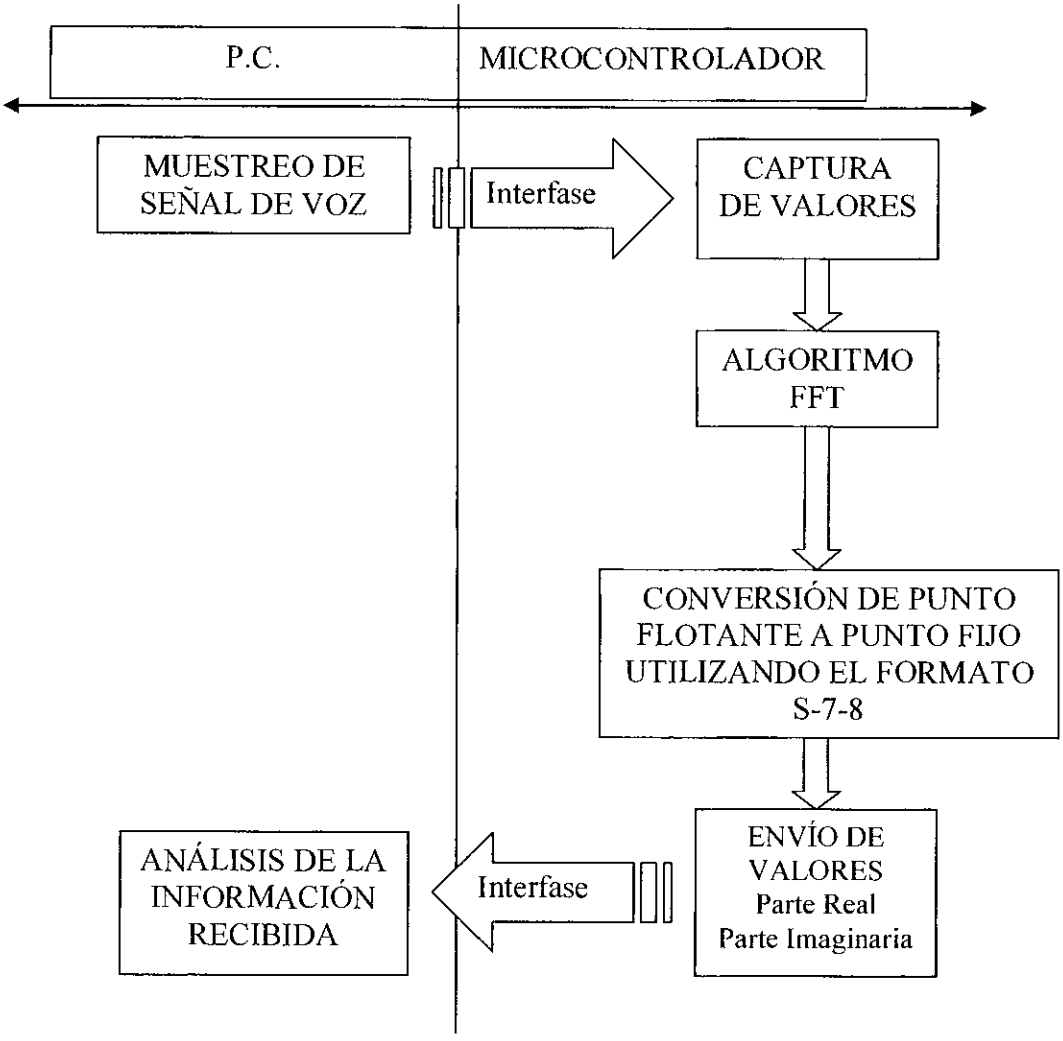
(Lyons, 1997)

V. METODOLOGÍA

Como se discutió con anterioridad, el algoritmo implementado puede tener diversas aplicaciones, según el uso que se pretenda dar a los resultados de la transformada de Fourier. En este caso, la herramienta desarrollada funciona como soporte en un proyecto de reconocimiento de voz desarrollado en forma paralela (mismo que se fundamenta en la información espectral de una señal de voz previamente muestreada). El sistema implementado se muestra en el siguiente diagrama:

Figura No.4

Esquema general de integración del proyecto:



Antes de definir el problema de la representación de numeros en el microcontrolador, es necesario definir los siguientes términos: Punto Flotante y Punto Fijo. La principal diferencia, y por lo tanto limitación del punto fijo en referencia con el punto flotante es su incapacidad de hacer que el punto "flote", es decir que pueda moverse dinámicamente a lo largo de un número para poder mostrar más decimales o menos decimales conforme sea necesario para poder representar con mayor fidelidad un valor determinado. En un formato de punto fijo, se tiene definida la cantidad de dígitos que representan la parte entera y la parte decimal de un número. Por ejemplo, si se define cuatro bits para representar la parte entera, y ocho bits para representar la parte decimal, el resultado de una operación como la siguiente

$$153/7 = 21.8571428571$$

tendrá el siguiente resultado en formato punto fijo:

$$5.85$$

Esto se debe a la incapacidad del punto decimal de moverse de su lugar, para acomodar una mayor cantidad de bits para representar el número entero.

Unas de las ventajas de los puntos fijos es que a diferencia del punto flotante, los cálculos se pueden realizar con mayor velocidad y requieren una cantidad menor de memoria de almacenamiento.

Al realizar la transformada de Fourier, se necesita de una gran cantidad de números que cuentan con decimales. Debido a que no es posible representar un número decimal en un microcontrolador, es necesario utilizar un formato de punto fijo que describa el número en forma binaria.

Debido a esta limitación es necesario escoger desde un principio la cantidad de valores binarios utilizados para representar los decimales y la cantidad de valores binarios utilizados para representar partes enteras de valores.

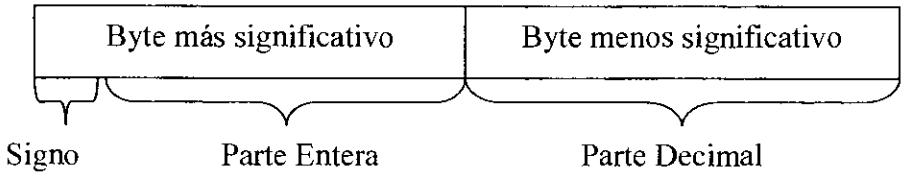
Debido a que todo el procedimiento está basado en 16 bits, el formato a utilizar por considerarse adecuado tiene la forma de

S-7-8

Donde S representa el signo, siete representa la cantidad de bits utilizados para la parte entera, y ocho representa la cantidad de bits usados para los decimales.

Figura No. 5

Esquema de posicionamiento de bits



Así, el séptimo bit del byte más significativo se utiliza para representar el signo (1= - , 0=+); del sexto bit hasta el bit cero, se utilizan para representar la parte entera del número. El byte menos significativo se utiliza enteramente para representar la parte decimal del número; así:

$$\text{Parte Decimal} = \text{bit } 7 * 2^{-1} + \text{bit } 6 * 2^{-2} + \ldots\ldots + \text{bit } 0 * 2^{-7}$$

De esta forma, el máximo número decimal posible de representación es: 0.99 (equivalente a 0.99609375, con dos cifras significativas).

Como requisito por parte de la aplicación de reconocimiento de voz a la cual se está aplicando, se requiere del almacenamiento y procesamiento de 4096 muestras, cada una con una resolución de 8 bits (cada muestra ocupa un byte de memoria). Por esta razón, se debió implementar una memoria externa de tipo RAM (Random Access Memory), dada la

insuficiencia de memoria propia del microcontrolador para manejar este número de muestras. La memoria que se utilizó fue IS61C256AH de Integrated Silicon Solution Inc. la cual tiene capacidad para 32,768 bytes de RAM de alta velocidad.

La memoria seleccionada está diseñada para implementarse en conjunto con un microprocesador. Esto no resulta en un obstáculo, debido a la suficiente cantidad de puertos de entrada y salida del microcontrolador. Fue suficiente asignar dos puertos (uno de ocho y otro de cinco bits) para el manejo del bus de direcciones, y un puerto (de ocho bits) para el manejo del bus de datos (Ver Anexo C).

La principal desventaja de utilizar una memoria externa, es la cantidad de puertos del microcontrolador que tienen que ser asignados para manejar el control del bus de direcciones y el bus de datos del circuito integrado. Esto requiere un manejo más complejo de la información que entra y sale del microcontrolador, ya que para que el proceso sea más eficiente, todas las rutinas de manejo de información se tienen que poder aplicar en todos los casos que se requiera leer o transmitir algo a la memoria externa. Otra desventaja que se presenta es la cantidad terminales que tienen que ser interconectadas entre el microcontrolador y la memoria, haciendo más fácil cometer un error a la hora de conectar todo.

A. Consideraciones de la interfase de comunicación

Actualmente existen diversos tipos de protocolos de comunicación para lograr el intercambio de información con una computadora personal. Las diferencias entre uno y otro son principalmente sus velocidades de transmisión y la cantidad de conductores que caracterizan las respectivas capas físicas de la arquitectura. El siguiente cuadro resume las principales características de las opciones de interfase consideradas para este proyecto.

Cuadro No.1

Opciones para interfase

INTERFASE	Máxima Velocidad de Transmisión (bits/seg)	Número de Conductores, (capa física)	Disponibilidad en Microcontroladores	Comunicación Síncrona o Asíncrona
SERIAL RS-232	115,200 bps*	3	Sí	Síncrona y Asíncrona
PARALELO IEEE-1284	50 a 100 KBytes/seg.	10	No	Síncrona
USB Low Speed	1.5 Mbps	4	No	Síncrona
Ethernet 100 Base T	100 Mbps.	8	No	Síncrona

* Velocidad limitada por puerto serial de computadora y no por capacidad del microcontrolador, que tiene capacidad de transmisión de hasta 1.5 Mbps.

A pesar que la mayoría de interfases no se encuentran incorporadas en los microcontroladores disponibles en el mercado, es posible implementar las interfases de forma externa. Sin embargo, dada la naturaleza experimental de este desarrollo, fue necesario elegir como interfase de transmisión el protocolo RS-232, ya que al implementar una memoria RAM adicional (por el requisito de procesamiento de la FFT, 4096 muestras) se utilizan la mayor parte de los puertos de entrada y salida para el manejo de los buses de datos y direcciones, imposibilitando la implementación de una interfase externa de cualquiera de los tipos mencionados en el Cuadro No.1.

Para el cumplimiento de todos los requerimientos necesarios, se utilizó un microcontrolador de Microchip PIC18C452. Se utilizo este microcontrolador debido a su alta velocidad (40 MHz), la posibilidad de emular su funcionamiento utilizando una computadora personal, el amplio set de instrucciones y la capacidad de realizar una multiplicación de dos bytes en *hardware* utilizando únicamente un ciclo de reloj. Realizar una multiplicación en un microcontrolador utiliza una gran cantidad de tiempo si no esta implementada en *hardware* ya que es necesario escribir el código para el manejo de la multiplicación, y ésta utiliza varios ciclos de reloj. Esta última característica resulta necesaria para los propósitos del algoritmo de la FFT, dada su estructura.

El cuadro siguiente muestra una comparación entre los diferentes dispositivos Microchip considerados para esta aplicación.

Cuadro No.2

Comparación de dispositivos Microchip considerados.

Características	PIC18C242	PIC17C44
Frecuencia de operación (MHz)	40	33
Memoria de programa (KBytes)	32	8
Memoria de datos (Bytes)	1536	454
Interruptos	17	11
Puertos de entrada/salida	5	5
Temporizadores	4	4
PWM	4	2
Comunicación serial	Sí	Sí
Puerto paralelo esclavo	Sí	No
Convertidores análogo digital	8	0
Número de instrucciones	75	58

De las observaciones del cuadro anterior, y tomando en consideración la posibilidad de simulación disponible, se optó por el PIC18C242. Se restringió la selección a estas dos

posibilidades porque son los únicos microcontroladores PIC disponibles que implementan la multiplicación en *hardware*.

B. Descripción del algoritmo FFT

El algoritmo se muestra en el siguiente diagrama de flujo.

Figura No. 6

Diagrama de flujo de la FFT

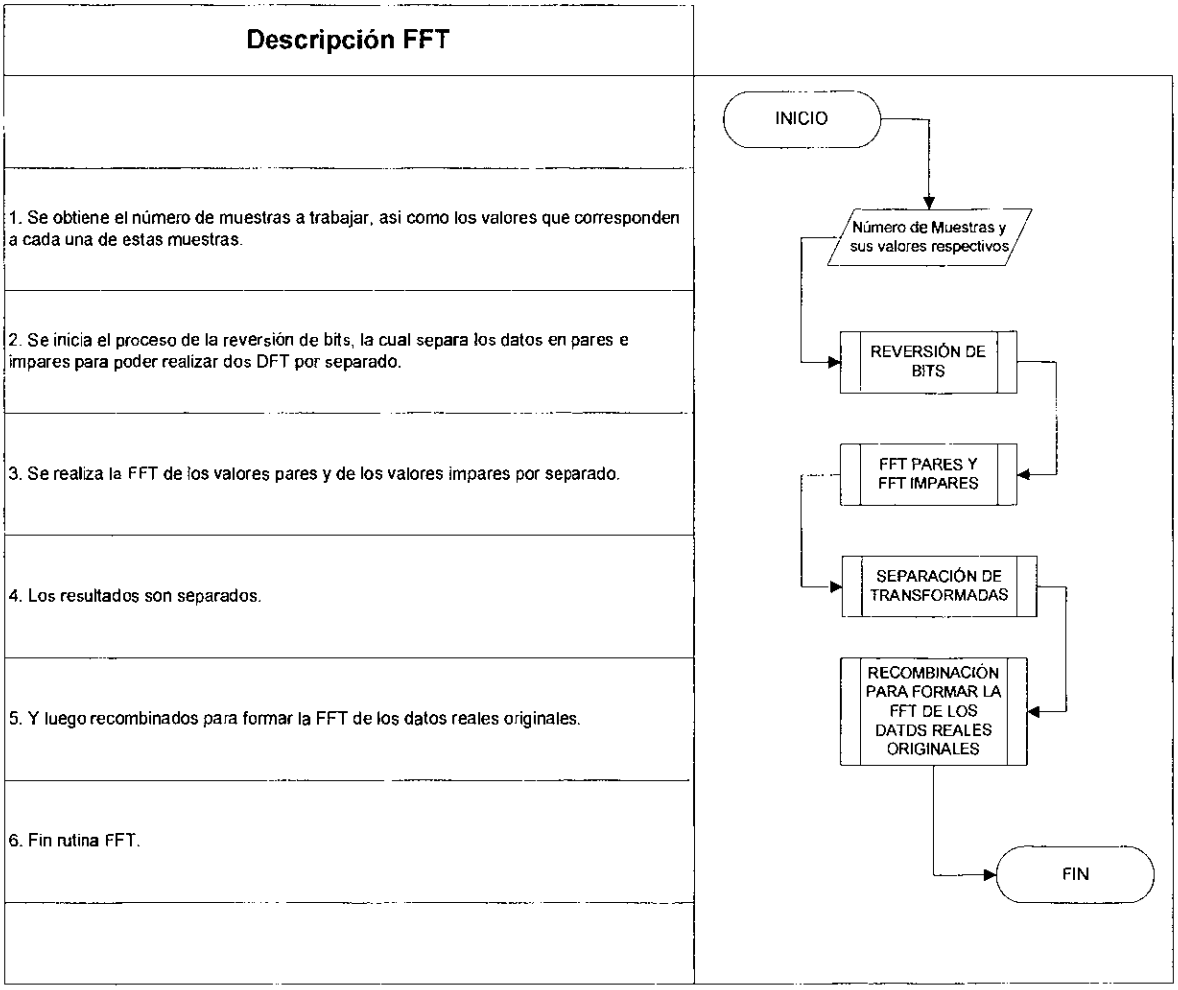
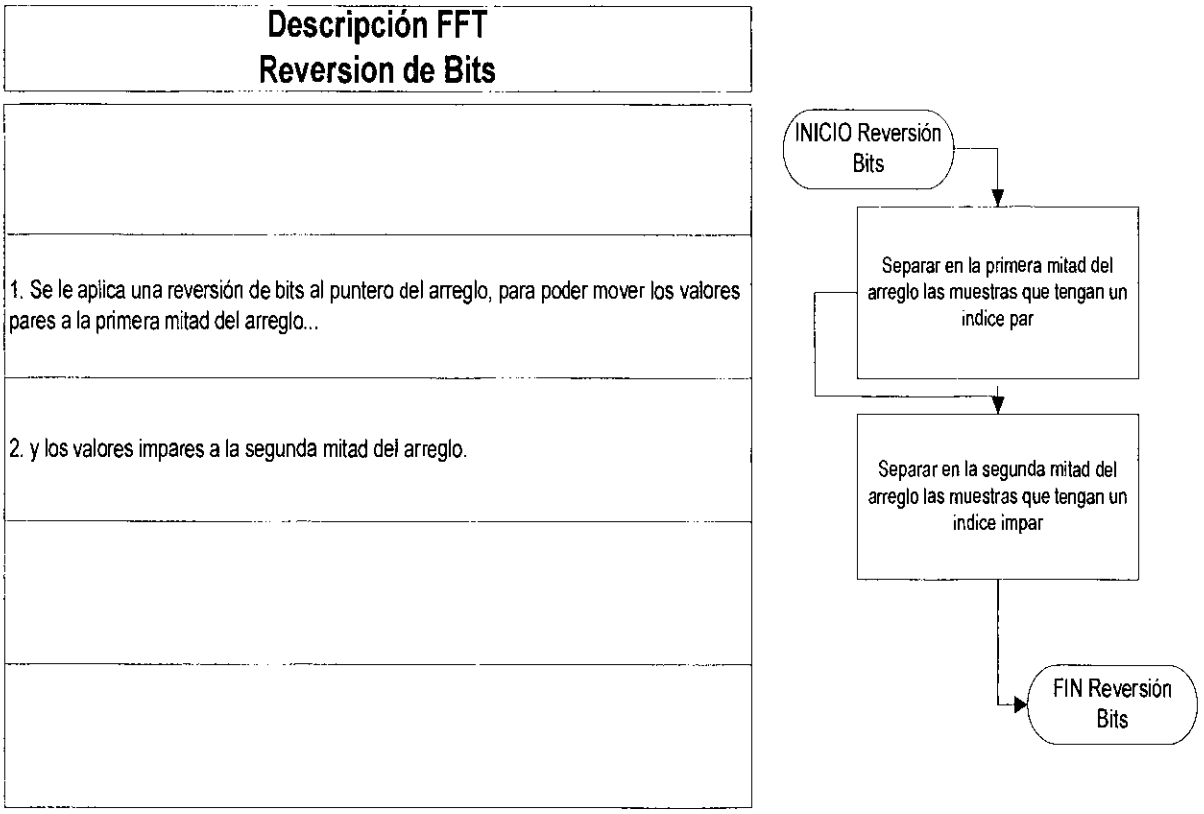


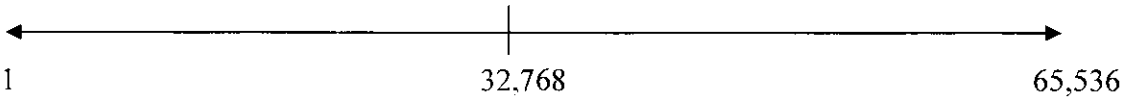
Figura No. 7

Procedimiento de reversión de bits

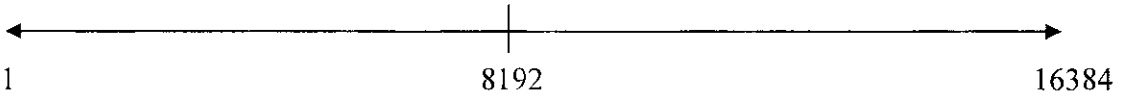


La primera parte del algoritmo constituye una fase de normalización de datos. Los datos enviados desde la computadora por interfase serial vienen codificados en un byte, que son el resultado de una conversión análogo-digital realizada por la tarjeta de sonido de la computadora. Dado que el microcontrolador asigna en memoria 2 bytes para cada muestra, se asigna el valor recibido a la parte alta, mientras que a la parte baja se le asigna cero. Es necesario normalizar estos valores (ubicarlos en una escala de -1 a 1 según el formato de punto fijo) para evitar que los resultados de la transformada sean números muy grandes, cuya representación binaria exceda el número de bits previsto por el formato de punto fijo.

Al inicio, todas las muestras obtenidas se ubican a lo largo de la siguiente escala numérica:



Cada valor se divide entre 4, de tal forma que las muestras se ubican ahora en la escala:



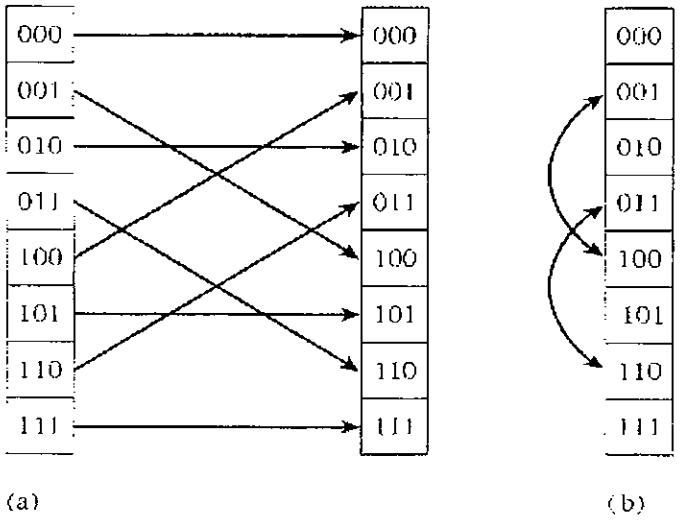
En este punto, se resta 8192 a cada número, para el caso de números menores a 8192, se invierte el orden de la resta, asignando un "1" lógico al bit 7 del byte más significativo (lo que significa que el resultado es negativo). El resultado es luego rotado 5 veces hacia la derecha (equivalente a dividir entre 32), con lo cual se ubican efectivamente todas las muestras entre -1 y 1, según el formato de punto fijo establecido (signo, 7 bits de parte entera y 8 bits de parte decimal).

Luego de la normalización, se realiza una parte fundamental de la FFT que es la reversión de bits (*bit reversal*). Esto se realiza para poder separar en números pares e impares el arreglo de datos de entrada a la FFT. La manera de trabajar la FFT es la siguiente:

Un arreglo de valores reales es separado en "pares" e "impares". Los pares son tomados como la parte real de un número complejo, y los impares son tomados como la parte imaginaria de un número complejo.

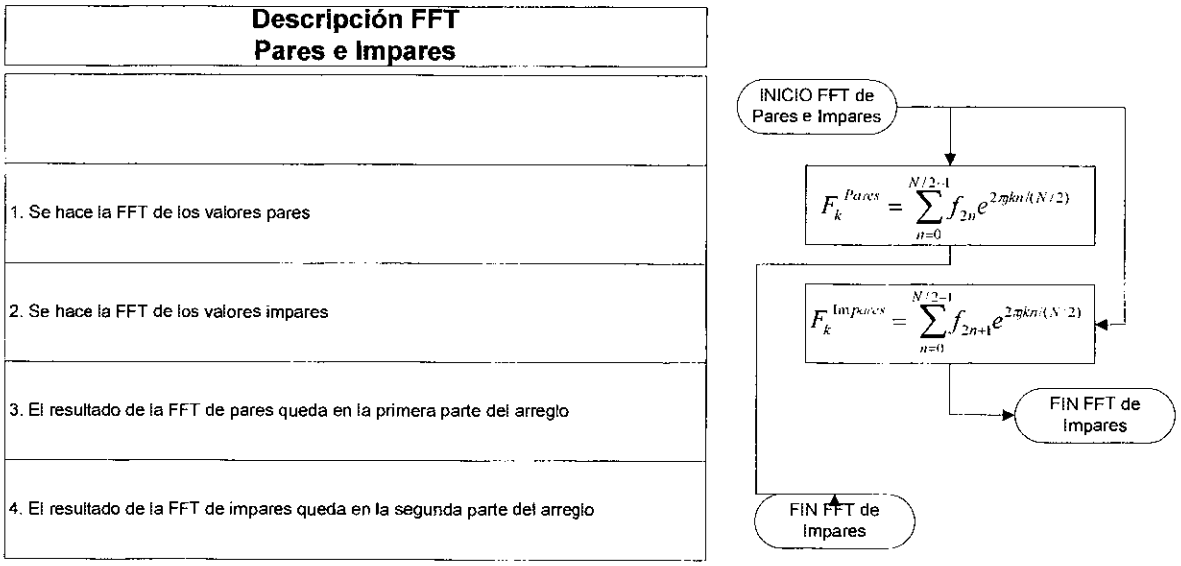
En la figura 5(a) se puede observar cómo es el procedimiento para realizar la reversión de bits. En la figura 5(b) se observa comó queda el arreglo de datos luego de realizar las operaciones que indican las flechas.

Figura No. 8 Diagrama de mariposa



Una vez se han separado los valores, se procede a realizar la FFT compleja sobre estos valores. Se aplica una FFT compleja a los valores pares, y una FFT compleja a los valores impares, tal como lo especifica el lema de Danielson-Lanczos:

Figura No. 9 Separación de pares e impares



Como se puede ver del lema de Danielson-Lanczos, se aplica por separado la DFT a la serie de números pares y a la serie de números impares, ya que según el lema, la DFT de un arreglo de tamaño N puede ser rescrita como la suma de dos DFT, cada una de largo N/2.

Recuérdese que

$$e^{\pm j\phi} = \cos(\phi) \pm jsen(\phi)$$

por lo que deben encontrarse los valores de los senos y cosenos. Estos se calculan por medio de un algoritmo de recursión, lo que es posible gracias a que sus argumentos forman una secuencia lineal. Así:

$$\cos(\theta + \delta) = \cos(\theta) - [\alpha \cos\theta + \beta \sin\theta]$$

$$\sin(\theta + \delta) = \sin(\theta) - [\alpha \sin\theta - \beta \cos\theta]$$

donde α y β corresponden a los coeficientes descritos por

$$\alpha \equiv 2\sin^2\left(\frac{\delta}{2}\right) \text{ y } \beta \equiv \sin\delta$$

donde $\alpha = alfa$, $\beta = beta$, $\delta = delta$.

Al hacer la analogía de estas recurrencias en el código de abajo, se observa que:

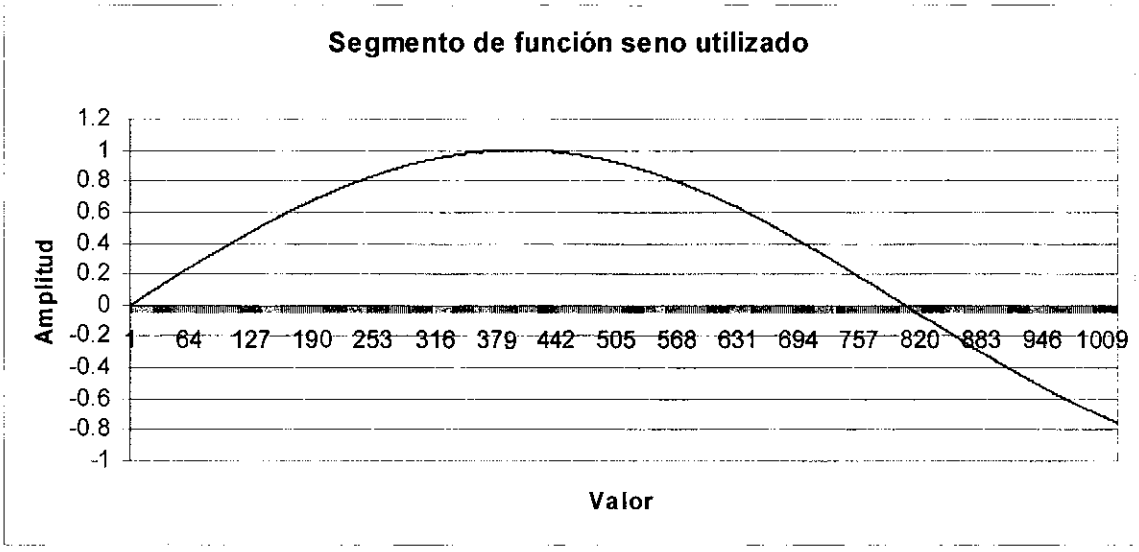
$$wr = \cos(\theta + \delta) = \cos(\theta) - [\alpha \cos\theta + \beta \sin\theta]$$
$$wi = \sin(\theta + \delta) = \sin(\theta) - [\alpha \sin\theta - \beta \cos\theta]$$
$$wpr = \alpha$$
$$wpi = \beta$$

Para poder simular una función de seno en el microcontrolador, se utilizan tablas. La razón por la que se utilizan tablas y no un método de aproximación numérica es debido a la

rapidez de un microcontrolador en solamente buscar los valores en una tabla, en lugar de realizar complicadas operaciones aritméticas que toman varios ciclos de reloj en ser ejecutadas.

El segmento de la función seno utilizada es la siguiente:

Figura No. 10 Segmento del seno utilizado

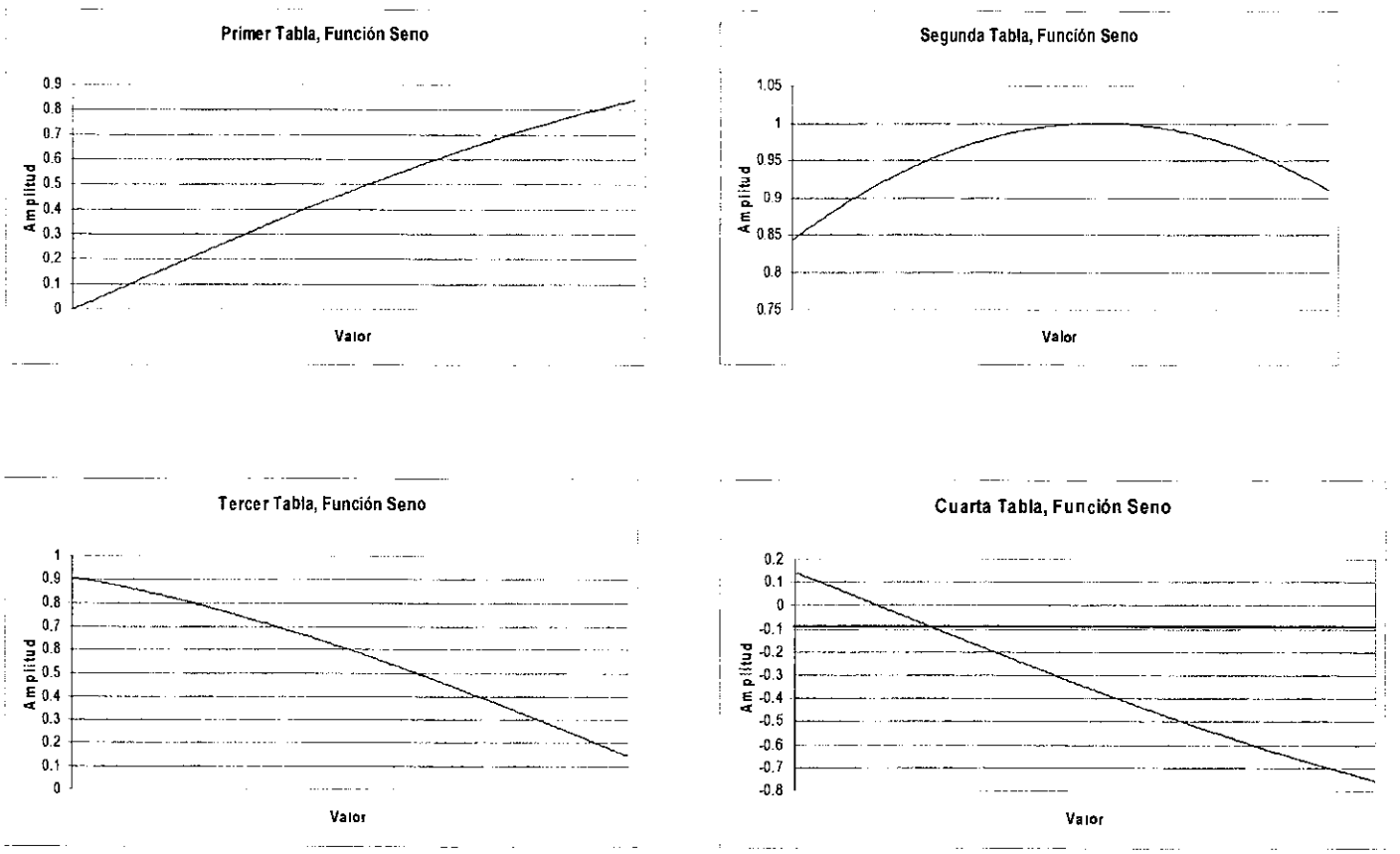


No es necesario incluir la función completa del seno, ya que el máximo valor que necesitará el seno es a PI (∏). Los siguientes valores a los que será necesario calcularles el seno tendrán un valor menor a éste. Esto nos ahorra espacio en el microcontrolador. Para lograr una mayor resolución con los valores del seno, se separa la función arriba descrita en cuatro partes. Cada gráfica abajo mostrada describe una parte del seno. La primera tabla describe el seno en los valores que abarcan de 0 a 0.99. La segunda tabla describe los valores que van de 1 a 1.99, la tercera describe los valores que van de 2 a 2.99 y la cuarta describe valores que van de 3 a 3.99.

Cada rango contiene 256 valores, sumando un total de 1024 valores para describir toda la función del seno a ser utilizada. Estos rangos se ubican en tablas diferentes, donde cada

valor utiliza dos posiciones de memoria (16 bits) para acoplarse al sistema de punto fijo utilizado de S-7-8.

Figura No.11 Separación en cuatro partes de la función seno



Luego de tener por separado las transformadas de los pares y de los impares, es necesario separar ambas transformadas para luego recombinarlas de la manera correcta. Para poder separar estas transformada, se utiliza la propiedad de la simetría de la transformada.

Si la información de entrada a la transformada es real, los componentes de la transformada discreta de Fourier satisfacen:

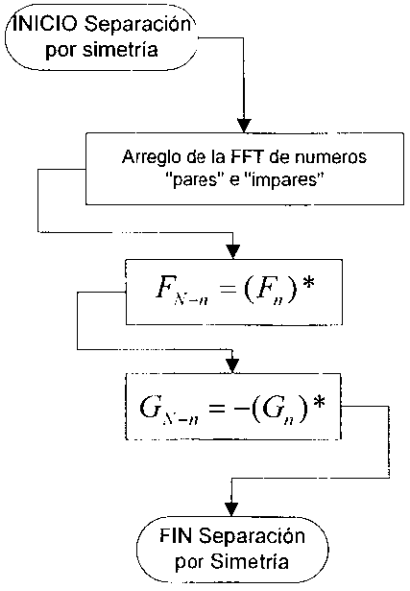
$$F_{N-n} = (F_n)^*$$

donde * denota al complejo conjugado. De la misma manera, la transformada discreta de Fourier de una entrada imaginaria tiene la simetría opuesta:

$$G_{N-n} = -(G_n)^*$$

Esto se observa en el siguiente diagrama de flujo:

Figura No. 12 Separación por simetría



La salida de la rutina anterior se puede expresar en forma matemática de la siguiente manera:

$$H_k = F_k^{\,Pares} + jF_k^{\,\mathrm{Im}\,pares}$$

donde se sabe del lema de Danielson-Lanczos y la DFT que

$$F_k^{\,Pares} = \sum_{n=0}^{N/2-1} f_{2n} e^{2\pi jkn/(N/2)}$$

$$F_k^{\text{Im pares}} = \sum_{n=0}^{N/2-1} f_{2n+1} e^{2\pi jkn/(N/2)}$$

Donde k va de 0 a N/2-1. Para mostrar la transformada F_n del arreglo original de datos, se pueden ordenar las ecuaciones anteriores para mostrar que :

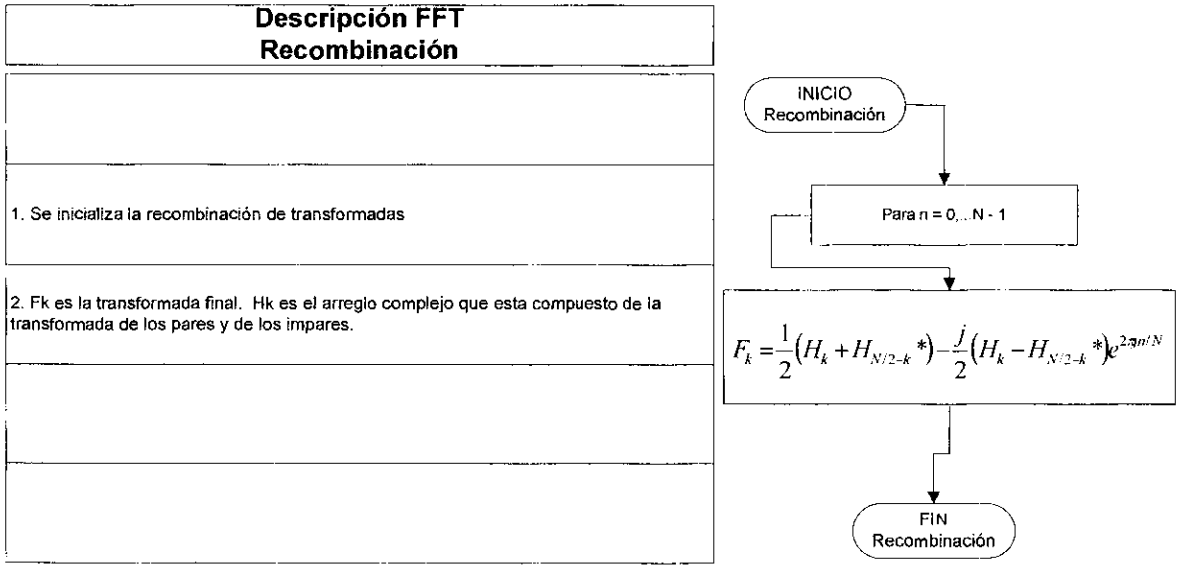
$$F_k = F_k^{Pares} + e^{2\pi jn/N} F_k^{\text{Im pares}} \quad n = 0,1,2,...,N-1$$

Si se expresa esto en términos de la transformada H_n de los datos reales (que se hacen pasar por complejos), el resultado es:

$$F_k = \frac{1}{2}(H_k + H_{N/2-k}^*) - \frac{j}{2}(H_k - H_{N/2-k}^*) e^{2\pi jn/N}$$

Debido a que $F_{N-k}^* = F_k$, no se necesita guardar el espectro entero (simetría de la DFT). El lado positivo de las frecuencias es suficiente y puede ser guardado en el mismo arreglo en el que estaban guardados los datos originales. Sin embargo, se necesitan los valores H_k, k = 0,1,2,...N/2, y la rutina anterior nos da valores hasta N/2-1. Debido a la simetría, se puede tomar el valor de $H_{k/2} = H_0$.

Figura No. 13 Recombinación de transformadas



Los resultados obtenidos son enviados a la computadora personal en un formato de 16 bits llamado S-7-8. El primer bit indica el signo del número, los siguientes siete bits indican el valor de la parte entera del número, y los últimos ocho bits indican los decimales del número.

El tiempo promedio de ejecución de la rutina en el microcontrolador, tomando en cuenta la recepción y envío de datos por medio de la interfase serial a 115200 bps es de aproximadamente 3.5 segundos.

Nota sobre el Código del Microcontrolador:

El código en Assembler es una traducción del código en C++. Para poder hacer el traslado de lenguajes, se utilizó una implementación de rutinas de punto fijo hechas por Robert Lacoste. Dado que las rutinas proporcionadas por Robert Lacoste están diseñadas para manejar solamente 256 muestras, fue necesario adaptarlas para acomodar el procesamiento de un mayor número de muestras, específicamente 4096 muestras.

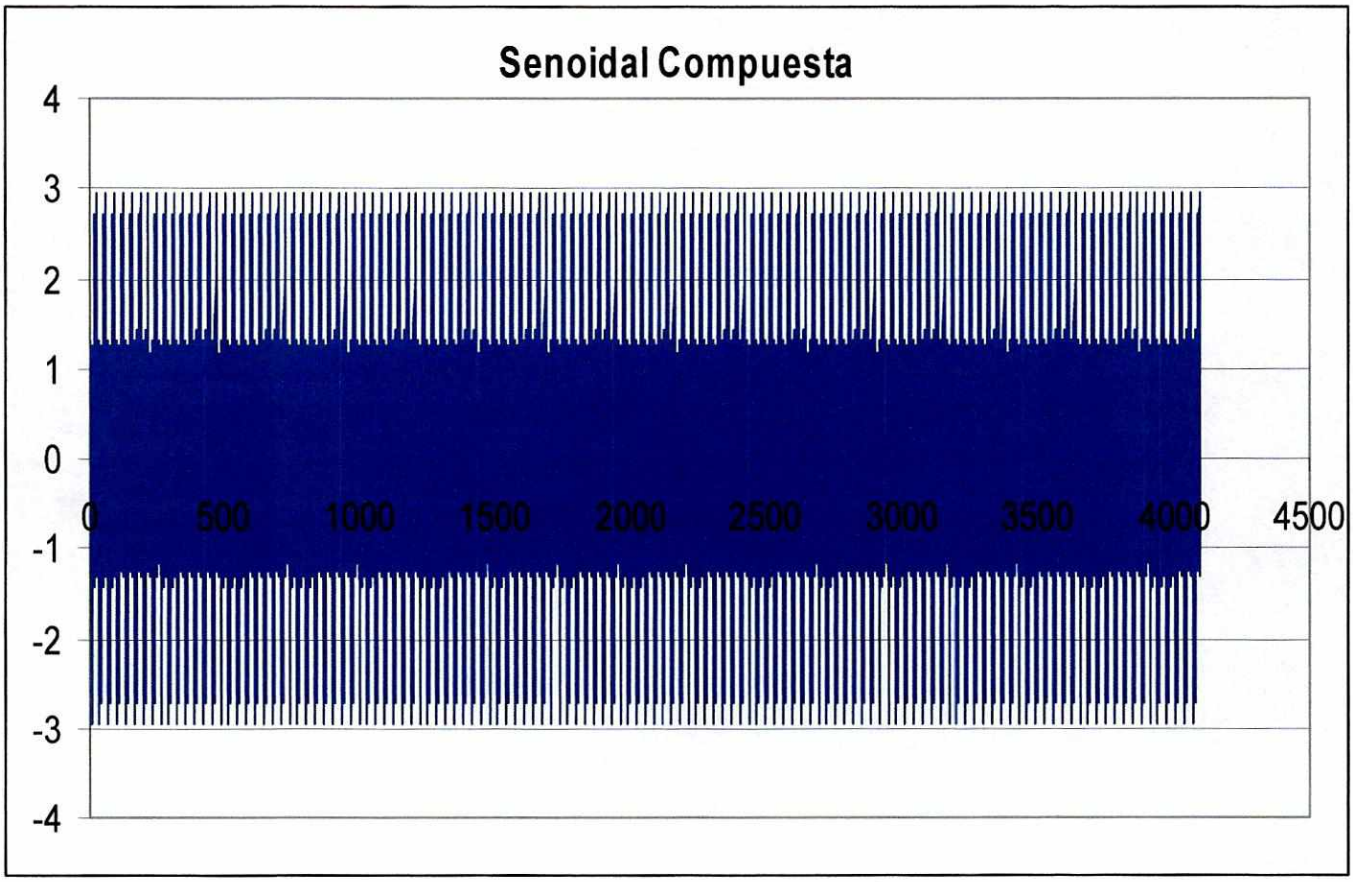
El PIC se mantiene en un ciclo infinito, esperando a que un interrupto del puerto serial lo saque del ciclo para poder cargar los datos en la RAM externa. Una vez tiene todos los datos guardados en RAM, empieza la rutina de la FFT. La primera parte realiza la FFT compleja, y la segunda parte encuentra la verdadera FFT a partir de las dos transformadas que nos da el código anterior. Estos valores de la FFT sobrescriben los valores originales de las muestras.

El código implementado, así como el diagrama del circuito se incluye en la sección de anexos, debidamente comentado para facilidad de lectura y entendimiento.

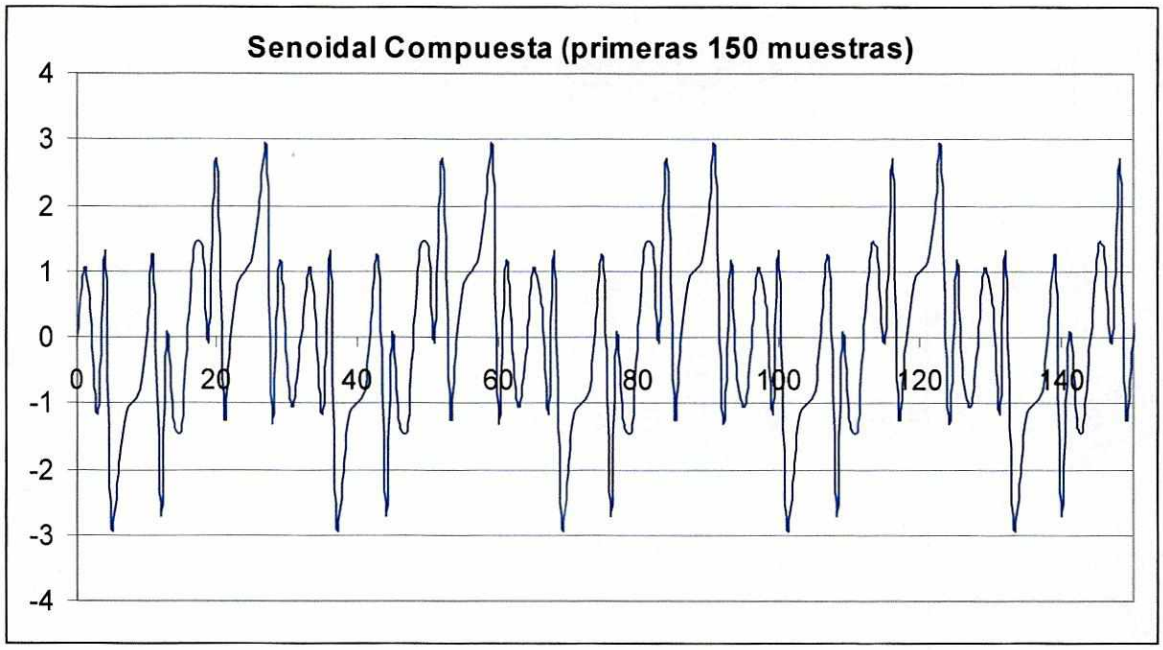
VI. ANÁLISIS Y PRESENTACIÓN DE RESULTADOS

Se comienza por analizar los resultados obtenidos mediante la aplicación de la herramienta de cálculo de FFT a una señal construida. Esta señal está constituida por cuatro señales senoidales de amplitud unitaria y con frecuencias de 200, 300, 1000 y 2000 Hz respectivamente. Se utilizó una frecuencia de muestreo de 4kHz, escogida de acuerdo a los principios del teorema de muestreo, obteniendo exactamente 4096 muestras. La señal enviada al microcontrolador se muestra en la gráfica No.1.

Gráfica No.1 (a)

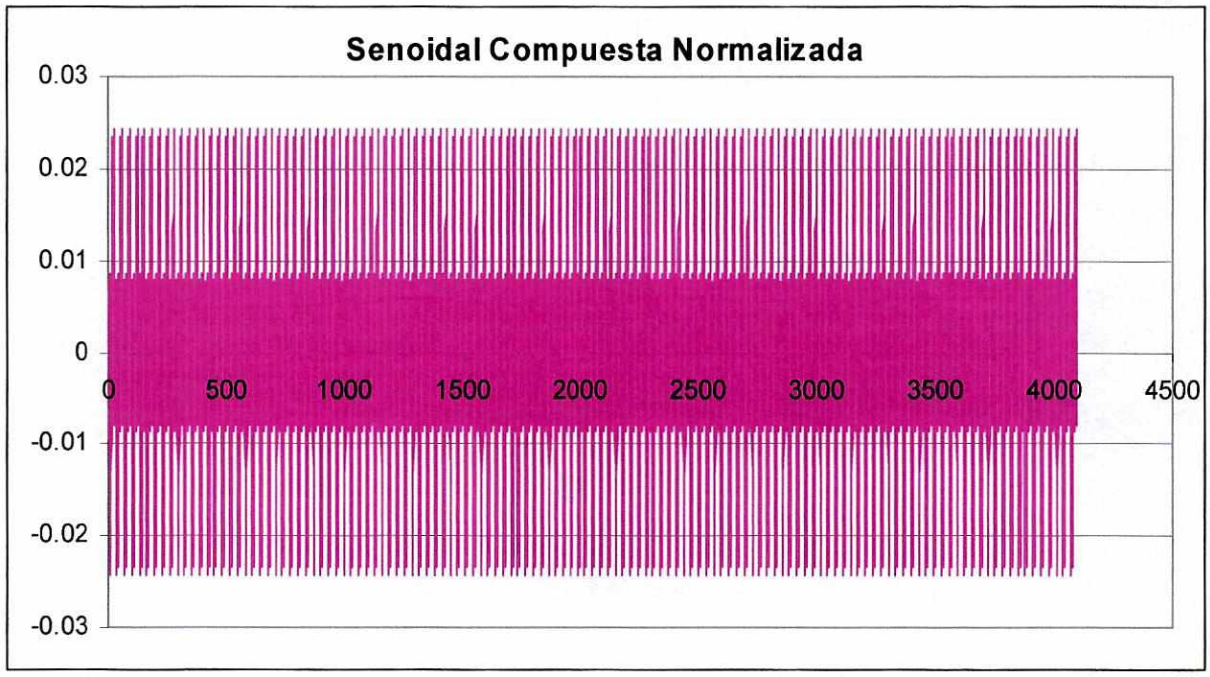


Gráfica No.1 (b)

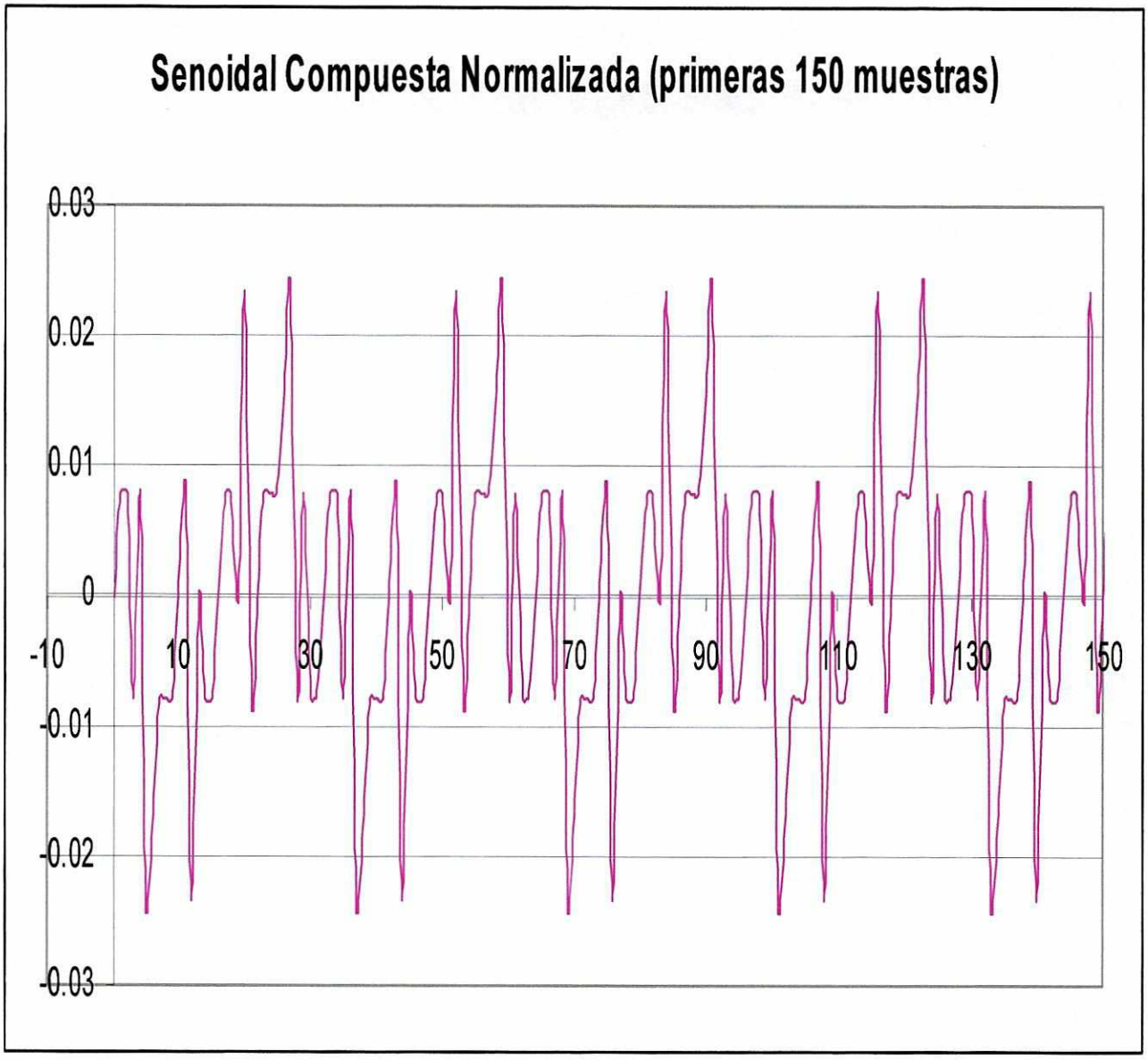


Estas muestras son sometidas al proceso de normalización descrito con anterioridad. El resultado se muestra en la siguiente gráfica:

Gráfica No.2 (a)

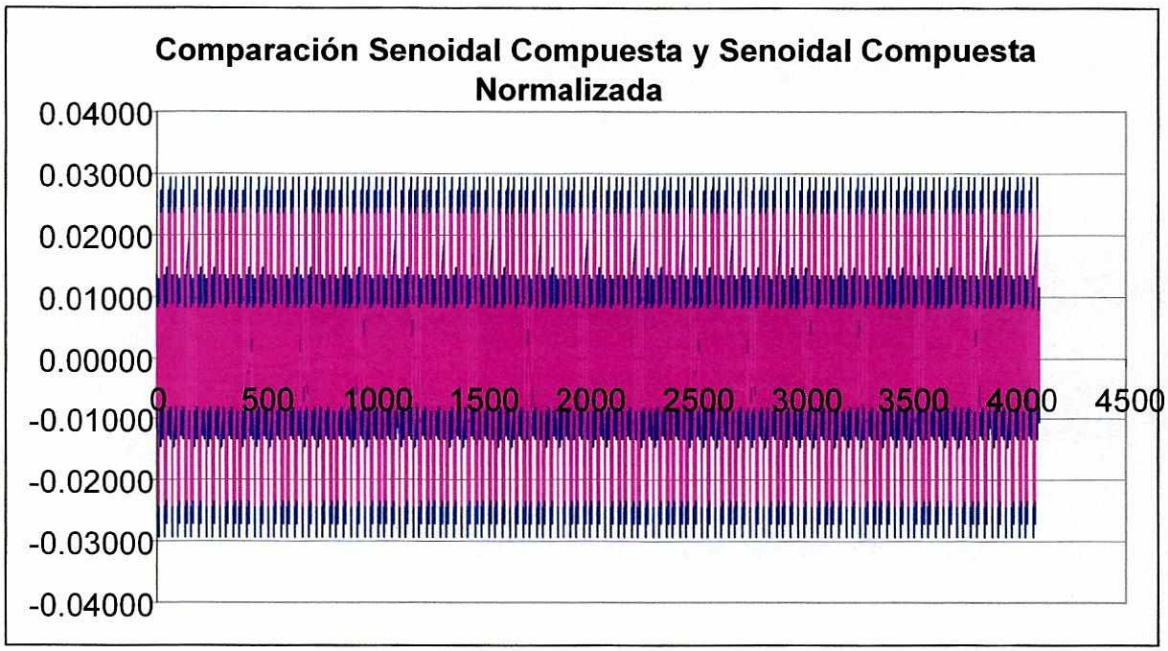


Gráfica No.2 (b)

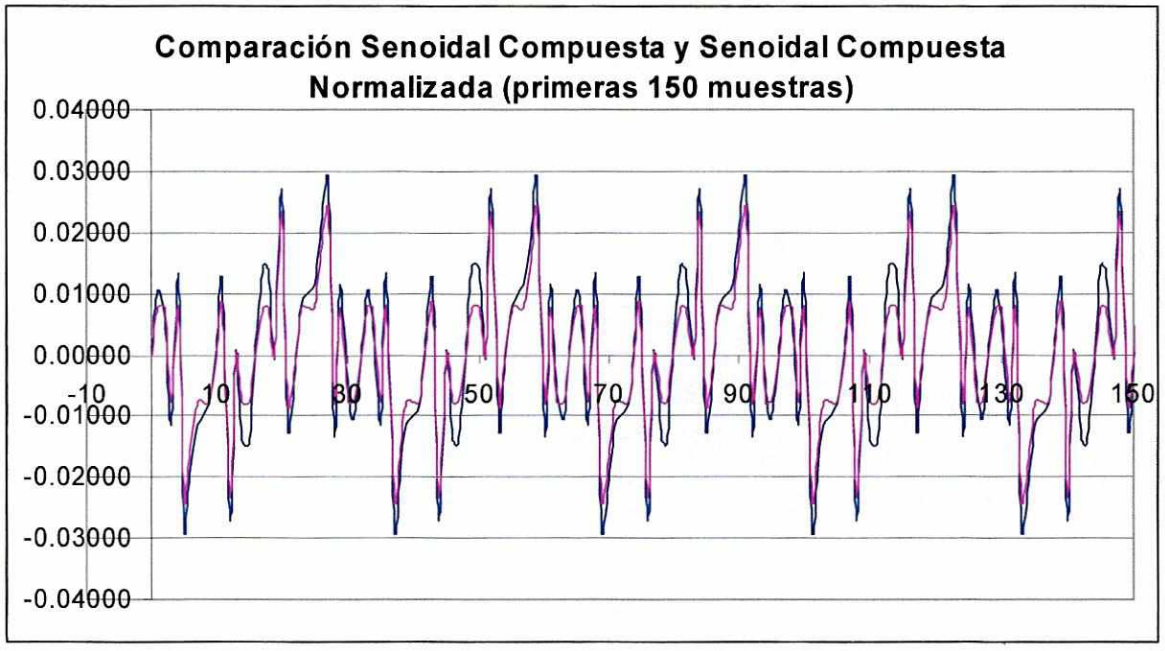


Las siguientes gráficas muestran una comparación entre la función senoidal y la función senoidal compuesta normalizada por el microcontrolador (comparación entre gráficas completas y en acercamiento). Nótese que la variación entre ambas no es únicamente de amplitud, sino que se presentan variaciones en la forma de la gráfica. Estas variaciones ocasionan la aparición de componentes frecuenciales errados, como se discute en la presentación de las gráficas de espectros frecuenciales.

Gráfica No. 2 (c)

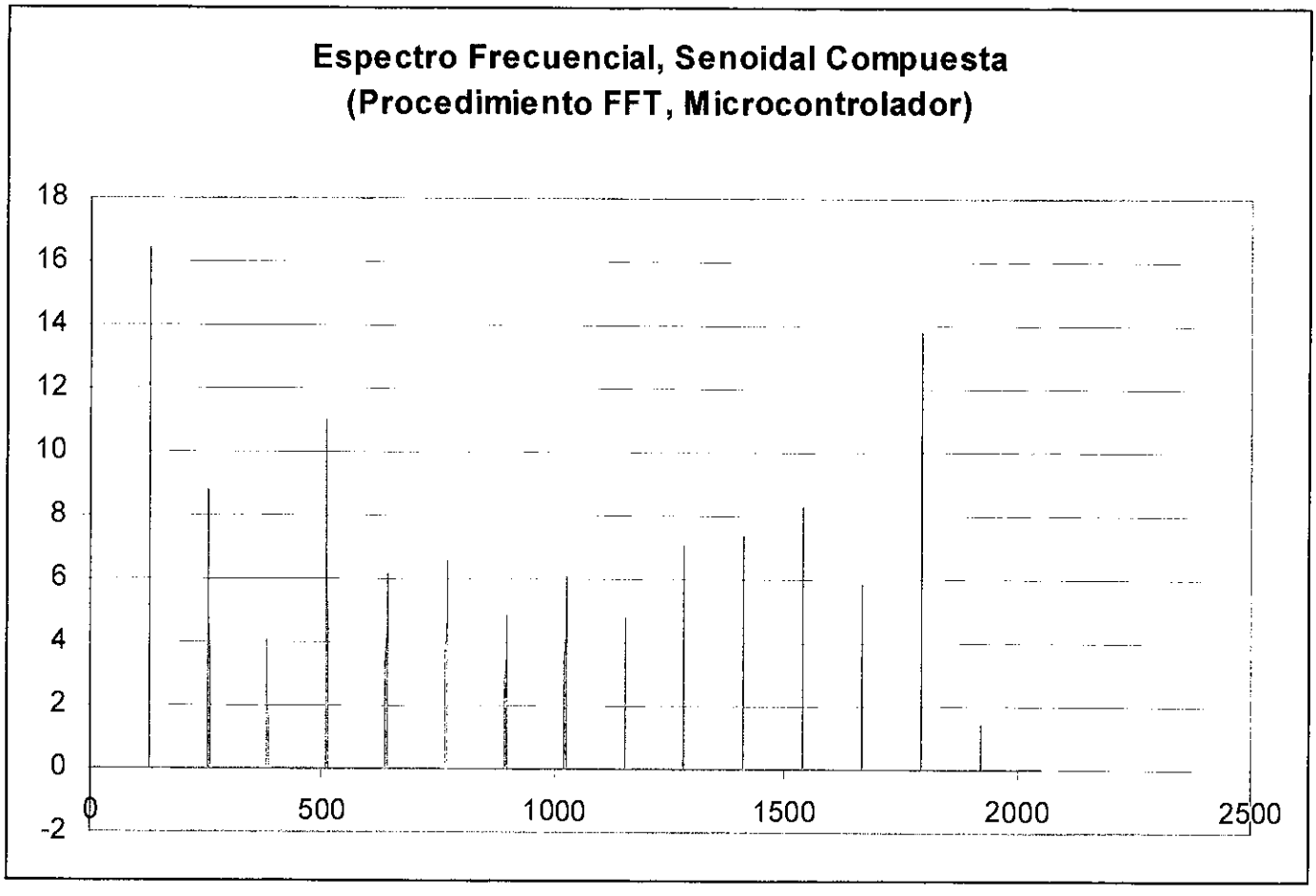


Gráfica No. 2 (d)



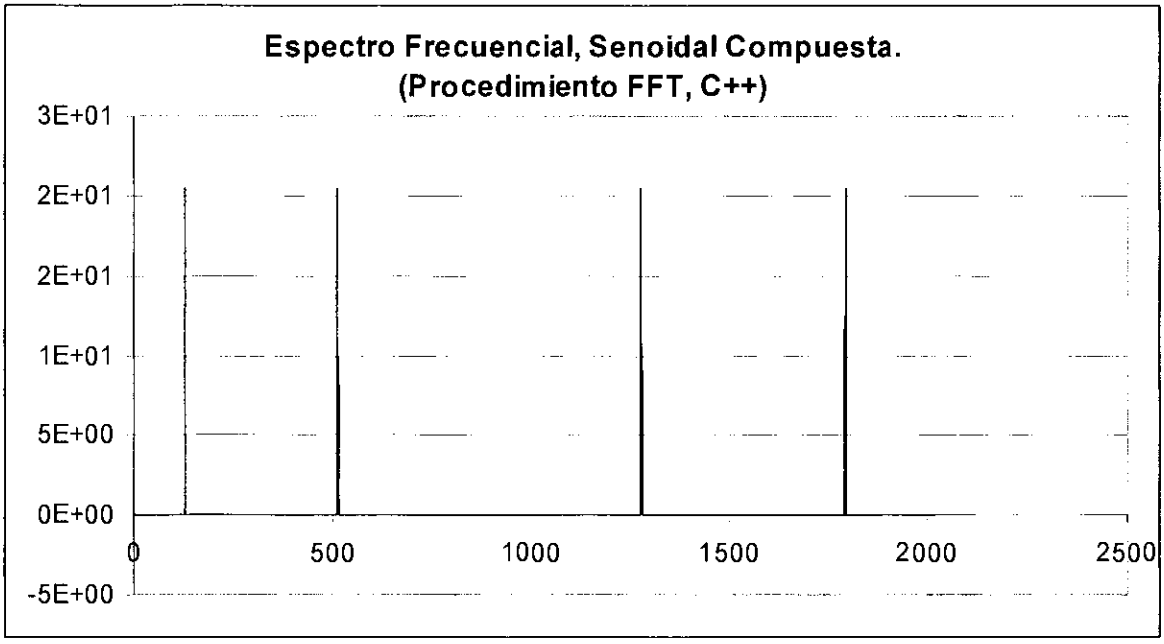
Una vez normalizados los datos, se aplica el algoritmo de transformada rápida de Fourier implementado en el microcontrolador. El espectro de amplitudes correspondiente se muestra en la siguiente gráfica.

Gráfica No.3



Para verificar el funcionamiento de la herramienta, se compararon sus resultados con una rutina equivalente desarrollada en el lenguaje de alto nivel C++, con la cual se obtuvieron los resultados mostrados en la gráfica siguiente.

Gráfica No.4



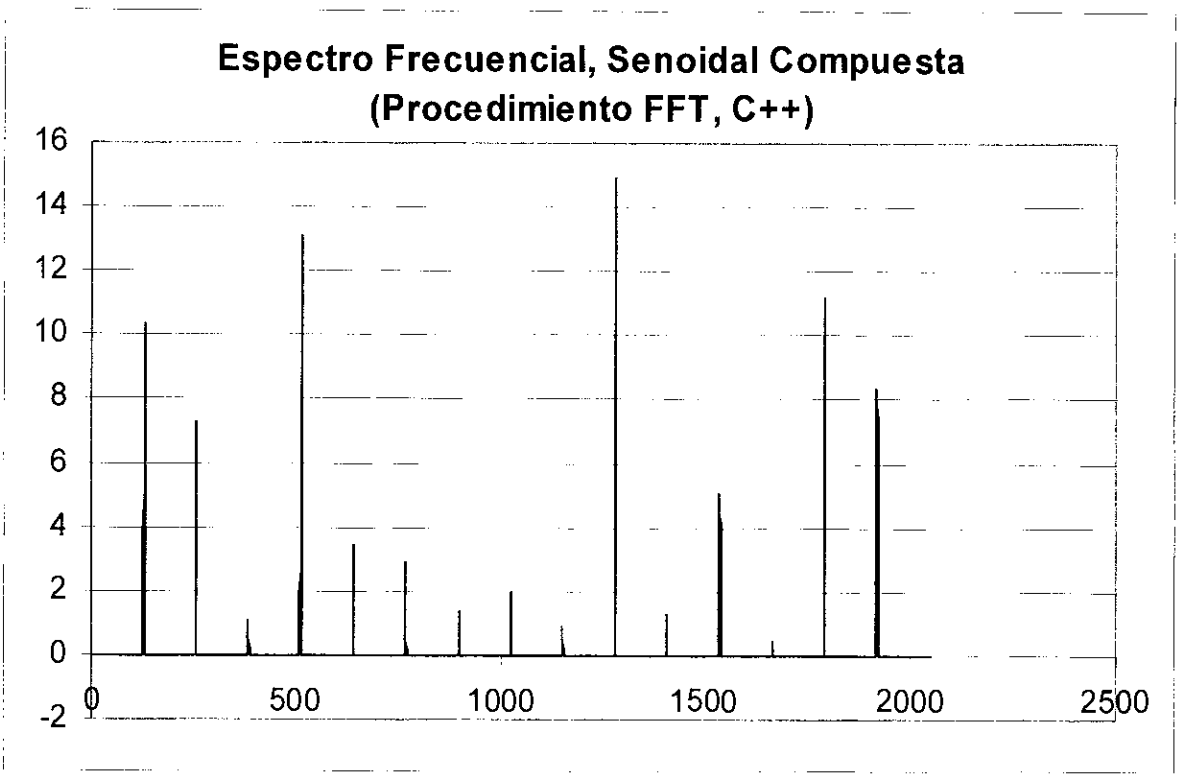
Puede observase, de la comparación entre la gráfica 3 y 4, que los resultados obtenidos de la herramienta implementada en el PIC muestran los mismos componentes frecuenciales que los mostrados en la gráfica No.4. Sin embargo, aparecen también componentes frecuenciales de menor amplitud. La diferencia espectral entre ambas gráficas se atribuye a dos factores principales:

1. La rutina de normalización, la cual, como puede observase en la gráfica 2, no preserva exactamente la forma de la señal original, debido a que la frecuencia de muestreo es exactamente el doble de la máxima frecuencia presente.

2. Los errores de aproximación generados al aplicar el algoritmo de transformada rápida de Fourier a muestras con el formato de punto fijo (restricción del número de decimales significativos).

Al analizar los resultados obtenidos en C++, pero utilizando como señal de entrada la función senoidal ya normalizada, se obtiene el siguiente resultado:

Gráfica No.5

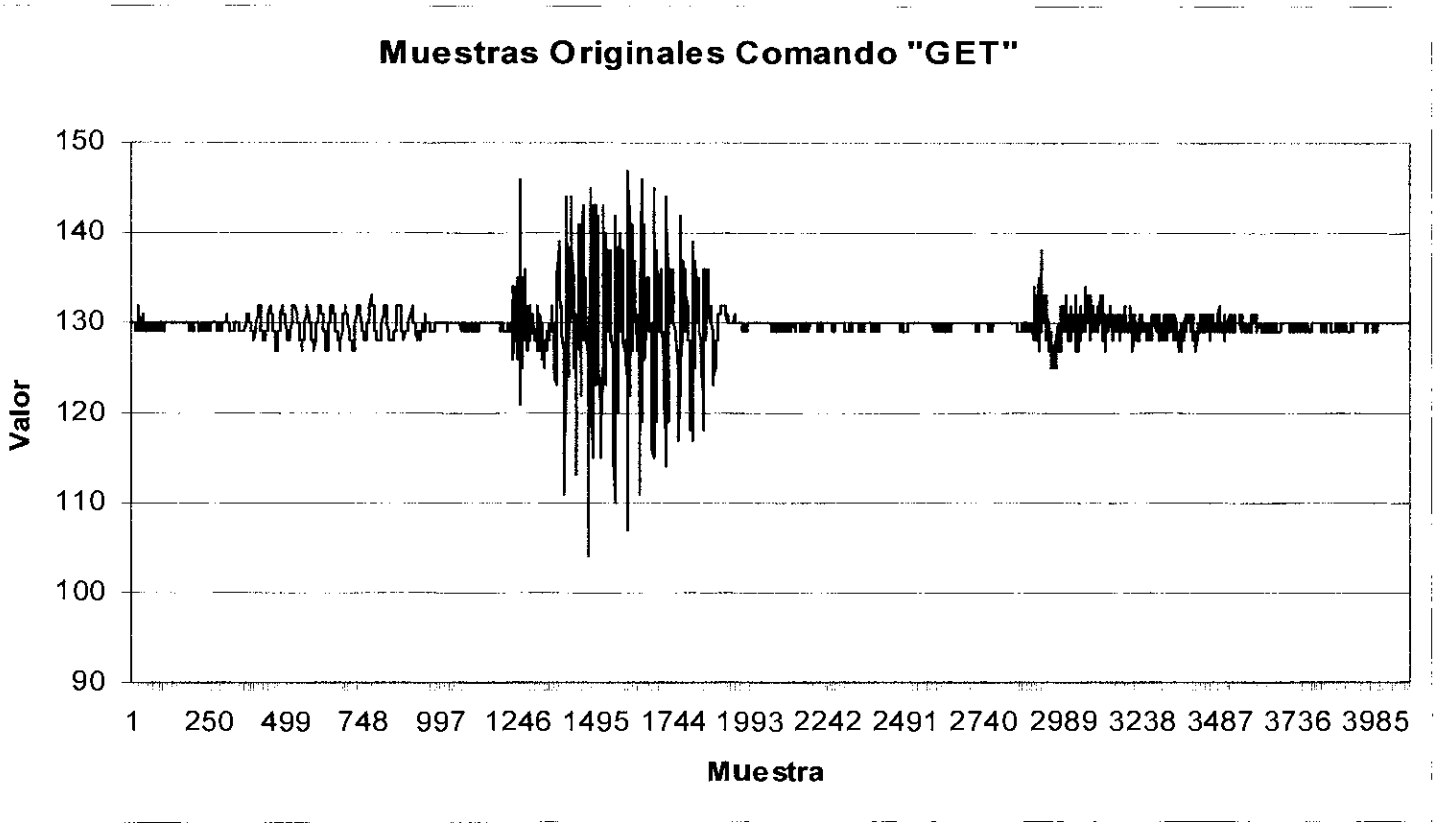


Nótese que la gráfica No.4 y la gráfica No.5 coinciden en cuanto al número de picos mostrados, con variaciones leves únicamente en las amplitudes de algunos de los componentes frecuenciales. De la comparación de estas gráficas se evidencia que el funcionamiento del algoritmo de la transformada rápida de Fourier implementado en el microcontrolador es comparable con el implementado en el lenguaje de alto nivel C++, y que las variaciones entre ambos se deben a errores de aproximación, ya que el procedimiento en C++ utiliza representación de punto flotante para cada número decimal, y no el formato de punto fijo implementado en el microcontrolador.

A continuación, se presentan gráficas comparativas de un comando de voz, capturado con la tarjeta de sonido y procesado tanto por el microcontrolador como por la rutina en software desarrollada en C++. Dadas las observaciones anteriores respecto a la normalización de los datos y la representación de punto fijo, las gráficas corresponden a la aplicación de los procedimientos a la señal de audio ya normalizada.

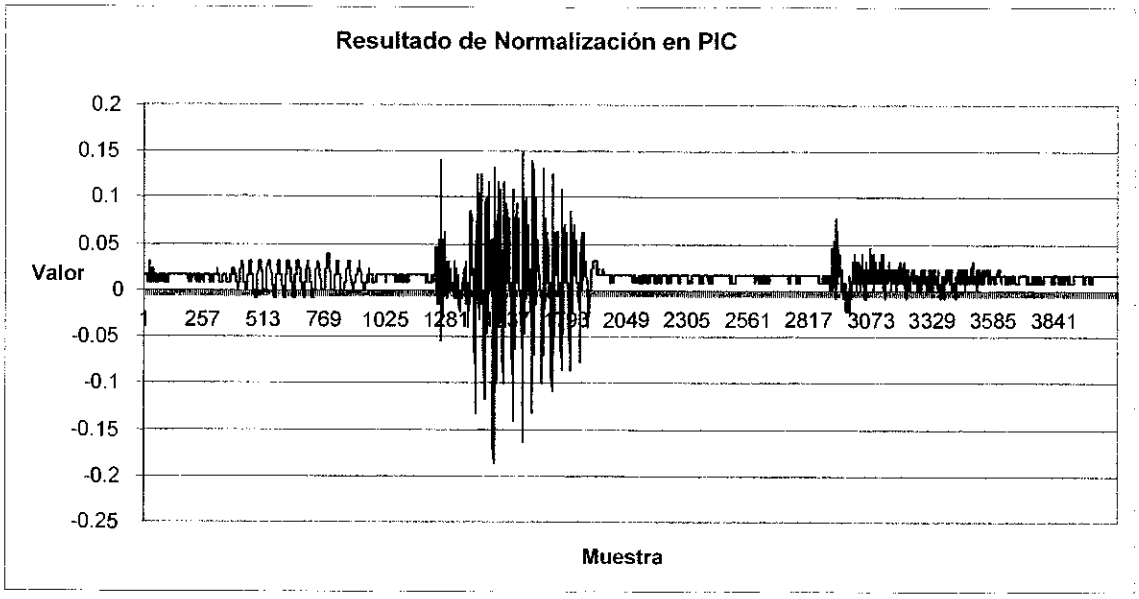
En la gráfica No. 6 se pueden observar las muestras originales del comando "GET" recabadas por la tarjeta de sonido. Estas son las muestras enviadas al microcontrolador, que luego se le aplica la normalización. La escala de valor (eje y) en la gráfica No. 6 corresponden al valor asignado por el convertidor análogo-digital de la tarjeta de sonido.

Gráfica No. 6



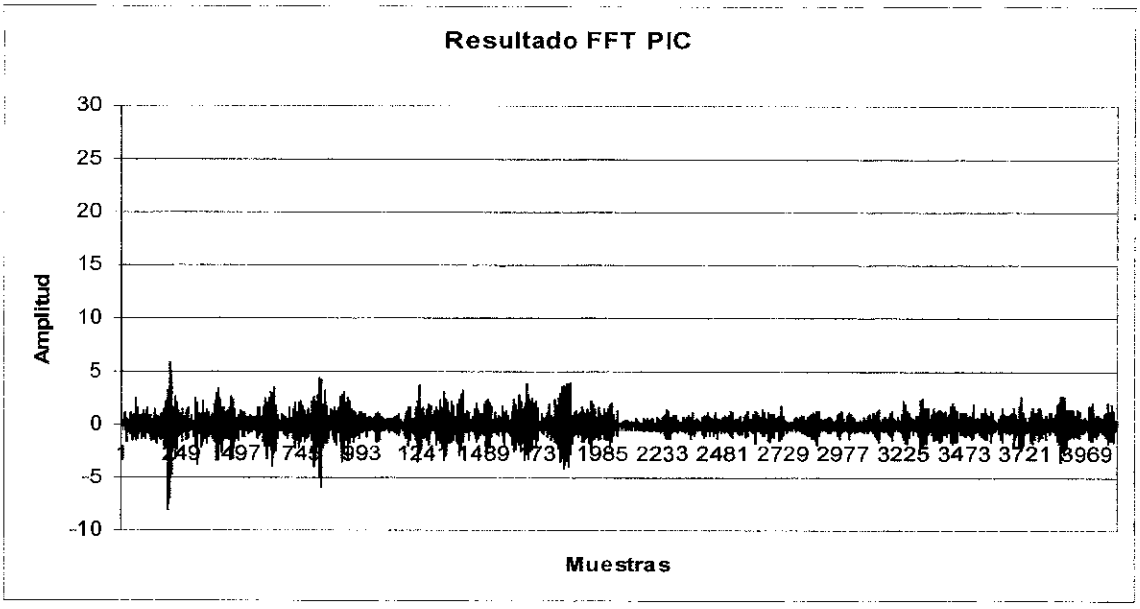
El resultado de la normalización realizada en el microcontrolador se puede apreciar en la gráfica No. 7. Se puede observar que la forma de la señal original, así como de la señal normalizada es parecida, con excepción de las escalas de amplitúd, ya que la señal normalizada esta en un rango mucho menor que la señal original.

Gráfica No. 7



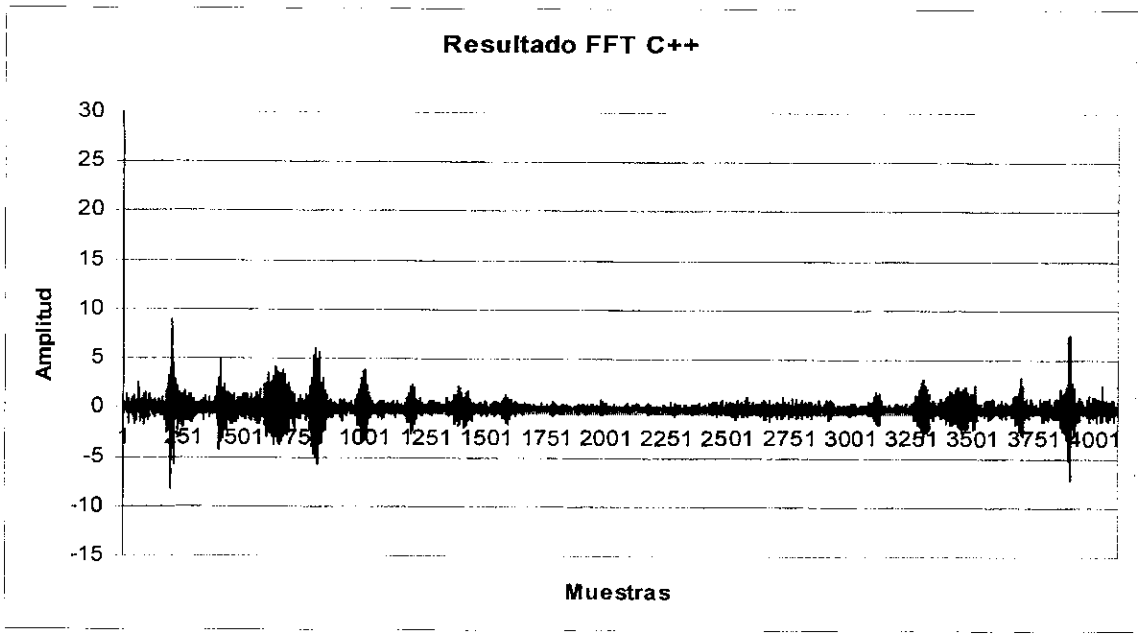
Luego se realizo el procedimiento de la FFT en el microcontrolador, pero sin el reordenamiento de los datos. Este resultado se muestra a continuación en la gráfica No.8:

Gráfica No. 8



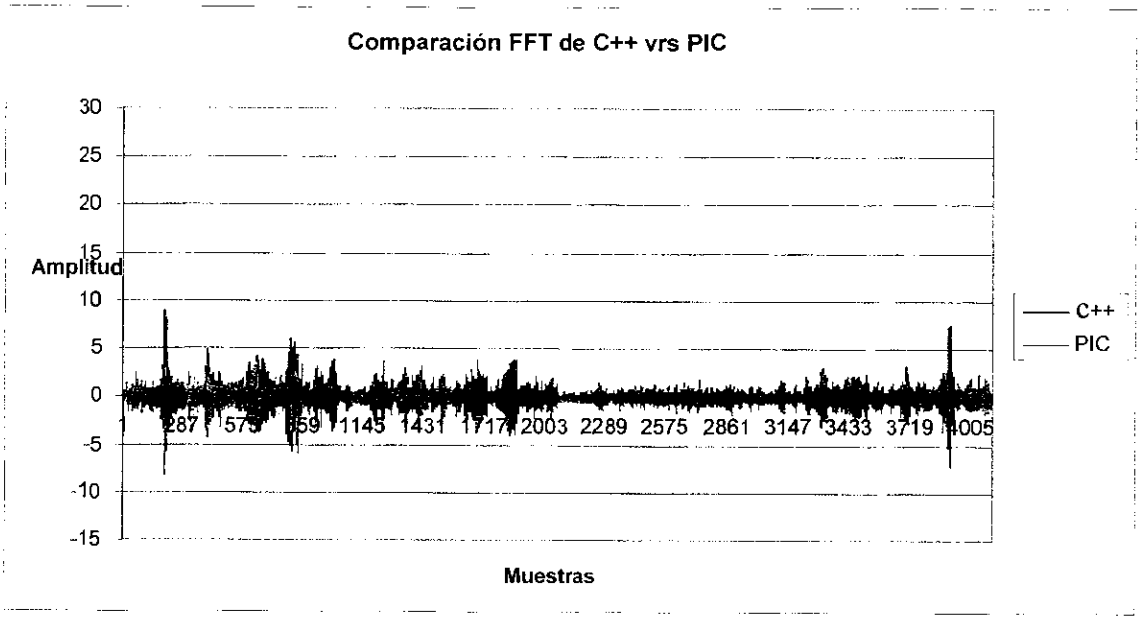
Para poder comparar resultados, se realizo la FFT en C++. Este resultado se observa en la gráfica No. 9.

Gráfica No. 9



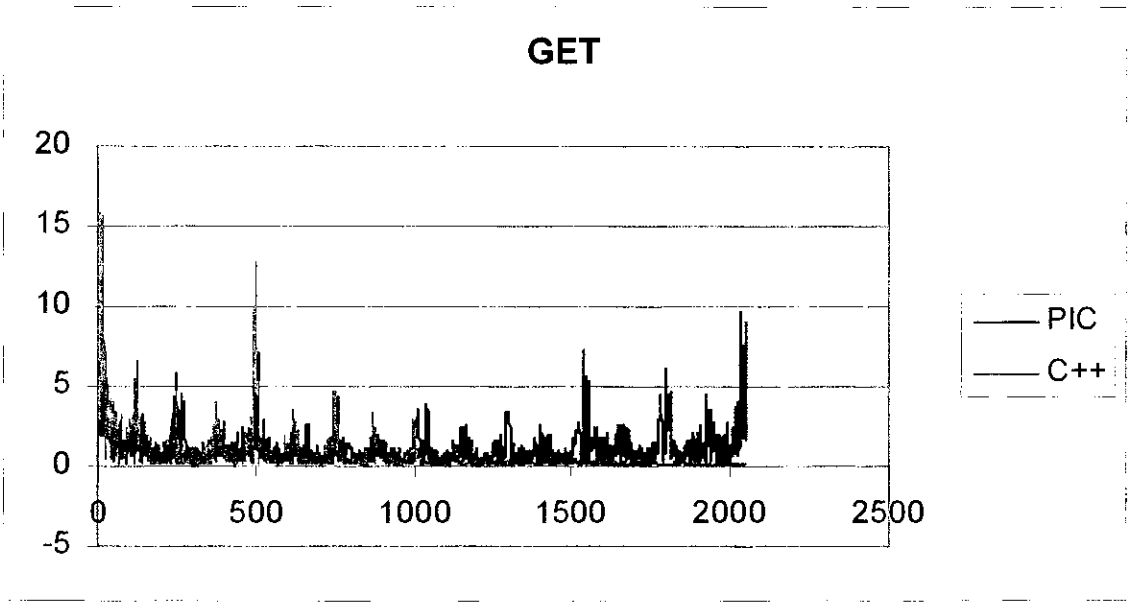
Al comparar ambos resultados, se observa que contienen los mismos componentes, pero con diferentes amplitudes y en algunos casos corridos. Esto se le atribuye a la falta de resolución para describir los números a cuenta del formato de punto fijo escogido.

Gráfica No. 10



El resultado final de todo el procedimiento de la FFT en el PIC así como en C++ se puede observar en la gráfica No. 11. Nótese que el resultado es muy parecido en la primera mitad de la gráfica. A partir de la mitad, el resultado del microcontrolador es significativamente diferente al procedimiento en C++. Debido al tipo de transformada que se utilizó, el espectro no se repite a partir de la mitad, por lo que los picos finales que no aparecen en la transformada de C++ pero si lo hacen en la transformada del PIC, se atribuyen a errores de aproximación.

Gráfica No. 11



VII. CONCLUSIONES

1. Es posible realizar un algoritmo de transformada rápida de Fourier en un microcontrolador.

2. Las necesidades de direccionamiento de la memoria externa utilizada reducen el número de puertos disponibles en el microcontrolador, lo que no permite la implementación de una interfase de comunicación con la PC diferente a la transmisión serial.

3. La utilización del formato de punto fijo propuesto, en conjunto con la rutina de normalización de la muestra restringen el número de bits asignados a la parte decimal del número a 8, limitando la resolución a un nivel insuficiente para la fiel representación de los resultados de la transformada.

4. El cálculo de senos en el microcontrolador utilizando tablas induce errores de aproximación que se reflejan en los espectros frecuenciales.

5. La velocidad de ejecución de la rutina en el microcontrolador es menor que la velocidad de ejecución de una rutina de software en una computadora personal (Pentium II, 266 MHz).

6. Para efectos de reconocimiento de voz, es preferible realizar la transformada rápida de Fourier en *software* ya que las computadoras personales hoy en día tienen una alta capacidad de procesamiento que rebasan las capacidades del micrcocontrolador.

7. Si el resultado de la transformada de Fourier no está basada en una aplicación de tiempo real o sensible al tiempo, y requiere una gran portabilidad, la solución por microcontrolador es preferible frente a una computadora personal.

8. Si se desea aumentar la velocidad de procesamiento, es preferible utilizar un número menor de muestras variando al mismo tiempo la cantidad de bits utilizados para representar los decimales y enteros, aumentando la cantidad de bits para representar los decimales.

9. En lugar de utilizar un microcontrolador para realizar la transformada de Fourier, es preferible utilizar un DSP (*digital signal processor*) ya que estos están diseñados para realizar estas funciones.

10. Utilizar el microcontrolador como un dispositivo de apoyo a una computadora personal no es eficiente debido a la perdida de tiempo en mandar y recibir información hacia el microcontrolador, así como la capacidad y velocidad de procesamiento de un microcontrolador con bus de 8 bits.

VIII. RECOMENDACIONES

Dado el impacto de los errores de aproximación debidos tanto al formato de punto fijo adoptado y a la aproximación de los valores de seno por medio de tablas, se sugiere diseñar el procedimiento adaptando una convención diferente de punto fijo, que de ser posible, aumenten las cifras significativas en cada número.

Se recomienda adapatar el mismo procedimiento, pero considerando otros dispositivos de memoria externa para el almacenamiento de los datos, que no requieran del uso excesivo de puertos para el direccionamiento; siempre y cuando el funcionamiento de estos dispositivos no altere significativamente el tiempo de ejecución del procedimiento.

IX. BIBLIOGRAFÍA

1. Chávez, J.J. 1994. *Elaboración de Proyectos de Investigación.* 2ª. Edición. XL Publicaciones. Guatemala. 75 págs.
2. Deller, J; J.H. Hansen, J. Proakis. 2000. *Discrete Time Processing of Speech Signals*. IEEE Press. E.E.U.U. 908 págs.
3. Fallside, F; W. Woods. 1985. *Computer Speech Processing.* Prentice Hall. E.E.U.U. 485 págs.
4. Lacoste, R. *Mathematical Functions for the PIC microcontroller* .
5. Lyons, R. 1997. *Understanding Digital Signal Processing.* Addison Wesley. E.E.U.U. 517 págs.
6. Press, W; S. Teukosky, W. Vetterling, B. Flannery. 1999. *Numerical Recipes in C.* Cambridge University Press. E.E.U.U. 994 págs.
7. Soliman, S; M. Srinath. 1998. *Continuous and Discrete Signals and Systems*. 2a. Edición. Prentice Hall. E.E.U.U. 525 págs.

X. APÉNDICES

APÉNDICE A

Código de Assembler para PIC18C452

```
;=============================================================================
;Pedro Viscovich
;Carnet No. 96095
;Tesis

;                              PROGRAMA PRINCIPAL DE FFT
;=============================================================================

;-----------------------------------------------------------------------------
; Configuración
;-----------------------------------------------------------------------------

                    PROCESSOR  18C452
                    RADIX      HEX
                    ERRORLEVEL 0
                    ERRORLEVEL -302


#include   "P18C452.INC"

;-----------------------------------------------------------------------------
; Vectores dc Interruptos
;-----------------------------------------------------------------------------

reset_vector        org H'0000'
                    goto start

serial_int  org H'0008'
                    goto serial_in

start_appl  org H'0050'


;-----------------------------------------------------------------------------
; Includes para el programa
;-----------------------------------------------------------------------------

#include   "vars.inc"           ; Declaramos variables
#include   "fixed.inc"          ; Rutinas de punto fijo
#include   "fft.inc"            ; FFT
#include   "irq.inc"            ; Rutina de Interrupt
#include   "serial.inc"         ; Rutina Scrial para mandar datos

;-----------------------------------------------------------------------------
; Inicio
;-----------------------------------------------------------------------------

start               nop
                    BANKSEL num_sample

                    call initialize                 ; inicializamos puertos y variables
                    call init_serial        ; rutina de inicializacion serial

mainloop  nop

;Rutina de Control de entrada Serial

;--> Hacer la FFT
```

```
                call initialize
                clrf num_sample,1
                clrf num_sample+1,1
                bcf TXSTA,TRMT,0
                clrf RCREG,0
                BANKSEL num_sample
                clrf ra+1,1
                call PORTDout
                nop
                bsf INTCON,GIE,0
                nop
                call DISABLE
prueba          nop
                btfss num_sample,5,1 ;Ciclo de espera de entrada de datos
                goto prueba                    ;por el puerto Serial
                nop
                bcf INTCON,GIE,0
                nop
                clrf num_sample,1
                clrf num_sample+1,1
                bcf PORTE,2,0

                call DISABLE
                bsf PORTE,1,0
                call realfft            ; FFT
                bcf PORTE,1,0

                bcf PORTC,0,0
                bsf PORTE,2,0
                call start_send
                bcf PORTE,2,0
                bcf PORTC,0,0
                nop
                goto mainloop


;-------------------------------------------------------------------------
; Inicializacion de Variables y Puertos
;-------------------------------------------------------------------------

initialize   nop

;--> Variables

                BANKSEL num_sample
                clrf num_sample,1
                clrf num_sample+1,1


;--> Puertos
                bcf T0CON,5,0
                movlw B'00000000'   ; Puerto RA salida
                movwf TRISA,0
                movlw B'00000110'
                movwf ADCON1,0
                clrf PORTA,0

                bcf INTCON2,7,0
                movlw B'00000000'   ; Puerto RB salida
                movwf TRISB,0
                clrf PORTB,0

                clrf PORTC,0
                clrf LATC,0
                movlw B'10000000'   ;Configuración Tx y Rx
                movwf TRISC,0

                movlw B'00000000'   ; Puerto RB salida
                movwf TRISE,0
                clrf PORTE,0
```

```
                call PORTDout

                goto endinit

PORTDin
                movlw B'11111111'  ; Puerto RD entrada
                movwf TRISD,0
                clrf PORTD,0
                return

PORTDout
                movlw B'00000000'  ; Puerto RD salida
                movwf TRISD,0
                clrf PORTD,0
                return

endinit

;----->Interruptos

                movlw B'00000000'  ;Configuramos interrupto
                movwf INTCON,0              ;en RCREG de puerto Serial
                movwf INTCON2,0
                movwf INTCON3,0
                movwf PIR1,0
                movwf PIE2,0
                movwf RCON,0
                movwf IPR1,0
                movwf IPR2,0
                movlw B'00100000'
                movwf PIE1,0
                bsf INTCON,PEIE,0
                return

;-------------------------------------------------------------------------
;Control de Entrada y Salida de RAM
;-------------------------------------------------------------------------

DISABLE
                movlw B'00000011'
                movwf PORTC,0
                return

READ
                movlw B'00000010'
                movwf PORTC,0
                return

;Despues de un READ, hacer DISABLE

WRITE
                movlw B'00000000'
                movwf PORTC,0
                return
;-------------------------------------------------------------------------
;Inicializacion Serial
;-------------------------------------------------------------------------
init_serial

                bsf     RCSTA,SPEN,0
                bsf     RCSTA,CREN,0
                bcf     TXSTA,SYNC,0
                bcf     TXSTA,TX9,0
                bsf     TXSTA,BRGH,0
                movlw   .16                         ;32.000 MHz
                movwf   SPBRG,0
                bsf     TXSTA,TXEN,0
                return
```

```
                END

;=================================================================
;                FIN
;=================================================================

;=================================================================
;
;                        serial_in
;
;
;Rutina para Responder al Interrupto de RCREG activado
;al llenarse el bufer de entrada del puerto serial.
;=================================================================

Normalizar
        ; Normalizamos valor que recibimos del Serial
        bcf STATUS,C,0
        rrcf ra,F,1         ; division por 4
        rrcf ra+1,F,1
        bcf STATUS,C,0
        rrcf ra,F,1
        rrcf ra+1,F,1
        movlw D'32'                 ;ra - H'2000'
        cpfslt ra,1
        goto restar
        bsf STATUS,C,0
        movlw H'20'
        subfwb ra,F,1
        bsf ra,7,1
        return
restar  subwf ra,F,1                ; Esta ahora de -1.0 a +1.0
        return

serial_in   ;Velocidad de transmision = 115200 bps
        ;Tiempo en pasar todos los datos por serial
        ;mas guardar las 4096 muestras en RAM
        ;es 0.35seg.
        bsf PORTE,0,0
        movff RCREG,ra
                                    ; Guardamos valor en la posicion a la que apunta num_sample
        clrf ra+1,1
        call Normalizar             ; Normalizamos a nuestra escala de punto flotante
        btfss ra,7,1
        goto posa
        goto nega

rotate  bcf STATUS,C,0
        rrcf ra,F,1
        rrcf ra+1,F,1
        bcf STATUS,C,0
        rrcf ra,F,1
        rrcf ra+1,F,1
        bcf STATUS,C,0
        rrcf ra,F,1
        rrcf ra+1,F,1
        bcf STATUS,C,0
        rrcf ra,F,1
        rrcf ra+1,F,1
        bcf STATUS,C,0
        rrcf ra,F,1
        rrcf ra+1,F,1
        return

posa    call rotate
        bcf ra,7,1
        goto cont

nega    bcf ra,7,1
        call rotate
```

```
                bsf ra,7,1
                goto cont

cont            movff num_sample,PORTA          ;Escogemos en que posicion de memoria
                movff num_sample+1,PORTB        ;queremos guardar los datos
                movff ra,PORTD                  ;Movemos los datos a RAM
                call WRITE
                call DISABLE                    ;Deshabilitamos Write en RAM (READ)
                infsnz num_sample+1,F,1         ;Preparamos para la siguiente muestra
                incf num_sample,F,1
                movff num_sample,PORTA          ;Escogemos en que posicion de memoria
                movff num_sample+1,PORTB        ;queremos guardar los datos
                movff ra+1,PORTD
                call WRITE                      ;Habilitamos Write en RAM (WRITE_)
                call DISABLE                    ;Deshabilitamos Write en RAM (READ)
                infsnz num_sample+1,F,1         ;Preparamos para la siguiente muestra
                incf num_sample,F,1
                bcf PORTE,0,0
                retfie

;=================================================================
;                       FIN SERIAL
;=================================================================
```

```
;=================================================================
;
;                              FIXED.INC
;
;Rutina de operacion de Punto Fijo de Robert Lacoste
;=================================================================


;-----------------------------------------------------------------
; m_lda :   copiar un registro cualquiera (2 bytes) a ra
;-----------------------------------------------------------------

m_lda           macro inreg
                movff inreg,WREG
                movwf ra,1
                movff inreg+1,WREG
                movwf ra+1,1
                endm

;-----------------------------------------------------------------
; m_ldai :  obtener la informacion a la que apunta indexreg y guardarla en ra
;-----------------------------------------------------------------

m_ldai          macro indexregh,indexregl
                movff indexregh,tmpi
                movff indexregl,tmpi+1
                bcf STATUS,C            ;x2
                rlcf tmpi+1,F,1
                rlcf tmpi,F,1
                movff tmpi,PORTA
                movff tmpi+1,PORTB
                call PORTDin            ;PORTD as input
                call READ
                movff PORTD,ra
                call DISABLE
                incf tmpi+1,F,1
                movff tmpi+1, PORTB     ;$+14
                call READ
                movff PORTD,ra+1
                call DISABLE
                endm

;-----------------------------------------------------------------
; m_ldb :   copiar un registro cualquiera (2 bytes) en rb
;-----------------------------------------------------------------

m_ldb           macro inreg
                movff inreg,WREG
                movwf rb,1
                movff inreg+1,WREG
                movwf rb+1,1
                endm

;-----------------------------------------------------------------
; m_ldbi :  obtener la informacion a la que apunta indexreg y guardarla en rb
;-----------------------------------------------------------------

m_ldbi          macro indexregh, indexregl
                movff indexregh,tmpi
                movff indexregl,tmpi+1
                bcf STATUS,C            ;x2
                rlcf tmpi+1,F,1
                rlcf tmpi,F,1
```

```
                movff tmpi,PORTA
                movff tmpi+1,PORTB
                call PORTDin                    ;PORTD as input
                call READ
                movff PORTD,rb
                call DISABLE
                incf tmpi+1,F,1
                movff tmpi+1, PORTB
                call READ
                movff PORTD,rb+1
                call DISABLE
                endm

;-----------------------------------------------------------------
; m_lddiv :copiar un registro (2 bytes) en registro rdiv
;-----------------------------------------------------------------

m_lddiv         macro inregh, inregl
                movff inregh,WREG
                movwf rdiv,1
                movff inregl,WREG
                movwf rdiv+1,1
                endm

;-----------------------------------------------------------------
; m_sta :   guardar el valor de A en un registro general outreg (2 bytes)
;-----------------------------------------------------------------

m_sta           macro outreg
                movff ra,WREG
                movwf outreg,1
                movff ra+1,WREG
                movwf outreg+1,1
                endm

;-----------------------------------------------------------------
; m_stai :  guardar el valor de ra en la direccion dememoria a la que apunta indexreg
;-----------------------------------------------------------------

m_stai          macro indexh, indexl
                movff indexl, tmpi+1
                movff indexh,tmpi
                bcf STATUS,C
                rlcf tmpi+1,F,1                 ;x2
                rlcf tmpi,F,1
                movff tmpi,PORTA
                movff tmpi+1,PORTB
                call PORTDout
                movff ra,PORTD
                movff ra,PORTD
                call WRITE
                call DISABLE
                incf tmpi+1,F,1
                movff tmpi+1, PORTB
                movff ra+1,PORTD
                movff ra+1,PORTD
                call WRITE
                call DISABLE
                endm

;-----------------------------------------------------------------
; m_ldaconst :      cargar un valor inmediato a ra
;-----------------------------------------------------------------

m_ldaconst      macro immvalue
                movlw high immvalue
                movwf ra,1
                movlw low  immvalue
                movwf ra+1,1
                endm
```

```
;-----------------------------------------------------------------
; m_mvba :          copiar rb a rA
;-----------------------------------------------------------------

m_mvba            macro
                  movff rb,WREG
                  movwf ra,1
                  movff rb+1,WREG
                  movwf ra+1,1
                  endm

;-----------------------------------------------------------------
; m_mvab :          copiar ra a rb
;-----------------------------------------------------------------

m_mvab            macro
                  movff ra,WREG
                  movwf rb,1
                  movff ra+1,WREG
                  movwf rb+1,1
                  endm

;-----------------------------------------------------------------
; m_add :  sumar rb a ra (A=A+B)
;-----------------------------------------------------------------

m_add
                  btfss ra,7,1
                  goto pos          ;Es ra>0?
                  goto neg          ;Es ra<0?

neg
                  bcf ra,7,1        ;Limpieza de signo (sabemos que ra<0)
                  btfss rb,7,1
                  goto pos3         ;Es rb>0?
                  goto neg3         ;Es rb<0?

neg3
                  bcf ra,7,1        ;CASO ra<0, rb<0
                  bcf rb,7,1        ;Limpieza de signo (sabemos que rb<0)
                  movff rb+1,WREG           ;Suma
                  addwf ra+1,1,1            ;de ra con
                  movff rb,WREG             ;rb, seteando
                  addwfc ra,1,1             ;el bit de
                  bsf ra,7,1        ;signo.
                  goto endadd

pos3                                        ;CASO ra<0, rb>0
                  movff ra,WREG
                  cpfsgt rb,1
                  goto tsteq2       ;Si rb<=ra...
                  movff ra+1,WREG           ;Si rb>ra entonces
                  subwf rb+1,1,1            ;se restan
                  movff ra,WREG             ;los valores
                  subwfb rb,1,1             ;pero no se setea
                  movff rb,ra
                  movff rb+1,ra+1
                  goto endadd               ;el bit de signo.

tsteq2                                      ;CASO ra<0, rb<=ra
                  movff ra,WREG
                  cpfseq rb,1       ;ra=rb?
                  goto ragrb2               ;ra>rb
                  goto tstlow2              ;Si ra=rb, compare parte baja.

tstlow2                                     ;CASO ra<0, rb=ra; comparacion parte baja
                  movff ra+1,WREG
                  cpfsgt rb+1,1
                  goto tsteql2              ;si rb+1=ra+1, rb=ra!
```

```
                movff ra+1,WREG         ;Si rb+1>ra, entonces
                subwf rb+1,1,1          ;reste los valores sin
                movff ra,WREG           ;setear bit de signo, pues
                subwfb rb,1,1           ;rb>ra y rb>0.
                movff rb,ra
                movff rb+1,ra+1
                goto endadd

tsteql2                                 ;CASO rb=ra
                movff ra+1,WREG
                cpfseq rb+1,1
                goto ragrb2
                clrf ra,1       ;El resultado es CERO, pues
                clrf ra+1,1     ;ra<0 y rb>0, pero en magnitud son iguales!
                goto endadd

ragrb2                                  ;CASO ra<0, ra>rb,
                movff rb+1,WREG         ;Se suman
                subwf ra+1,1,1          ;los valores,
                movff rb,WREG           ;seteando bit de
                subwfb ra,1,1           ;signo (ra<0).
                bsf ra,7,1
                goto endadd
pos                                     ;CASO ra>0
                btfss rb,7,1
                goto pos2       ;rb>0...
                goto neg2       ;rb<0...

pos2                                    ;CASO ra>0, rb>0
                movff rb+1,WREG         ;Como ambos positivos,
                addwf ra+1,1,1          ;basta sumar
                movff rb,WREG           ;ambos registros.
                addwfc ra,1,1
                goto endadd

neg2                                    ;CASO ra>0, rb<0
                bcf rb,7,1
                movff ra,WREG
                cpfsgt rb,1
                goto tsteqh     ;rb<=ra
                movff ra+1,WREG         ;CASO ra>0, rb>ra
                subwf rb+1,1,1          ;En este caso,
                movff ra,WREG           ;se suman ambos registros
                subwfb rb,1,1           ;pero como rb>ra y rb<0,
                bsf ra,7,1      ;seteamos el bit de signo.
                goto endadd

tsteqh                                  ;CASO ra>0, rb<=ra
                movff ra,WREG
                cpfseq rb,1
                goto ragrb      ;rb<ra...
                goto testlow            ;rb=ra...

ragrb                                   ;CASO ra>0, rb<ra
                movff rb+1,WREG         ;Se suman ambos registros,
                subwf ra+1,1,1          ;no se setea el bit de signo.
                movff rb,WREG
                subwfb ra,1,1
                goto endadd

testlow                                 ;CASO ra>0, rb=ra... prueba parte baja!
                movff ra+1,WREG
                cpfsgt rb+1,1
                goto testleq            ;rb+1<=ra+1;
                movff ra+1,WREG         ;CASO ra>0, rb+1>r+1a
                subwf rb+1,1,1          ;Se sabe que rb<0 en este caso, entonces
                movff ra,WREG           ;se suman los valores y se setea
                subwfb rb,1,1           ;el bit de signo.
                movff rb,ra
                movff rb+1,ra+1
```

```
                bsf ra,7,1
                goto endadd

testleq                                         ;CASO ra>0, rb+1<=ra+1
                movff ra+1,WREG
                cpfseq rb+1,1
                goto ragrb              ;rb+1<ra+1, sumc sin setear bit de signo.
                clrf ra,1               ;partes bajas tambien son iguales,
                clrf ra+1,1             ;resultado igual a cero!
                goto endadd

endadd
                return
;-------------------------------------------------------------------------
; m_sub :  restar rb a ra (A=A-B)
;-------------------------------------------------------------------------

m_sub
                btfss rb,7,1
                goto caso1              ;Si rb es positivo
                goto caso2              ;Si rb es negativo
caso2
                btfss ra,7,1
                goto posA               ;Si ra es positivo
                goto negA               ;Si ra es negativo

caso1           btfss ra,7,1
                goto posA2
                goto negA2
negA2
                bcf ra,7,1              ;Si rb es positivo y ra es Negativo
                bcf rb,7,1
                movff rb+1,WREG
                addwf ra+1,1,1
                rnovff rb,WREG
                addwfc ra,1,1
                bsf ra,7,1
                goto endsub

posA
                movff rb+1,WREG                 ; Si ra es Positivo y rb es Negativo, entonces
                addwf ra+1,1,1                  ;ra-(-rb)=ra+rb
                movff rb,WREG                   ; y guardamos el resultado en ra
                addwfc ra,1,1
                bcf ra,7,1
                goto endsub


;-------------------------------------------------------------------------
;
;                       CASO EN EL QUE RA Y RB SON POSTIVOS
;
;-------------------------------------------------------------------------
posA2
                movff ra,WREG
                cpfsgt rb,1
                goto pisrbeqra
                goto posrbgtposra

posrbgtposra
                movff ra+1,WREG
                subwf rb+1,1,1
                movff ra,WREG
                subwfb rb,1,1
                movff rb,ra
                movff rb+1,ra+1
                bsf ra,7,1
                goto endsub
```

```
pisrbeqra
                cpfseq rb,1
                goto posragtposrb
                movff ra+1,WREG
                cpfsgt rb+1,1
                goto pisrbleqral
                goto posrblgtposral

posragtposrb
                movff rb+1,WREG
                subwf ra+1,1,1
                movff rb,WREG
                subwfb ra,1,1
                goto endsub

pisrbleqral
                cpfseq rb+1,1
                goto posralgtposrbl
                clrf ra,1
                clrf ra+1,1
                goto endsub

posralgtposrbl
                movff rb+1,WREG
                subwf ra+1,1,1
                movff rb,WREG
                subwfb ra,1,1
                goto endsub

posrblgtposral
                movff ra+1,WREG
                subwf rb+1,1,1
                movff ra,WREG
                subwfb rb,1,1
                movff rb,ra
                movff rb+1,ra+1
                bsf ra,7,1
                goto endsub

;-------------------------------------------------------------------
;
;               FIN CASO RA Y RB SON POSITIVOS
;
;-------------------------------------------------------------------
;-------------------------------------------------------------------
;
;               CASO EN EL QUE RA Y RB SON NEGATIVOS
;
;-------------------------------------------------------------------
negA
                bcf rb,7,1          ;Si ra es Negativo y rb es Negativo, entonces
                bcf ra,7,1          ;ya sabemos que son negativos, y limpiamos el bit
                movff ra,WREG               ;de negativo (bit 7 MSB en ambas variables). Luego
                cpfsgt rb,1         ;Comparamos si rb>ra
                goto isrbeqra               ;Caso rb no es > que ra
                goto negrbgtnegra   ;Caso rb es > que ra

negrbgtnegra
                movff ra+1,WREG             ;Si -rb>-ra entonces hacemos la resta de
                subwf rb+1,1,1              ;rb-ra. Debido a que la formula es -ra-(-rb)
                movff ra,WREG               ;nos queda de la siguiente forma: -ra+rb, y como
                subwfb rb,1,1               ;rb es mayor, entonces el resultado es positivo
                movff rb,ra                 ;y lo guardamos en ra despues de hacer la resta.
                movff rb+1,ra+1
                goto endsub

isrbeqra
                cpfseq rb,1         ;Si -rb no es > que -ra, entonces comparamos si
                goto negragtnegrb   ;rb = ra. Si son iguales, entonces comparamos si
                movff ra+1,WREG             ;rb+1 > ra+1 (LSB).
```

```
                cpfsgt rb+1,1
                goto isrbleqral             ;Caso rb+1 no es > que ra+1
                goto negrblgtnegral  ;Caso rb+1 > ra+1

negragtnegrb
                movff rb+1,WREG             ;Caso en el que ra > rb.  Hacemos la siguiente
                subwf ra+1,1,1              ;opcracion:  -ra-(-rb) = -ra+rb = -ra
                movff rb,WREG
                subwfb ra,1,1
                bsf ra,7,1
                goto endsub

negrblgtnegral
                movff ra+1,WREG             ;Si -rb+1 > -ra+1 entonces -rb>-ra y por lo
                subwf rb+1,1,1              ;tanto la equacion queda de la siguiente forma:
                movff ra,WREG               ;-ra-(-rb) = -ra+rb, por lo que hacemos la resta
                subwfb rb,1,1               ;de rb-ra, y el resultado lo guardamos en ra.
                movff rb,ra
                movff rb+1,ra+1
                goto endsub

isrbleqral
                cpfseq rb+1,1
                goto negralgtnegrbl
                goto negrbeqnegra

negrbeqnegra
                clrf ra,1
                clrf ra+1,1
                goto endsub

negralgtnegrbl
                movff rb+1,WREG
                subwf ra+1,1,1
                movff rb,WREG
                subwfb ra,1,1
                bsf ra,7,1
                goto endsub

;-------------------------------------------------------------------------
;
;                       FIN CASO RA Y RB SON NEGATIVOS
;
;-------------------------------------------------------------------------
endsub
                return

;-------------------------------------------------------------------
; m_mult : multiplica ra a rb (A=A*B)
;-------------------------------------------------------------------

m_mult
                clrf WREG,0
                btfsc rb,7,1                ; probar si rb<0
                bsf WREG,7,0
                btfsc ra,7,1          ; probar si ra<0
                xorlw B'10000000'
                movff WREG,sign
                bcf rb,7,1
                bcf ra,7,1

                movff ra+1,WREG             ; bajo x bajo
                mulwf rb+1,1
                movff PRODH,mres1
                movff PRODL,mres0
                movff ra,WREG               ; alto x alto
                mulwf rb,1
                movff PRODH,mres3
                movff PRODL,mres2
                movff ra+1,WREG             ;bajo x alto
```

```
                mulwf rb,1
                movff PRODL,WREG
                addwf mres1,F,1
                movff PRODH,WREG
                addwfc mres2,F,1
                clrf WREG,0
                addwfc mres3,F,1
                movff ra,WREG                   ; alto x bajo
                mulwf rb+1,1
                movff PRODL,WREG
                addwf mres1,F,1
                movff PRODH,WREG
                addwfc mres2,F,1
                clrf WREG,0
                addwfc mres3,F,1

                movff mres2, ra                 ; guardar resultado en ra
                movff mres1, ra+1
                tstfsz sign,1
                bsf ra,7,1

                movlw B'01111111'               ;Rutina para evitar el problema
                andwf ra,W,1                    ;de -0.00000000
                tstfsz WREG,0                   ;Si ra y ra+1 son 0, entonces
                goto end_mult                   ;limpie el bit de signo.
                tstfsz ra+1,1
                goto end_mult
                bcf ra,7,1

end_mult  return

;-------------------------------------------------------------------------
; m_divi :  divide A con rdiv (A=A/DIV), rdiv es un entero sin signo
;;;-------------------------------------------------------------------------

#define         AARGB0 (ra)
#define         AARGB1 (ra+1)
#define         ACCB0  (ra)
#define         ACCB1  (ra+1)
#define    BARGB0 (rdiv)
#define         BARGB1 (rdiv+1)
#define         REMB0  (mres0)
#define         REMB1  (mres1)

#include   "div16.inc"                  ; Rutina de Microchip para division
                                        ; de 16 bits / 16 bits

m_divi          bcf msign,0,1
                btfss ra,7,1
                goto m_divdiv
                bsf msign,0,1

                comf ra,F,1             ; si A<0, A=-A
                comf ra+1,F,1
                movlw 1
                addwf ra+1,F,1
                movlw 0
                addwfc ra,F,1

m_divdiv  call FXD1616U                 ; hacer la division

                btfss msign,0,1         ; si A es <0
                goto end_mdivi
                comf ra,F,1             ; A=-A
                comf ra+1,F,1
                movlw 1
                addwf ra+1,F,1
                movlw 0
                addwfc ra,F,1
```

```
end_mdivi return

;-----------------------------------------------------------------
; m_div2 : divide A con 2 (A=A/2)
;-----------------------------------------------------------------

m_div2
                clrf sign,1
                btfsc ra,7,1            ; divide byte mas significativo
                bsf sign,7,1
                bcf ra,7,1              ; y menos significativo
                bcf STATUS,C,0
                rrcf ra,1,1
                rrcf ra+1,1,1
                btfsc sign,7,1
                bsf ra,7,1

                movlw B'01111111'  ;Rutina para evitar el problema
                andwf ra,W,1                ;de -0.00000000
                tstfsz WREG,0               ;Si ra y ra+1 son 0, entonces
                goto endm_div2              ;limpie el bit de signo.
                tstfsz ra+1,1
                goto endm_div2
                bcf ra,7,1
endm_div2return

;-----------------------------------------------------------------
; m_sin :   reemplaza ra con sen(A)
;
;-----------------------------------------------------------------
                radix dec
tblcero
                dw      0,      31,     63,     95,     127,    159,    191,    223
                dw      255,    287,    319,    351,    383,    415,    447,    479
                dw      511,    543,    575,    607,    639,    671,    703,    735
                dw      766,    798,    830,    862,    894,    926,    957,    989
                dw      1021,   1053,   1084,   1116,   1148,   1179,   1211,   1243
                dw      1274,   1306,   1337,   1369,   1401,   1432,   1464,   1495
                dw      1527,   1558,   1589,   1621,   1652,   1683,   1715,   1746
                dw      1777,   1808,   1840,   1871,   1902,   1933,   1964,   1995
                dw      2026,   2057,   2088,   2119,   2150,   2181,   2212,   2242
                dw      2273,   2304,   2335,   2365,   2396,   2427,   2457,   2488
                dw      2518,   2548,   2579,   2609,   2640,   2670,   2700,   2730
                dw      2760,   2790,   2821,   2851,   2881,   2910,   2940,   2970
                dw      3000,   3030,   3059,   3089,   3119,   3148,   3178,   3207
                dw      3237,   3266,   3295,   3325,   3354,   3383,   3412,   3441
                dw      3470,   3499,   3528,   3557,   3586,   3615,   3643,   3672
                dw      3700,   3729,   3757,   3786,   3814,   3842,   3871,   3899
                dw      3927,   3955,   3983,   4011,   4039,   4067,   4094,   4122
                dw      4150,   4177,   4205,   4232,   4260,   4287,   4314,   4341
                dw      4368,   4395,   4422,   4449,   4476,   4503,   4530,   4556
                dw      4583,   4609,   4636,   4662,   4688,   4714,   4741,   4767
                dw      4793,   4819,   4844,   4870,   4896,   4921,   4947,   4972
                dw      4998,   5023,   5048,   5074,   5099,   5124,   5149,   5173
                dw      5198,   5223,   5248,   5272,   5296,   5321,   5345,   5369
                dw      5393,   5418,   5441,   5465,   5489,   5513,   5536,   5560
                dw      5583,   5607,   5630,   5653,   5676,   5699,   5722,   5745
                dw      5768,   5791,   5813,   5836,   5858,   5881,   5903,   5925
                dw      5947,   5969,   5991,   6013,   6034,   6056,   6077,   6099
                dw      6120,   6141,   6162,   6183,   6204,   6225,   6246,   6267
                dw      6287,   6308,   6328,   6348,   6368,   6389,   6409,   6428
                dw      6448,   6468,   6487,   6507,   6526,   6546,   6565,   6584
                dw      6603,   6622,   6641,   6659,   6678,   6696,   6715,   6733
                dw      6751,   6769,   6787,   6805,   6823,   6840,   6858,   6875
        radix hex

                radix dec
tbluno
                dw      6893,   6910,   6927,   6944,   6961,   6978,   6995,   7011
```

```
        dw      7028,   7044,   7060,   7077,   7093,   7109,   7124,   7140
        dw      7156,   7171,   7187,   7202,   7217,   7232,   7247,   7262
        dw      7277,   7292,   7306,   7320,   7335,   7349,   7363,   7377
        dw      7391,   7405,   7418,   7432,   7445,   7458,   7472,   7485
        dw      7498,   7510,   7523,   7536,   7548,   7561,   7573,   7585
        dw      7597,   7609,   7621,   7632,   7644,   7655,   7667,   7678
        dw      7689,   7700,   7711,   7722,   7732,   7743,   7753,   7763
        dw      7774,   7784,   7794,   7803,   7813,   7823,   7832,   7841
        dw      7850,   7860,   7869,   7877,   7886,   7895,   7903,   7912
        dw      7920,   7928,   7936,   7944,   7951,   7959,   7967,   7974
        dw      7981,   7988,   7995,   8002,   8009,   8016,   8022,   8029
        dw      8035,   8041,   8047,   8053,   8059,   8065,   8070,   8076
        dw      8081,   8086,   8091,   8096,   8101,   8106,   8110,   8115
        dw      8119,   8123,   8127,   8131,   8135,   8139,   8142,   8146
        dw      8149,   8152,   8155,   8158,   8161,   8164,   8166,   8169
        dw      8171,   8173,   8175,   8177,   8179,   8181,   8182,   8184
        dw      8185,   8186,   8187,   8188,   8189,   8190,   8190,   8191
        dw      8191,   8191,   8191,   8191,   8191,   8191,   8191,   8190
        dw      8189,   8189,   8188,   8187,   8185,   8184,   8183,   8181
        dw      8179,   8178,   8176,   8174,   8172,   8169,   8167,   8164
        dw      8162,   8159,   8156,   8153,   8150,   8146,   8143,   8139
        dw      8136,   8132,   8128,   8124,   8120,   8116,   8111,   8107
        dw      8102,   8097,   8092,   8087,   8082,   8077,   8071,   8066
        dw      8060,   8055,   8049,   8043,   8037,   8030,   8024,   8017
        dw      8011,   8004,   7997,   7990,   7983,   7976,   7968,   7961
        dw      7953,   7946,   7938,   7930,   7922,   7914,   7905,   7897
        dw      7888,   7880,   7871,   7862,   7853,   7844,   7834,   7825
        dw      7815,   7806,   7796,   7786,   7776,   7766,   7756,   7745
        dw      7735,   7724,   7714,   7703,   7692,   7681,   7670,   7658
        dw      7647,   7635,   7624,   7612,   7600,   7588,   7576,   7564
        dw      7551,   7539,   7526,   7514,   7501,   7488,   7475,   7462
        radix hex

        radix dec

tbldos
        dw      7448,   7435,   7422,   7408,   7394,   7380,   7367,   7352
        dw      7338,   7324,   7310,   7295,   7281,   7266,   7251,   7236
        dw      7221,   7206,   7191,   7175,   7160,   7144,   7128,   7113
        dw      7097,   7081,   7064,   7048,   7032,   7015,   6999,   6982
        dw      6965,   6948,   6931,   6914,   6897,   6880,   6862,   6845
        dw      6827,   6809,   6792,   6774,   6756,   6738,   6719,   6701
        dw      6682,   6664,   6645,   6626,   6608,   6589,   6570,   6550
        dw      6531,   6512,   6492,   6473,   6453,   6433,   6413,   6394
        dw      6373,   6353,   6333,   6313,   6292,   6272,   6251,   6230
        dw      6210,   6189,   6168,   6147,   6125,   6104,   6083,   6061
        dw      6040,   6018,   5996,   5974,   5952,   5930,   5908,   5886
        dw      5864,   5841,   5819,   5796,   5774,   5751,   5728,   5705
        dw      5682,   5659,   5636,   5613,   5589,   5566,   5542,   5519
        dw      5495,   5471,   5447,   5423,   5399,   5375,   5351,   5327
        dw      5303,   5278,   5254,   5229,   5204,   5180,   5155,   5130
        dw      5105,   5080,   5055,   5029,   5004,   4979,   4953,   4928
        dw      4902,   4877,   4851,   4825,   4799,   4773,   4747,   4721
        dw      4695,   4668,   4642,   4616,   4589,   4563,   4536,   4509
        dw      4483,   4456,   4429,   4402,   4375,   4348,   4321,   4294
        dw      4266,   4239,   4212,   4184,   4156,   4129,   4101,   4073
        dw      4046,   4018,   3990,   3962,   3934,   3906,   3878,   3849
        dw      3821,   3793,   3764,   3736,   3707,   3679,   3650,   3622
        dw      3593,   3564,   3535,   3506,   3477,   3448,   3419,   3390
        dw      3361,   3332,   3303,   3273,   3244,   3215,   3185,   3156
        dw      3126,   3096,   3067,   3037,   3007,   2978,   2948,   2918
        dw      2888,   2858,   2828,   2798,   2768,   2738,   2708,   2677
        dw      2647,   2617,   2586,   2556,   2526,   2495,   2465,   2434
        dw      2404,   2373,   2342,   2312,   2281,   2250,   2219,   2189
        dw      2158,   2127,   2096,   2065,   2034,   2003,   1972,   1941
        dw      1910,   1879,   1847,   1816,   1785,   1754,   1722,   1691
        dw      1660,   1628,   1597,   1566,   1534,   1503,   1471.  1440
        dw      1408,   1377,   1345,   1314,   1282,   1251,   1219,   1187
        radix hex

        radix dec
```

```
tbltres
        dw      1156,   1124,   1092,   1060,   1029,   997,    965,    933
        dw      902,    870,    838,    806,    774,    742,    711,    679
        dw      647,    615,    583,    551,    519,    487,    455,    423
        dw      391,    359,    327,    295,    263,    231,    199,    167
        dw      135,    103,    71,     39,     0,      32792,  32824,  32856
        dw      32888,  32920,  32952,  32984,  33016,  33048,  33079,  33111
        dw      33143,  33175,  33207,  33239,  33271,  33303,  33335,  33367
        dw      33399,  33431,  33463,  33495,  33526,  33558,  33590,  33622
        dw      33654,  33686,  33717,  33749,  33781,  33813,  33844,  33876
        dw      33908,  33940,  33971,  34003,  34034,  34066,  34098,  34129
        dw      34161,  34192,  34224,  34255,  34287,  34318,  34350,  34381
        dw      34412,  34444,  34475,  34506,  34538,  34569,  34600,  34631
        dw      34662,  34693,  34724,  34756,  34787,  34818,  34849,  34879
        dw      34910,  34941,  34972,  35003,  35034,  35064,  35095,  35126
        dw      35156,  35187,  35217,  35248,  35278,  35309,  35339,  35370
        dw      35400,  35430,  35461,  35491,  35521,  35551,  35581,  35611
        dw      35641,  35671,  35701,  35731,  35761,  35790,  35820,  35850
        dw      35879,  35909,  35939,  35968,  35997,  36027,  36056,  36085
        dw      36115,  36144,  36173,  36202,  36231,  36260,  36289,  36318
        dw      36347,  36375,  36404,  36433,  36461,  36490,  36518,  36547
        dw      36575,  36603,  36632,  36660,  36688,  36716,  36744,  36772
        dw      36800,  36828,  36855,  36883,  36911,  36938,  36966,  36993
        dw      37021,  37048,  37075,  37102,  37130,  37157,  37184,  37211
        dw      37237,  37264,  37291,  37318,  37344,  37371,  37397,  37423
        dw      37450,  37476,  37502,  37528,  37554,  37580,  37606,  37632
        dw      37657,  37683,  37709,  37734,  37760,  37785,  37810,  37835
        dw      37860,  37885,  37910,  37935,  37960,  37985,  38009,  38034
        dw      38058,  38083,  38107,  38131,  38156,  38180,  38204,  38227
        dw      38251,  38275,  38299,  38322,  38346,  38369,  38392,  38416
        dw      38439,  38462,  38485,  38508,  38530,  38553,  38576,  38598
        dw      38621,  38643,  38665,  38687,  38710,  38731,  38753,  38775
        dw      38797,  38819,  38840,  38861,  38883,  38904,  38925,  38946
        radix hex

m_sin
        movff rb,sin_saveb    ; guarda rb
        movff rb+1,sin_saveb+1

        movff ra,rt
        movff ra+1,rt+1
        clrf rt,1
        btfss ra,1,1
        goto tst5c
        goto tst5s

tst5s
        btfss ra,0,1
        goto dos
        goto tres

tst5c
        btfss ra,0,1
        goto cero
        goto uno

cero
        BCF STATUS,C
        rlcf rt+1,1
        rlcf rt,1
        call pointer0
        call t_read
        call t_rotate
        goto end_sin

uno
        BCF STATUS,C
        rlcf rt+1,1
        rlcf rt,1
        call pointer1
```

```
                call t_read
                call t_rotate
                goto end_sin

dos
                BCF STATUS,C
                rlcf rt+1,1
                rlcf rt,1
                call pointer2
                call t_read
                call t_rotate
                goto end_sin

tres
                BCF STATUS,C
                rlcf rt+1,1
                rlcf rt,1
                call pointer3
                call t_read
                call t_rotate
                goto end_sin

pointer0
                movlw upper (tblcero)
                movff WREG,TBLPTRU
                movlw low (tblcero)
                addwf rt+1,W,1
                movff WREG,TBLPTRL
                movlw high (tblcero)
                addwfc rt,W,1
                movff WREG,TBLPTRH
                return

pointer1
                movlw upper (tbluno)
                movff WREG,TBLPTRU
                movlw low (tbluno)
                addwf rt+1,W,1
                movff WREG,TBLPTRL
                movlw high (tbluno)
                addwfc rt,W,1
                movff WREG,TBLPTRH
                return

pointer2
                movlw upper (tbldos)
                movff WREG,TBLPTRU
                movlw low (tbldos)
                addwf rt+1,W,1
                movff WREG,TBLPTRL
                movlw high (tbldos)
                addwfc rt,W,1
                movff WREG,TBLPTRH
                return

pointer3
                movlw upper (tbltres)
                movff WREG,TBLPTRU
                movlw low (tbltres)
                addwf rt+1,W,1
                movff WREG,TBLPTRL
                mnvlw high (tbltres)
                addwfc rt,W,1
                movff WREG,TBLPTRH
                return

t_rotate    BCF STATUS,C,0
                rrcf ra,1
                rrcf ra+1,1
```

```
                BCF STATUS,C,0
                rrcf ra,1
                rrcf ra+1,1

                BCF STATUS,C,0
                rrcf ra,1
                rrcf ra+1,1

                BCF STATUS,C,0
                rrcf ra,1
                rrcf ra+1,1

                BCF STATUS,C,0
                rrcf ra,1
                rrcf ra+1,1
                return

t_read
                tblrd*+                         ;leer directamente de la tabla
                movff TABLAT,x0+1;Se lee el LSB primero
                tblrd*+
                movff TABLAT,x0                 ;Luego se lee el MSB
                movff x0,ra
                movff x0+1,ra+1
                return

end_sin         m_ldb sin_saveb
                return

;=============================================================================
;               FIN FIXED.INC
;=============================================================================
```

```
;*****************************************************************
;
;                                   DIV16.INC
;
;*****************************************************************

;    16/16 Bit Division Macros

#define _CSTATUS,0
#define _ZSTATUS,2

UDIV1616L    macro
                DM: 7
        MOVLW       D'16'
        MOVWF       LOOPCOUNT

LOOPU1616    RLCF        AARGB0,W
        RLCF        REMB1, F
        RLCF        REMB0, F
        MOVF        BARGB1,W
        SUBWF       REMB1, F
        MOVF        BARGB0,W
        BTFSS       _C
        INCFSZ      BARGB0,W
        SUBWF       REMB0, F
        BTFSC       _C
        GOTO        UOK66LL
        MOVF        BARGB1,W
        ADDWF       REMB1, F
        MOVF        BARGB0,W
        BTFSC       _C
        INCFSZ      BARGB0,W
        ADDWF       REMB0, F
        BCF         _C
UOK661L             RLCF        AARGB1, F
        RLCF        AARGB0, F
        DECFSZ      LOOPCOUNT, F
        GOTO        LOOPU1616
        endm

;*****************************************************************
;*****************************************************************

;    16/16 Bit Division de Punto Fijo (Sin signo) 16/16 -> 16.16

;    Input:  16 bit punto fijo (sin signo) en AARGB0, AARGB1
;            16 bit punto fijo (sin signo) en BARGB0, BARGB1

;    Uso:   CALL   FXD1616U

;    Output: 16 bit punto fijo (sin signo) cociente en AARGB0, AARGB1
;            16 punto fijo (sin signo) residuo en REMB0, REMB1

;    Resultado: AARG, REM <-- AARG / BARG


FXD1616U     CLRF        REMB0
        CLRF        REMB1

        UDIV1616L

        RETLW       0x00
;*****************************************************************
```

```
;*****************************************************************************



;-----------------------------------------------------------------------
;                                   FFT REAL
;
; Algoritmo de "Numerical recipes in C", cambiado de C++ a Assembler
;
;-----------------------------------------------------------------------

fft     nop                             ; Entrada FFT

; ------> inicializacion
        clrf j,1                ; (HIGH) j=0
        clrf j+1,1              ; (LOW)

; ------> reversion de bits
        clrf i,1                ; (HIGH) for(i=0;i<n-1;i+=2) {
        clrf i+1,1

iloop1  nop

if1     movff i,WREG                    ; if (j>i) {
        cpfsgt j,1
        goto no1
        goto si1
no1     cpfseq j,1
        goto endif1
        movff i+1,WREG
        cpfsgt j+1,1
        goto endif1

si1     nop
        m_ldai j,j+1                    ; m_ldai(j)
        m_ldbi i,i+1                    ; m_ldbi(i)
        m_stai i,i+1                    ; m_stai(i)
        m_mvba                          ; m_mvba()
        m_stai j,j+1                    ; m_stai(j)

        infsnz j+1,F,1
        incf j,F,1
        infsnz i+1,F,1
        incf i,F,1

        m_ldai j,j+1                    ; m_ldai(j+1)
        m_ldbi i,i+1                    ; m_ldbi(i+1)
        m_stai i,i+1                    ; m_stai(i+1)
        m_mvba                          ; m_mvba()
        m_stai j,j+1                    ; m_stai(j+1)

        bcf STATUS,N,0
        decf j+1,F,1
        btfsc STATUS,N,0
        goto tst1
        goto ok1
tst1    btfss j+1,7,1
        decf j,F,1
ok1     bcf STATUS,N,0
        decf i+1,F,1
        btfsc STATUS,N,0
        goto tst15
        goto endif1
tst15   btfss i+1,7,1
        decf i,F,1
```

```
endif1    nop                          ; } endif
          movlw B'00000000'
          movwf m+1,1                  ;(LOW)
          movlw B'00001000'
          movwf m,1                    ;(HIGH)

while1
          movlw .0                     ; while(m>=2 && j>=m) {
          cpfseq m,1
          goto while1si                ;es mayor que 2
          movlw D'2'
          cpfslt m+1,1
          goto while1si
          goto wend1

while1si
          movff m,WREG                 ;Comparamos si j>m
          cpfslt j,1
          goto $ + 6                   ;Si, entonces compare si es igual  ALOO
          goto wend1                   ;No, entonces salga del while1
          cpfseq j,1                   ;Si es igual, compare m+1
          goto while11si               ;Si es mayor, vaya a while11si
          movff m+1,WREG               ;Comparamos la parte baja de m (m+1)
          cpfslt j+1,1
          goto while11si               ;Si es mayor, entonces vaya a while11si
          goto wend1                   ;Si es menor, entonces salga del while1

while11si
          movff m+1,WREG               ; j-=m
          subwf j+1,F,1
          movff m,WREG
          subwfb j,F,1

          bcf STATUS,C,0               ; m>>=1
          rrcf m,F,1
          rrcf m+1,F,1

wloop1    goto while1                  ; }

wend1     movff m+1,WREG               ; j+=m
          addwf j+1,F,1
          movff m,WREG
          addwfc j,F,1

inext1    infsnz i+1,F,1               ; } next i
          incf i,1
          infsnz i+1,F,1
          incf i,1
          btfss i,4,1                  ;Cambiar a 4
          goto iloop1
          nop

; ------> FFT
          movlw D'2'                   ; mmax=2
          movwf mmax+1,1
          movlw D'0'
          movwf mmax,1
while2
          btfss mmax,4,1               ; while(n>mmax) {
          goto $+6
          goto wend2

          movff mmax+1, WREG           ; istep=mmax<<1
          bcf STATUS,C,0
          rlcf WREG,W,0                ;nota
          movwf istep+1,1
          movff mmax,WREG
          rlcf WREG,W,0                ;nota
          movwf istep,1
```

```
        ; theta=2pi/mmax
        m_ldaconst H'324'           ; m_ldaconst(3.14159265459)
        m_lddiv mmax, mmax+1                ; m_lddiv(mmax>>1)
        nop

        bcf STATUS,C,0
        rrcf rdiv,F,1
        rrcf rdiv+1,F,1

        call m_divi                         ; m_divi()
        m_sta theta                         ; m_sta(&theta)
        nop

        ; wpr=-2*sin(theta/2)^2
        movlw D'0'
        movwf rdiv,1
        movlw D'2'                          ; m_lddiv(2)
        movwf rdiv+1,1
        call m_divi                         ; m_divi()
        call m_sin                  ; m_sin()
        nop
        m_mvab                              ; m_mvab()
        call m_mult                         ; m_mult()
        m_mvab                              ; m_mvab()
        m_ldaconst H'8200'          ; m_ldaconst(-2.0)
        call m_mult                         ; m_mult()
        m_sta wpr                   ; m_sta(&wpr)
        nop

        ; wpi=n_sin(theta)
        m_lda theta                         ; m_lda(theta)
        call m_sin                  ; m_sin()
        m_sta wpi                   ; m_sta(&wpi)

        ; wr=1, wi=0
        m_ldaconst H'0100'          ; m_ldaconst(1.0)
        m_sta wr                    ; m_sta(&wr)
        m_ldaconst H'0000'          ; m_ldaconst(0.0)
        m_sta wi                    ; m_sta(&wi)
        nop

ffttst1 nop

        clrf m,1                    ; for(m=0;m<mmax-1;m+=2) {
        clrf m+1,1
mloop1  movff mmax+1,tmpi2+1                :mmax-1
        movff mmax,tmpi2

        movlw D'1'
        subwf tmpi2+1,1,1
        movlw D'0'
        subwfb tmpi2,1,1

        movff tmpi2,WREG            ;Comparamos si mmax-1 (tmpi2) > m
        cpfslt m,1
        goto $ + 6                  ;Si, entonces vaya a ffttst2
        goto ffttst2                ;No, entonces compare LSB
        cpfseq m,1
        goto mend1
        movff tmpi2+1,WREG                  ;Comparamos la parte baja de m (m+1)
        cpfslt m+1,1
        goto mend1                          ;Si es mayor, entonces vaya a mend1
                                            ;Si es menor, entonces vaya a ffttst2


ffttst2 nop
        movff m+1,i+1
        movff m,i

iloop2  nop                                 ; j=i+mmax
```

```
        movff i+1,j+1
        movff i,j
        movff mmax+1,WREG
        addwf j+1,F,1
        movff mmax,WREG
        addwfc j,F,1
        nop

ffttst3 nop

        ; h1r=wr*data[j]-wi*data[j+1]
        m_lda wi                        ; m_lda(wi)
        infsnz j+1,F,1                          ; m_ldbi(j+1)
        incf j,F,1
        m_ldbi j,j+1
        nop

        movlw D'1'
        subwf j+1,1,1
        movlw D'0'
        subwfb j,1,1

        call m_mult                             ; m_mult()
        m_sta h1r                       ; m_sta(&h1r)
        m_lda wr                        ; m_lda(wr)
        m_ldbi j, j+1                           ; m_ldbi(j)
        call m_mult                             ; m_mult()
        m_ldb h1r                       ; m_ldb(h1r)
        call m_sub                      ; m_sub()
        m_sta h1r                       ; m_sta(&h1r)
        nop

        ; h1i=wr*data[j+1]+wi*data[j]
        m_lda wi                        ; m_lda(wi)
        m_ldbi j, j+1                           ; m_ldbi(j)
        call m_mult                             ; m_mult()
        m_sta h1i                       ; m_sta(&h1i)
        m_lda wr                        ; m_lda(wr)

        infsnz j+1,F,1                          ; m_ldbi(j+1)
        incf j,F,1
        m_ldbi j, j+1

        call m_mult                             ; m_mult()
        m_ldb h1i                       ; m_ldb(h1i)
        call m_add                      ; m_add()
        m_sta h1i                       ; m_sta(&h1i)
        nop

        movlw D'1'
        subwf j+1,1,1
        movlw D'0'
        subwfb j,1,1

        ; rutina de mariposa

        ; data[j]=(data[i]-h1r)/2
        nop
        m_ldai i, i+1                           ; m_ldai(i)
        m_ldb h1r                       ; m_ldb(h1r)
        call m_sub                              ; m_sub()
        m_stai j, j+1                           ; m_stai(j)
        nop

        ; data[j+1]=(data[i+1]-h1i)/2
        infsnz i+1,F,1                          ; m_ldai(i+1)
        incf i,F,1

        m_ldai i, i+1
```

```
        movlw D'1'
        subwf i+1,1,1
        movlw D'0'
        subwfb i,1,1

        m_ldb h1i                       ; m_ldb(h1i)
        call m_sub                              ; m_sub()
        infsnz j+1,F,1                          ; m_stai(j+1)
        incf j,F,1
        m_stai j, j+1
        nop

        movlw D'1'
        subwf j+1,1,1
        movlw D'0'
        subwfb j,1,1

        ; data[i]=(data[i]+h1r)/2
        m_ldai i, i+1                           ; m_ldai(i)
        m_ldb h1r                       ; m_ldb(h1r)
        call m_add                              ; m_add()
        m_stai i, i+1                           ; m_stai(i)
        nop

        ; data[i+1]=(data[i+1]+h1i)/2
        infsnz i+1,F,1                          ; m_ldai(i+1)
        incf i,F,1
        m_ldai i, i+1


        movlw D'1'
        subwf i+1,1,1
        movlw D'0'
        subwfb i,1,1

        m_ldb h1i                       ; m_ldb(h1i)
        call m_add                              ; m_add()
        infsnz i+1,F,1                          ; m_stai(i+1)
        incf i,F,1
        m_stai i, i+1
        nop

        movlw D'1'
        subwf i+1,1,1
        movlw D'0'
        subwfb i,1,1

inext2
        btfss istep,4,1
        goto $+6
        goto iend2
        bcf STATUS,C,0
        movff istep+1,WREG
        addwf i+1,F,1
        movff istep,WREG
        addwfc i,F,1
        btfss i,4,1
        goto iloop2

        ; wtemp=wr
iend2   m_lda wr                        ; m_lda(wr)
        m_sta wtemp                             ; m_sta(&wtemp)
        nop
        ; wr=wr*wpr-wi*wpi+wr
        m_lda wi                        ; m_lda(wi)
        m_ldb wpi                       ; m_ldb(wpi)
        call m_mult                             ; m_mult()
        m_sta h1r                       ; m_sta(&h1r)
        m_lda wr                        ; m_lda(wr)
```

```
        m_ldb wpr               ; m_ldb(wpr)
        call m_mult                     ; m_mult()
        m_ldb h1r               ; m_ldb(h1r)
        call m_sub                      ; m_sub()
        m_ldb wr                ; m_ldb(wr)
        call m_add                      ; m_add()
        m_sta wr                ; m_sta(&wr)
        nop

        ; wi=wi*wpr+wtemp*wpi+wi
        m_lda wtemp                     ; m_lda(wtemp)
        m_ldb wpi               ; m_ldb(wpi)
        call m_mult                     ; m_mult()
        m_sta h1i               ; m_sta(&h1i)
        m_lda wi                ; m_lda(wi)
        m_ldb wpr               ; m_ldb(wpr)
        call m_mult                     ; m_mult()
        m_ldb h1i               ; m_ldb(h1i)
        call m_add                      ; m_add()
        m_ldb wi                ; m_ldb(wi)
        call m_add                      ; m_add()
        m_sta wi                ; m_sta(&wi)
        nop

mnext1  infsnz m+1,F,1                  ; } next m
        incf m,F,1
        infsnz m+1,F,1
        incf m,F,1
        goto mloop1
mend1   nop

        movff istep+1,mmax+1            ; mmax=istep
        movff istep,mmax

wloop2  goto while2                     ; } end while
wend2   return
```

```
;-------------------------------------------------------------------
;                                   FFT REAL
;
; Algoritmo de "Numerical recipes in C", cambiado de C++ a Assembler
;
;-------------------------------------------------------------------

realfft     nop

            call fft
            nop

            ; theta=2pi/n
            m_ldaconst H'324'           ; m_ldaconst(3.14159265459)

            movlw B'00000000'
            movwf rdiv+1,1                      ;(LOW)    m_lddiv(n>>1)
            movlw B'00001000'
            movwf rdiv,1                        ;(HIGH)

            call m_divi                         ; m_divi()
            m_sta theta                         ; m_sta(&theta)

            ; wpr=-2*sin(theta/2)^2
            movlw D'0'
            movwf rdiv,1
            movlw D'2'                          ; m_lddiv(2)
            movwf rdiv+1,1
            call m_divi                         ; m_divi()
            call m_sin                  ; m_sin()
            nop
            m_mvab                              ; m_mvab()
            call m_mult                         ; m_mult()
            m_mvab                              ; m_mvab()
            m_ldaconst H'8200'          ; m_ldaconst(-2.0)
            call m_mult                         ; m_mult()
            m_sta wpr                   ; m_sta(&wpr)
            nop

            ; wpi=n_sin(theta)
            m_lda theta                         ; m_lda(theta)
            call m_sin                  ; m_sin()
            m_sta wpi                   ; m_sta(&wpi)

            ; wr=1+wpr, wi=wpi
            m_ldb wpr                   ; m_ldb(wpr)
            m_ldaconst H'0100'          ; m_ldaconst(1.0)
            call m_add                  ; m_add()
            m_sta wr                    ; m_sta(&wr)
            m_lda wpi                   ; m_lda(wpi)
            m_sta wi                    ; m_sta(&wi)
            nop

            clrf i1,1
            movlw D'2'                          ; for(i=2;i<=(n>>2);i++) {
            movwf i+1,1
            clrf i,1
iloop3
            movlw B'00000100'
            cpfsgt i,1
            goto testequall
            goto iend3

testequall
```

```
        cpfseq i,1
        goto ok9
        goto tstlow9
tstlow9
        tstfsz i+1,1
        goto iend3
        goto ok9

ok9
        movff i+1,i1+1                  ; i1=i+i-2
        movff i,i1
        movff i+1,WREG                  ; byte menos significativo primero
        addwf i1+1,1,1
        movff i,WREG                    ; y luego byte mas significativo con carry
        addwfc i1,1,1
        movlw D'2'
        subwf i1+1,1,1
        movlw D'0'
        subwfb i1,1,1
        nop

        movff i1+1,WREG                 ; i2=1+i1
        movwf i2+1,1
        movff i1,WREG
        movwf i2,1
        infsnz i2+1,F,1
        incf i2,F,1
        nop

        movff i2+1,WREG                 ; i3=n-i2+1
        sublw B'00000000'
        movwf i3+1,1
        movff i2,WREG
        sublw B'00010000'
        movwf i3,1
        btfsc STATUS,C
        decf i3,F,1
        infsnz i3+1,F,1
        incf 13,F,1
        nop

        movff i3+1,i4+1                 ; i4=1+i3
        movff i3,i4
        infsnz i4+1,F,1
        incf i4,F,1
        nop

        ; h1r=(data[i1]+data[i3])/2
        m_ldai i1, i1+1                 ; m_ldai(i1)
        m_ldbi i3, i3+1                 ; m_ldbi(i3)
        call m_add              ; m_add()
        call m_div2                     ; m_div2()
        m_sta h1r               ; m_sta(&h1r)
        nop

        ; h1i=(data[i2]-data[i4])/2
        m_ldai i2, i2+1                 ; m_ldai(i2)
        m_ldbi i4, i4+1                 ; m_ldbi(i4)
        call m_sub              ; m_sub()
        call m_div2                     ; m_div2()
        m_sta h1i               ; m_sta(&h1i)
        nop

        ; h2r=(data[i2]+data[i4])/2
        m_ldai i2, i2+1                 ; m_ldai(i2)
        m_ldbi i4, i4+1                 ; m_ldbi(i4)
        call m_add              ; m_add()
        call m_div2                     ; m_div2()
        m_sta h2r               ; m_sta(&h2r)
        nop
```

```
; h2i=-(data[i1]-data[i3])/2
m_ldai i3, i3+1                         ; m_ldai(i3)
m_ldbi i1, i1+1                         ; m_ldbi(i1)
call m_sub                      ; m_sub()
call m_div2                             ; m_div2()
m_sta h2i                       ; m_sta(&h2i)
nop

; data[i1]=h1r+wr*h2r-wi*h2i
m_lda wi                        ; m_lda(wi)
m_ldb h2i                       ; m_ldb(h2i)
call m_mult                             ; m_mult()
m_sta wtemp                             ; m_sta(&wtemp)
m_lda wr                        ; m_lda(wr)
m_ldb h2r                       ; m_ldb(h2r)
call m_mult                             ; m_mult()
m_ldb wtemp                             ; m_ldb(wtemp)
call m_sub                      ; m_sub()
m_ldb h1r                       ; m_ldb(h1r)
call m_add                      ; m_add()
m_stai i1, i1+1                         ; m_stai(i1)
nop

; data[i2]=h1i+wr*h2i+wi*h2r
m_lda wi                        ; m_lda(wi)
m_ldb h2r                       ; m_ldb(h2r)
call m_mult                             ; m_mult()
m_sta wtemp                             ; m_sta(&wtemp)
m_lda wr                        ; m_lda(wr)
m_ldh h2i                       ; m_ldb(h2i)
call m_mult                             ; m_mult()
m_ldb wtemp                             ; m_ldb(wtemp)
call m_add                      ; m_add()
m_ldb h1i                       ; m_ldb(h1i)
call m_add                      ; m_add()
m_stai i2, i2+1                         ; m_stai(i2)
nop

; data[i3]=h1r-wr*h2r+wi*h2i
m_lda wr                        ; m_lda(wr)
m_ldb h2r                       ; m_ldb(h2r)
call m_mult                             ; m_mult()
m_sta wtemp                             ; m_sta(&wtemp)
m_lda wi                        ; m_lda(wi)
m_ldb h2i                       ; m_ldb(h2i)
call m_mult                             ; m_mult()
m_ldb wtemp                             ; m_ldb(wtemp)
call m_sub                      ; m_sub()
m_ldb h1r                       ; m_ldb(h1r)
call m_add                      ; m_add()
m_stai i3, i3+1                         ; m_stai(i3)
nop

; data[i4]=-h1i+wr*h2i+wi*h2r
m_lda wr                        ; m_lda(wr)
m_ldb h2i                       ; m_ldb(h2i)
call m_mult                             ; m_mult()
m_sta wtemp                             ; m_sta(&wtemp)
m_lda wi                        ; m_lda(wi)
m_ldb h2r                       ; m_ldb(h2r)
call m_mult                             ; m_mult()
m_ldb wtemp                             ; m_ldb(wtemp)
call m_add                      ; m_add()
m_ldb h1i                       ; m_ldh(h1i)
call m_sub                      ; m_sub()
m_stai i4, i4+1                         ; m_stai(i4)
nop

; wtemp=wr
```

```
            m_lda wr                    ; m_lda(wr)
            m_sta wtemp                         ; m_sta(&wtemp)

            ; wr=wr*wpr-wi*wpi+wr
            m_lda wi                    ; m_lda(wi)
            m_ldb wpi                   ; m_ldb(wpi)
            call m_mult                         ; m_mult()
            m_sta h1r                   ; m_sta(&h1r)
            m_lda wr                    ; m_lda(wr)
            m_ldb wpr                   ; m_ldb(wpr)
            call m_mult                         ; m_mult()
            m_ldb h1r                   ; m_ldb(h1r)
            call m_sub                  ; m_sub()
            m_ldb wr                    ; m_ldb(wr)
            call m_add                  ; m_add()
            m_sta wr                    ; m_sta(&wr)

            ; wi=wi*wpr+wtemp*wpi+wi
            m_lda wi                    ; m_lda(wi)
            m_ldb wpr                   ; m_ldb(wpr)
            call m_mult                         ; m_mult()
            m_sta h1r                   ; m_sta(&h1r)
            m_lda wtemp                         ; m_lda(wtemp)
            m_ldb wpi                   ; m_ldb(wpi)
            call m_mult                         ; m_mult()
            m_ldb h1r                   ; m_ldb(h1r)
            call m_add                  ; m_add()
            m_ldb wi                    ; m_ldb(wi)
            call m_add                  ; m_add()
            m_sta wi                    ; m_sta(&wi)
            nop

inext3      infsnz i+1,F,1                      ; } next i
            incf i,F,1
            goto iloop3
iend3       nop

            ; h1r=data[0]
            clrf i,1
            movlw 0                             ; m_ldai(0)
            movwf i+1,1
            m_ldai i,i+1
            m_sta h1r                   ; m_sta(&h1r)

            ; data[0]=h1r+data[1]
            movlw 1                             ; m_ldbi(1)
            movwf i+1,1
            m_ldbi i, i+1
            call m_add                  ; m_add()
            movlw 0                             ; m_stai(0)
            movwf i+1,1
            m_stai i, i+1

            ; data[1]=h1r-data[1]
            m_lda h1r                   ; m_lda(h1r)
            movlw 1                             ; m_ldbi(1)
            movwf i+1,1
            m_ldbi i, i+1
            call m_sub                  ; m_sub()
            movlw 1                             ; m_stai(1)
            movwf i+1,1
            m_stai i, i+1
endrft      return
;=============================================================================
;                   FIN FFT
;=============================================================================
```

```
;=============================================================================
;                              SERIAL.INC
;
;Rutina para leer los datos del puerto serial y guardarlos en RAM Externa
;=============================================================================

start_send
                BANKSEL num_sample
                clrf num_sample,1
                clrf num_sample+1,1
                call PORTDin
                goto start_send1

send            nop
                movff ra,TXREG
verify          btfss TXSTA,TRMT,0
                goto verify
                return

start_send1     nop
                nop
                movff num_sample+1,PORTB    ;Escogemos en que posicion de memoria
                nop
                movff num_sample,PORTA              ;querernos guardar los datos
                nop
                call READ
                nop
                nop
                movff PORTD,ra                      ;Movemos los datos a RAM
                nop
                nop
                call DISABLE
                nop
                call send
                nop
                bcf STATUS,C,0
                movlw .1
                addwf num_sample+1,F,1
                movlw .0
                addwfc num_sample,F,1
                btfss num_sample,5,1
                goto start_send1
                return
```

```
;=============================================================================
;                                   VARS.INC
;
;
;Rutina de Declaracion de Variables.  Todas se encuentran en Banco 4
;=============================================================================

;-----------------------------------------------------------------------------
; Banco 4
;-----------------------------------------------------------------------------

        cblock H'0402'          ; arithmetic working registers
                ra:2            ; (format s-2-13)
                rb:2            ; (format s-2-13)
                tmpi:2          ; temporary variable
                tmpi2:2         ; temporary variable
                num_sample:2    ; Numero de Muestra
                m:2
                j:2
                i:2
                wr:2            ; (formato s-2-13)
                wpr:2           ; (formato s-2-13)
                wi:2            ; (formato s-2-13)
                wpi:2           ; (formato s-2-13)
                theta:2         ; (formato s-2-13)
                mmax:2          ; contadores
                rdiv:2          ; (8 bit unsigned int)
                istep:2
                i1:2
                i2:2
                i3:2
                i4:2
                h1r:2           ; (formato s-2-13)
                h1i:2           ; (formato s-2-13)
                h2r:2           ; (formato s-2-13)
                h2i:2           ; (formato s-2-13)
                wtemp:2         ; (formato s-2-13)
                mres3:1
                mres2:1
                mres1:1
                mres0:1
                msign:1
                x0:2            ; para m_sin
                x1:2
                x2:2
                sin_saveb:2
                rt:2
                sign:1
                LOOPCOUNT
        endc

;=============================================================================
;                               Fin VARS3.INC
;=============================================================================
```

APÉNDICE B

Descripción del Código

Para poder entender la rutina de la FFT de una forma más fácil, se explicara el algoritmo de la FFT en C++. El código equivalente en Assembler de PIC se puede ver en el anexo A.

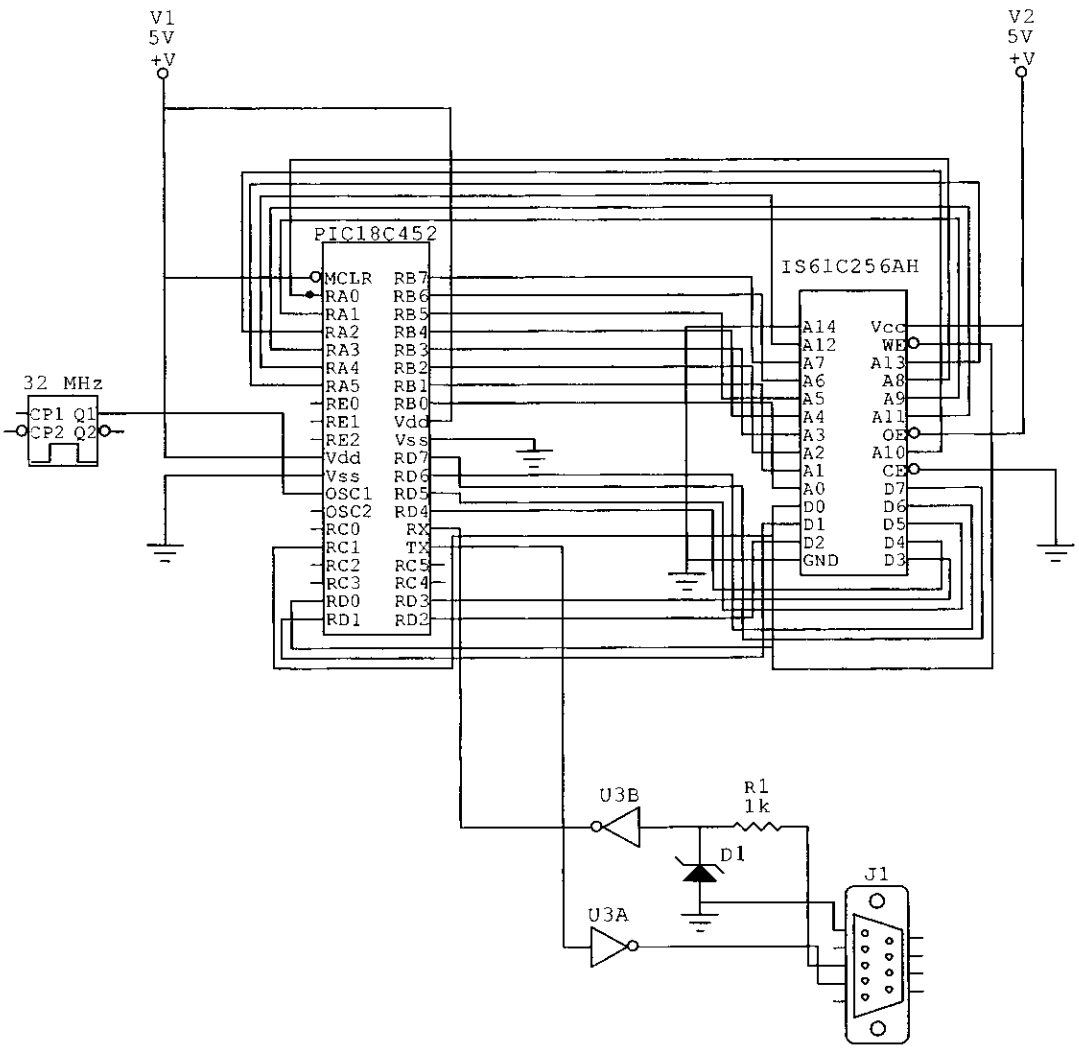
C++

```
#define SWAP(a,b) tempr=(a); (a)=(b); (b)=tempr      //Definición de instrucción para
                                                          //intercambiar valores
void four1(unsigned long nn,int isign)               //Rutina de reversión de bits y
{                                                         //lema de Danielson-Lanczos
        unsigned long n,mmax,m,j,i,istep;                 //Declaración de Variables
        float tempr,tempi;                                //wi = seno(theta), wr = coseno(theta)
        double wtemp,wr,wpr,wpi,wi,theta;                 //wpr = -alfa = -2sen(0.5*delta)
        isign = 1;                                        //wpi = beta = seno(delta)
                                                          //n = numero total de muestras
        n = nn << 1;                                            //nn = numero total de muestras/2
        j=1;
        for (i = 1;i < n;i+=2)                            //Esta es la parte de la reversión de bits
        {
                if (j > i)
                {
                        SWAP(data[j],data[i]);            //Se intercambian
                        SWAP(data[j+1],data[i+1]);        //los números complejos.
                }

                m = n>> 1;
                while (m >= 2 && j >= m)
                {
                        j -= m;
                        m >>= 1;
                }
                j += m;
        }

//Aquí empieza la sección del lema de  Danielson-Lanczos

mmax = 2:
while (n > mmax)                                          //Cielo se ejecuta log 2 nn veces
{
        istep = mmax << 1;
        theta = isign*(6.28318530717959/mmax);            //Inicialización de recurrencia
        wtemp = sin(0.5*theta);                                 //trigonométrica
        wpr = -2.0*wtemp*wtemp;
        wpi = sin(theta);
        wr = 1.0;
        wi = 0.0;
        for (m = 1;m < mmax;m += 2)
        {
                for (i = m;i <= n;i += istep)
                {
                j = i + mmax;
                tempr = wr*data[j] - wi*data[j+1];        //Formula de Euler
                tempi = wr*data[j+1] + wi*data[j];        //Formula de Euler
```

```
                data[j] = data[i]-tempr;                    //Estructura de Mariposa
                data[j+1] = data[i+1] - tempi;              //Estructura de Mariposa
                data[i] += tempr;                           //Sumatoria (Reales)
                data[i+1] += tempi;                         //Sumatoria (Imaginarios)
                }
                wr = (wtemp=wr)*wpr-wi*wpi+wr;              //Recurrencia trigonométrica
                wi = wi*wpr+wtemp*wpi+wi;
        }
        mmax = istep;
}
}



void realft(unsigned long n, int isign)
{
        void four1(unsigned long nn,int isign);
        unsigned long i,i1,i2,i3,i4,np3;
        float c1=0.5,c2,h1r,h1i,h2r,h2i;
        double wr,wi,wpr,wpi,wtemp,theta;

        theta = 3.141592653589793/(double) (n>>1);          //Recurrencia trigonométrica
        if (isign == 1)
        {
                c2 = -0.5;
                four1(n>>1,1);                                          //FFT
        }
        else                                                            //FFT inversa
        {
                c2 = 0.5;
                theta = -theta;
        }
        wtemp = sin(0.5*theta);
        wpr = -2.0*wtemp*wtemp;
        wpi = sin(theta);
        wr = 1.0+wpr;
        wi = wpi;
        np3 = n+3;

        for (i = 2;i<=(n>>2);i++)
        {
                i4 = 1+(i3=np3-(i2=1+(i1=i+i-1)));
                h1r = c1*(data[i1]+data[i3]);                   //Las dos transformadas
                h1i = c1*(data[i2]-data[i4]);                   //se separan del arreglo
                h2r = -c2*(data[i2]+data[i4]);
                h2i = c2*(data[i1]-data[i3]);
                data[i1] = h1r+wr*h2r-wi*h2i;           //Aquí se recombinan para
                data[i2] = h1i+wr*h2i+wi*h2r;           //formar la transformada
                data[i3] = h1r-wr*h2r+wi*h2i;           //verdadera de los datos reales
                data[i4] = -h1i+wr*h2i+wi*h2r;          //originales
                wr = (wtemp=wr)*wpr-wi*wpi+wr;                  //Recurrencia trigonométrica
                wi = wi*wpr+wtemp*wpi+wi;
        }
        if (isign == 1)                                                 //Transformada inversa
        {
                data[1] = (h1r=data[1])+data[2];
                data[2] = h1r-data[2];
        }
}
```

APÉNDICE C

Diagrama del Circuito:



Memoria Externa: IS61C256AH
Microcontrolador: PIC18C452

En el microcontrolador, todos los pines con designación RXx corresponden a las entradas y salidas de los puertos. Por ejemplo, las designaciones de RB0 hasta RB7 corresponden a todas las siete entradas y salidas del puerto B. En la memoria externa, todas las designaciones que tengan Ax se refieren al bus de direcciones, mientras que todas las designaciones Dx se refieren al bus de datos, donde x es el número de bit correspondiente.