Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela Mecánica Eléctrica

IMPLEMENTACIÓN DE UN ANALIZADOR DE ESPECTRO PARA FRECUENCIAS DE AUDIO UTILIZANDO UN PROCESADOR DIGITAL DE SEÑALES (DSP)

Héctor Leonel Munguía Valiente
Asesorado por el Ing. Conrado Guillermo Szasdi Soto

Guatemala, julio de 2006

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

IMPLEMENTACIÓN DE UN ANALIZADOR DE ESPECTRO PARA FRECUENCIAS DE AUDIO UTILIZANDO UN PROCESADOR DIGITAL DE SEÑALES (DSP)

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

HÉCTOR LEONEL MUNGUÍA VALIENTE

ASESORADO POR EL INGENIERO CONRADO GUILLERMO SZASDI SOTO

AL CONFERÍRSELE EL TÍTULO DE
INGENIERO ELECTRÓNICO

GUATEMALA, JULIO DE 2006

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	Inga. Glenda Patricia García Soria
VOCAL II	Lic. Amahán Sánchez Álvarez
VOCAL III	Ing. Julio David Galicia Celada
VOCAL IV	Br. Kenneth Issur Estrada Ruiz
VOCAL V	Br. Elisa Yazminda Vides Leiva
SECRETARIA	Inga. Marcia Ivonne Véliz Vargas

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

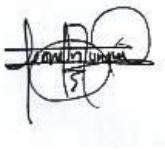
DECANO	Ing. Herbert René Miranda Barrios
EXAMINADOR	Ing. Gustavo Benigno Orozco Godínez
EXAMINADOR	Ing. Kenneth Ramiro Barnett Castellanos
EXAMINADOR	Ing. Enrique Edmundo Ruiz Carballo
SECRETARIA	Inga. Gilda Marina Castellanos de Illescas

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

IMPLEMENTACIÓN DE UN ANALIZADOR DE ESPECTRO PARA FRECUENCIAS DE AUDIO UTILIZANDO UN PROCESADOR DIGITAL DE SEÑALES (DSP),

tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, el 15 de marzo de 2001.



Héctor Leonel Munguía Valiente

Guatemala, 25 de abril de 2006

Ingeniero
Julio César Solares Peñate
Coordinador Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería
Universidad de San Carlos de Guatemala.

Ingeniero Solares:

Por este medio me dirijo a usted, para informarle que he revisado el trabajo de graduación titulado, **Implementación de un analizador de espectro para frecuencias de audio utilizando un procesador digital de señales (DSP)**, que desarrolló el estudiante Héctor Leonel Munguía Valiente, de la carrera de ingeniería electrónica, el cual, a mi criterio, cumple con los objetivos propuestos.

Por ello, el autor de este trabajo de graduación y yo, como su asesor, nos hacemos responsables por el contenido y conclusiones del mismo.

Atentamente,

Ing. Conrado G. Szasdi Soto
Colegiado No. 3116

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Guatemala, 15 de mayo 2006.

Señor Director
Ing. Mario Renato Escobedo Martinez
Escuela de Ingenieria Mecànica Elèctrica
Facultad de Ingenieria, USAC.

Señor Director:

Me permito dar aprobación al trabajo de Graduación titulado: **Implementación de un analizador de espectro para frecuencias de audio utilizando un procesador digital de señales (DSP)** desarrollado por el estudiante; Hèctor Leonel Munguia Valiente, por considerar que cumple con los requisitos establecidos para tal fin.

Sin otro particular, aprovecho la oportunidad para saludarle.

Atentamente,

ID Y ENSEÑAD A TODOS

Ing. Julio Cesar Solares Peñate
Coordinador Area de Electrònica

JCSP/ao

UNIVERSIDAD DE SAN CARLOS
FACULTAD DE INGENIERIA
DIRECCION ESCUELA DE INGENIERIA
MECANICA ELECTRICA
DE GUATEMALA

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

El Director de la Escuela de Ingenieria Mecànica Elèctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Area, al trabajo de Graduación del estudiante; Héctor Leonel Munguia Valiente titulado: **Implementación de un analizador de espectro para frecuencias de audio utilizando un procesador digital de señales (DSP),** procede a la autorización del mismo.

Ing. Mario Renato Escobedo Martínez
DIRECTOR

UNIVERSIDAD DE SAN CARLOS
FACULTAD DE INGENIERIA
DIRECCION ESCUELA DE INGENIERIA
MECANICA ELECTRICA
GUATEMALA

GUATEMALA, 16 DE MAYO 2,006.

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG. 243-2006.

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al trabajo de graduación titulado: **IMPLEMENTACIÓN DE UN ANALIZADOR DE ESPECTRO PARA FRECUENCIAS DE AUDIO UTILIZANDO UN PROCESADOR DIGITAL DE SEÑALES (DSP),** presentado por el estudiante universitario **Héctor Leonel Munguía Valiente,** procede a la autorización para la impresión del mismo.

IMPRÍMASE.

Ing. Murphy Olympo Paiz Recinos
DECANO

Guatemala, julio 19 de 2,006

/gdech

Todo por la Tecnología Mecánica
Dr. Carlos Martínez Durán
2006: Centenario de su Nacimiento

AGRADECIMIENTOS A:

DIOS

Por haberme permitido culminar este trabajo, dándome la fuerza y el entendimiento necesario.

MIS PADRES

Quienes me apoyaron y me dieron el aliento necesario para terminar el presente trabajo de graduación.

MI ESPOSA

Por no permitir que este proyecto quedara en el olvido y por ser una continua fuente de inspiración.

AL BEBÉ QUE ESTA EN CAMINO

Porque el anuncio de tu llegada fue el impulso final, para la conclusión de este proyecto

ROBERTO Y KARLA

Por sus oraciones y apoyo.

ALFREDO, VICKY, JOSIAS Y RAQUEL

Por motivarme a culminar este esfuerzo.

ING. CONRADO SZASDI

Por su paciencia y valiosa asesoría.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES VII
LISTA DE ABREVIATURAS XI
GLOSARIO XIII
RESUMEN XIX
OBJETIVOS XXI
INTRODUCCIÓN XXIII

1. **EL DOMINIO DE LA FRECUENCIA** 1
   - 1.1 Introducción 1
   - 1.2 El modelo fasorial 2
     - 1.2.1 El modelo fasorial discreto 3
     - 1.2.2 Modelado de señales senoidales 3
   - 1.3 Series de Fourier 5
   - 1.4 Transformada de Fourier 6
   - 1.5 Transformada discreta de Fourier (DFT) 7
   - 1.6 Transformada rápida de Fourier (FFT) 11
     - 1.6.1 Decimación en el tiempo 13
     - 1.6.2 La Mariposa 16
   - 1.7 Análisis espectral utilizando la FFT 17
   - 1.8 La FFT y los DSP 20

2. **PROCESADORES DIGITALES DE SEÑALES (DSP)** 21
   - 2.1 Introducción 21
   - 2.2 Arquitectura de los DSP 22

2.2.1 CPU 23
2.2.2 Organización de la memoria 25
2.2.2.1 Arquitectura Von Neumman 26
2.2.2.2 Arquitectura Harvard 26
2.2.2.3 Arquitectura Von Neumman modificada 27
2.2.3 Conjunto de instrucciones 28
2.2.3.1 Computacionales 28
2.2.3.2 Carga y almacenamiento 29
2.2.3.3 Control de programa 29
2.2.3.4 Multifunción 29
2.2.3.5 Misceláneas 29
2.2.4 Modos de direccionamiento 30
2.2.4.1 Direccionamiento de autoincremento 30
2.2.4.2 Direccionamiento circular 30
2.2.4.3 Direccionamiento de bit en reversa 30
2.2.5 Periféricos 31
2.2.5.1 Controlador DMA 31
2.2.5.2 Puertos seriales 31
2.2.5.3 Puerto *host* 32
2.2.5.4 Puerto de comunicación 32
2.2.5.5 Temporizador 32
2.2.5.6 Puertos especiales 33
2.3 Formato de datos 33
2.3.1 Datos de punto fijo 33
2.3.2 Datos de punto flotante 35
2.4 Programación de los DSP 36
2.4.1 Programación en lenguaje ensamblador 36
2.4.2 Programación en lenguaje C 37

3. **SELECCIÓN DE COMPONENTES DE HARDWARE** 39
3.1 Introducción 39
3.2 Especificaciones del sistema 39
3.2.1 Algoritmo a ejecutar 40
3.2.2 Memoria interna 41
3.2.3 Sistema de adquisición de señales 41
3.2.4 Comunicación con PC 42
3.2.5 Formato de datos 42
3.2.6 Herramientas para desarrollo de software 42
3.2.7 Requerimientos adicionales 43
3.3 Principales fabricantes de chips DSP 43
3.3.1 Texas Instruments™ 44
3.3.1.1 TMS320C3x 45
3.3.1.2 TMS320C4x 48
3.3.1.3 TMS320C67x 50
3.3.2 Analog Devices™ 51
3.3.2.1 ADSP-21000 SHARC 52
3.4 Sistemas de evaluación 54
3.4.1 Texas Instruments™ TMS320C3x DSP Starter Kit 54
3.4.2 Analog Devices™ ADSP-21065L EZ-Kit Lite 56
3.5 Criterios para selección del sistema a utilizar 57
3.6 Selección del sistema 58
3.6.1 Costo 58
3.6.2 Herramientas de desarrollo 58
3.6.3 Documentación 59
3.6.4 Hardware 59
3.6.5 Conclusión 60

4. **COMPONENTES DE HARDWARE Y SOFTWARE DEL DSK** 63
4.1 Introducción 63
4.2 Arquitectura del DSP TMS320C31 63
4.2.1 CPU 64
4.2.1.1 Multiplicador de números enteros y de punto flotante 64
4.2.1.2 Unidad aritmética lógica (ALU) 65
4.2.1.3 Unidades auxiliares para aritmética de registros (ARAUs) 66
4.2.1.4 Archivo primario de registros del CPU 66
4.2.2 Organización de la memoria 66
4.2.2.1 Memoria RAM y cache 67
4.2.2.2 Modos de direccionamiento de memoria 68
4.2.3 Operación del bus interno 69
4.2.4 Interrupciones 70
4.2.5 Periféricos 70
4.2.5.1 Temporizadores 70
4.2.5.2 Puerto serial 71
4.2.5.3 Acceso directo a memoria (DMA) 71
4.3 Circuito de interfase analógica (AIC) 72
4.4 Software para programación del DSP 73
4.4.1 Ensamblador DSK 73
4.4.2 El depurador 73
4.5 Procedimiento para elaboración de programas 75
4.6 Funcionamiento general del sistema DSK 76
4.6.1 Interfase de hardware 76
4.6.2 Interfase de hardware para *host* 76
4.6.3 Interfase de hardware con el AIC TLC32040 77
4.6.4 Mapa de memoria del DSK 78

4.6.5 Kernel de comunicaciones del DSK 79
4.6.5.1 Paquete de datos 79
4.6.5.2 Comandos 79
4.7 Software para comunicación con PC 81

5. **DESARROLLO DEL SOFTWARE PARA EL SISTEMA DSK** 83
5.1 Introducción 83
5.2 Especificaciones del software 83
5.3 Diagrama de flujo 84
5.4 Programación del AIC y adquisición de señales 85
5.4.1 Reiniciar AIC 88
5.4.2 Inicializar temporizador del 'C31 89
5.4.3 Inicializar puerto serial 90
5.4.4 Programar AIC 91
5.4.4.1 Comunicación primaria 92
5.4.4.2 Comunicación secundaria 92
5.4.5 Adquisición y almacenamiento de datos 94
5.5 Programación del algoritmo para el cálculo de la FFT 95
5.5.1 Creación de tabla con factores twiddle 98
5.5.2 Cálculo de la mariposa 99
5.5.3 Cálculo final 101
5.5.3.1 Función de ventana 102
5.5.3.2 Cálculo del logaritmo 103
5.5.3.3 Empaquetar resultado 104
5.6 Programa definitivo 104
5.7 Ensamblado 107
5.8 Depuración 108
5.8.1 Código de inicialización 108
5.8.2 Código para el cálculo de la FFT 108

6. **SOFTWARE PARA PC** 111
6.1 Introducción 111
6.2 Especificaciones 111
6.3 Desarrollo del software 112
6.3.1 Algoritmo 112
6.3.2 Inicializar comunicación con el DSP 114
6.3.3 Inicialización de la aplicación 116
6.3.4 Actualización de parámetros y graficación 119
6.4 Compilación 124
6.5 Depuración 125
6.5.1 Pruebas 126
6.6 Programa ejecutable 129

CONCLUSIONES 131
RECOMENDACIONES 133
BIBLIOGRAFÍA 135
APÉNDICE I 137
APÉNDICE II 149

ÍNDICE DE ILUSTRACIONES

FIGURAS

1	El modelo fasorial	2
2	Representación fasorial de cosΦ	5
3	Diagrama de flujo de una FFT de 4 puntos para k=0	15
4	Diagrama de flujo completo para una FFT de 4 puntos	16
5	Mariposa típica para una FFT de 4 puntos	17
6	Señal en el dominio del tiempo	18
7	Espectro de frecuencia para la señal de ejemplo de la figura 6	19
8	Unidades computacionales de un DSP típico	24
9	Arquitectura Von Neumman	26
10	Arquitectura Harvard	27
11	Representación de punto flotante	35
12	Principales familias de DSP producidos por Texas Instruments™	45
13	Diagrama a bloques del TMS320C30	47
14	Diagrama a bloques del TMS320C40	49
15	Diagrama a bloques del TMS320C6701	50
16	Principales familias de DSP producidos por Analog Devices™	52
17	Diagrama a bloques del ADSP-21065L	53
18	Diagrama a bloques del TMS320C3x DSP Starter Kit	55
19	Diagrama a bloques del ADSP21065L EZ-Kit Lite	56
20	Arquitectura del TMS320C31	65
21	Diagrama funcional a bloques del TLC32040	72

22	Pantalla del programa depurador	74
23	Proceso para escribir programas para el sistema DSK	75
24	Mapa de memoria del DSK	78
25	Formato de paquete de datos del TMS320C31	79
26	Diagrama de flujo del software para el DSK	86
27	Diagrama a bloques de código para programación del AIC	87
28	Conexión entre el DSP y el AIC	88
29	Formato de datos para comunicación primaria	92
30	Formato de datos para comunicación secundaria	93
31	Diagrama de flujo del proceso de adquisición de señales	95
32	Ejemplo de una FFT radix-2 de 8 puntos	96
33	Diagrama de flujo del software para el cálculo de la FFT	97
34	Mariposa típica en el cálculo de una FFT radix-2	99
35	Diagrama de flujo de cálculo final de la FFT	101
36	Ejemplo del cálculo efectuado por la función de ventana	103
37	Distribución de datos y código en la memoria interna del ‘C31	106
38	Especificaciones de software para PC	112
39	Algoritmo para desarrollo de software	113
40	Pantalla del analizador de espectro sin menú de opciones	117
41	Pantalla del analizador de espectro con menú de opciones	118
42	Configuración para ejecutar pruebas de medición de señales	126
43	Espectro de frecuencia para una onda senoidal de 1 Khz	127
44	Espectro de frecuencia de una señal senoidal de 5 Khz	128
45	Espectro de frecuencia de una señal triangular de 1 Khz	128

TABLAS

I	Integrantes de la familia TMS320C3x	48
II	Diferentes miembros de la familia TMS320C4x	49
III	Principales integrantes de la familia TMS320C67x	51
IV	Componentes de la familia ADSP-2100	54
V	Comparación entre el DSK y el EZ-Kit Lite	61
VI	Comandos principales del kernel de comunicaciones y su función	80
VII	Rutinas de interfase de PC con el DSK	82
VIII	Palabras de control, para programar temporizador del ‘C31	90
IX	Palabras de control para programación del puerto serial	91
X	Valores iniciales para registros A, B de control del AIC	94
XI	Funciones en C utilizadas en el programa para PC	114
XII	Opciones de la pantalla de menú del analizador de espectro	118

LISTA DE ABREVIATURAS

ADC	Convertidor de una señal analógica en digital.
AIC	Circuito de interfase analógica.
ALU	Unidad aritmética lógica.
ARAUS	Unidad para aritmética de registros.
CODEC	Codificador decodificador.
CPU	Unidad central de proceso.
DAC	Convertidor de una señal digital en analógica.
dB	Decibel.
DFT	Transformada discreta de Fourier
DMA	Acceso directo a memoria.
DOS	Sistema operativo de disco.
DSK	Kit de inicialización a los DSP.
DSP	Procesador digital de señales
E/S	Abreviatura de Entrada/Salida
EPROM	Memoria programable de solo lectura.
FFT	Transformada rápida de Fourier
GFLOP	Mil millones de operaciones de punto flotante
IDE	Entorno de desarrollo integrado
ISA	Arquitectura standard de la industria.
KHz	Unidad de medida de frecuencia
MAC	Multiplicador acumulador
MBit/Seg	Millones de bits por segundo
MFPOPS	Millones de operaciones de punto flotante por segundo.
MHz	Unidad de medida de frecuencia.

MIPS	Millones de operaciones de punto fijo por segundo
mP	Microprocesador
nS	Nano segundo, unidad de medida de tiempo.
PAL	Arreglo lógico programable.
PC	Abreviatura de computadora personal.
PCI	Interfase para conexión de periféricos.
PWM	Modulación de ancho de pulso.
RAM	Memoria de acceso aleatorio.
RCA	Conector standard para entrada y salida de audio.
SHARC	Arquitectura Harvard mejorada.
VLIW	Palabra de instrucción muy larga.

GLOSARIO

Algoritmo Secuencia de pasos a seguir, para la resolución de un problema.

Análisis espectral Análisis que muestra el contenido de frecuencias de una señal.

Arquitectura En un microprocesador, se refiere a aquellos elementos que conforman la unidad central de proceso (CPU), incluye además, la organización de la memoria, el conjunto de instrucciones, y periféricos asociados.

Bit Abreviatura de *dígito binario*, la unidad de almacenamiento más pequeña que existe en una computadora.

Boot-loader Permite cargar programas y datos en un DSP, desde una memoria interna, o desde el puerto serial.

Buffer Dispositivo digital capaz de amplificar corriente, ya sea negando o no, el estado lógico de la entrada a la salida.

Bus	Conjunto de líneas de conexión, conformado por señales para traslado de información, control o direccionamiento en una computadora.
Chip	Empaquetado de dispositivos electrónicos.
Código	Conjunto de instrucciones escritas con la sintaxis adecuada para un programa de computadora.
Compilador	El software que convierte el código fuente en lenguaje que puede ser entendido por un microprocesador
Compilar	Convertir un programa escrito en forma de texto en un lenguaje cualquiera de programación, que la computadora pueda interpretar.
Convolución	Término matemático para designar la operación de multiplicar dos señales.
Decimado	En el desarrollo de un algoritmo para el cálculo de la transformada de Fourier, es el proceso de divisiones sucesivas de una señal.
Demodulación	Realiza la operación inversa a la modulación, para recuperar la señal original.
Depurador	Software que ayuda al programador a localizar errores en su programa

Desbordamiento	Exceder la capacidad de un registro o un espacio de memoria.
Desplazador	Es una operación que mueve un número binario a la izquierda o a la derecha.
Direccionamiento	Se refiere a la manera como se localizan los operandos (ver operandos) durante la ejecución de un programa.
Directivas	Instrucción que se inserta en el código fuente de un programa, para indicarle al compilador que modifique el código fuente de una manera u otra antes de proceder a la compilación
DSP	Microprocesador, especializado en el procesamiento digital de señales.
Ensamblador	Lenguaje de programación, tal como el código de máquina, que requiere el que programador escriba sus programas utilizando crípticas instrucciones que la computadora puede interpretar y ejecutar.
Factor twiddle	Es un factor utilizado en el cálculo de la transformada rápida de Fourier
Fasor	Representación de un número complejo utilizando su amplitud y su fase.

FFT	Transformada rápida de Fourier, permite ejecutar eficientemente algoritmos de la transformada de Fourier en un microprocesador o DSP.
Host	Componente de un DSP, que le permite comunicación con otro sistema, como una computadora.
Interfase	Medio para el intercambio de información entre dos sistemas.
Interrupción	En programación se refiere a interrumpir la operación de un microprocesador, para ejecutar otro conjunto de instrucciones.
Kernel	En los DSP, es el conjunto de instrucciones fundamentales para que el DSP pueda comunicarse con otros sistemas.
Lenguaje	En programación, conjunto de términos y reglas gramaticales para escribir instrucciones que puedan ser ejecutadas por una computadora.
Lógica combinacional	Implementación de sistemas digitales, con compuertas lógicas, en las cuales la salida se determina directamente en cualquier momento de la combinación presente de entradas, sin tener en cuenta las entradas anteriores.

Memoria cache	Memoria auxiliar, de rápido acceso, para la ejecución de instrucciones repetitivas.
Microprocesador	Sistema digital, capaz de interpretar códigos de instrucción y realizar tareas de procesamiento de datos, especificados por un programa.
Modulación	Es el proceso que consiste en variar una señal (portadora), de acuerdo con el valor instantáneo de otra señal (moduladora)
Operando	Es el valor sobre el cual se va a ejecutar la operación de un programa, por ejemplo el valor que se va a sumar o restar a un registro o a una posición de memoria.
Periférico	Dispositivo de entrada/salida que a través de un direccionamiento y una secuencia de instrucciones intercambia información.
Pila de memoria	Es una parte de la unidad de memoria accesada por una dirección que siempre se incrementa o decrementa, después del acceso a la memoria.
Punto fijo	Representación de los números enteros, en formato que puede ser entendido por la computadora.
Punto flotante	Representación de los números reales, en formato que puede ser entendido por la computadora.

Radix-2	Método para realizar el cálculo de la transformada rápida de Fourier en un microprocesador.
Registro	Celdas binarias de memoria, utilizadas en los microprocesadores en la ejecución de instrucciones.
Servo control	Se refiere al control mediante microprocesadores, de motores utilizados en aplicaciones electrónicas.
Temporizador	Dispositivo que genera pulsos oscilatorios.
Vinculación	Expresión que se refiere a la operación del compilador, que viene después de la conversión inicial del código fuente del programador a un código de máquina temporal denominado código objeto, y que vincula las distintas secciones y módulos del programa ya compilado entre sí.

RESUMEN

En este trabajo de graduación, se describe la implementación de un analizador de espectro para frecuencias de audio, utilizando un procesador digital de señales (DSP).

En la implementación de este proyecto se involucran tres áreas fundamentales, la matemática asociada al procesamiento digital de señales, el hardware de los dispositivos DSP y los lenguajes de programación necesarios para desarrollar el sistema.

Con relación al área matemática, en el trabajo se describe el algoritmo para el cálculo de la transformada rápida de Fourier (FFT), que es esencial, para el análisis espectral utilizando dispositivos digitales.

También, se describen ampliamente los dispositivos DSP y sus características internas, proporcionando los fundamentos para la selección apropiada de componentes que permiten alcanzar los objetivos de diseño trazados.

Finalmente, se detalla el proceso necesario para la programación tanto del dispositivo DSP, como de una computadora, que realiza las funciones de control y visualización.

El proceso de desarrollo de este sistema no fue fácil. Aunque existe una gran cantidad de documentación, es difícil organizarla y hacerla coherente con los propósitos que se desean alcanzar. Sin embargo, a pesar de las dificultades, el resultado fue satisfactorio, ya que no sólo se logra la implementación del sistema propuesto, sino que también se describe el proceso de diseño, que es de gran utilidad como guía para la implementación de otros sistemas basados en dispositivos DSP.

OBJETIVOS

General

Implementar un analizador de espectro para frecuencias de audio, basado en DSP para el cálculo de los componentes espectrales de una señal, con una interfase para PC.

Específicos

1. Fijar los criterios necesarios para la selección de un DSP adecuado para la implementación del proyecto propuesto.

2. Desarrollar los programas necesarios para implementar un analizador de espectro en el dispositivo DSP seleccionado.

3. Desarrollar un programa para PC, que despliegue el espectro de la señal procesada por el DSP, permita controlar los parámetros de funcionamiento, y los controles del analizador de espectro.

INTRODUCCIÓN

El procesamiento digital de señales, es una de las tecnologías más importantes que encontramos en los inicios de este siglo XXI, y su aplicación se extiende a una infinidad de productos y dispositivos, en áreas como las comunicaciones, radar y sonar, instrumentación médica, reproducción de música de alta fidelidad, robótica para mencionar algunas.

Una característica de las técnicas de procesamiento digital de señales, es que tienen un alto grado de procesamiento matemático, específicamente multiplicaciones y adiciones, por lo que para implementar dispositivos que puedan procesar señales digitales, se necesitan microprocesadores optimizados para realizar miles de multiplicaciones y adiciones en el menor tiempo posible. Existen muchos fabricantes, que han desarrollado microprocesadores especializados para este tipo de tareas, y son conocidos como Procesadores Digitales de señal (DSP por sus siglas en inglés).

Los fabricantes de DSP, tienen muchas herramientas para que los interesados puedan conocer y familiarizarse con sus procesadores. Una de estas herramientas, es un sistema de evaluación, que incluye el hardware y software necesario para experimentar con el DSP.

El elemento central del hardware es un DSP. El software consiste en herramientas necesarias para compilar programas, cargarlos al DSP y comprobar el funcionamiento del dispositivo en tiempo real, y con software desarrollado por el usuario, estas tarjetas incluyen un puerto paralelo o serial para comunicarse con una PC, que sirve de interfase para el usuario.

En este trabajo de graduación, se utiliza un sistema de evaluación como el mencionado en el párrafo anterior, para la implementación de un analizador de espectro de frecuencias de audio (frecuencias en el rango de 0-20 Khz).

El trabajo consta de seis capítulos descritos brevemente en los siguientes párrafos.

En el capítulo I, se detallan los fundamentos teóricos del análisis del contenido de frecuencia de las señales. El objetivo principal, es llegar a la definición de un algoritmo para el cálculo de la FFT, y demostrar como este algoritmo, permite ejecutar eficientemente el cálculo de los componentes de frecuencia de una señal y la relación que esto tiene con su implementación en un sistema digital.

El capítulo II, esta dedicado a los DSP, explicando qué son y cómo están diseñados y construidos. Además, se detalla el tipo de datos que pueden manejar, los lenguajes de programación y herramientas de software disponibles para programarlos.

Ya se mencionó que en la implementación del analizador de espectro se utiliza un sistema de evaluación. En el capítulo III, se detallan los criterios que se deben tomar en cuenta durante el proceso de selección de componentes de hardware y software de dicho sistema, como su aplicación en la selección del sistema utilizado para desarrollar este trabajo de graduación.

En el capítulo IV, se describen los componentes de hardware y herramientas de software del sistema seleccionado para la implementación del analizador de espectro. El hardware incluye el DSP y un chip utilizado para la adquisición de señales. Las herramientas de software comprenden las utilizadas para la creación de programas que pueden ser ejecutados en el DSP y los programas necesarios para la comunicación con una PC.

El proceso de programación del software que es ejecutado en el DSP, se describe en el capítulo V, este es esencialmente la implementación de un algoritmo que calcula una FFT de 256 puntos en tiempo real.

En el capítulo VI, se describen los pasos necesarios para la programación de la aplicación para la PC, que permite el control del sistema y proporciona la interfase para que el usuario pueda interactuar con el sistema y visualizar la pantalla del analizador de espectro.

Los resultados obtenidos son valiosos, ya que describen de manera organizada, el proceso necesario para la implementación de un sistema digital basado en dispositivos DSP y se demuestran con la implementación práctica, del analizador de espectro para frecuencias de audio.

1. EL DOMINIO DE LA FRECUENCIA

1.1 Introducción

En el procesamiento digital de señales, la información contenida en el dominio de la frecuencia es de mucha utilidad. Por ejemplo, si se conoce la respuesta en frecuencia de un canal de comunicación, es posible determinar las frecuencias que pueden ser enviadas sin distorsión apreciable. El análisis espectral es otro ejemplo de la utilidad del dominio de la frecuencia. El espectro de frecuencias de una señal, proporciona valiosa información, en campos como el reconocimiento de voz, audio, astronomía y muchas otras áreas de la ciencia y la tecnología, que se benefician directamente, del conocimiento de los componentes espectrales de una señal.

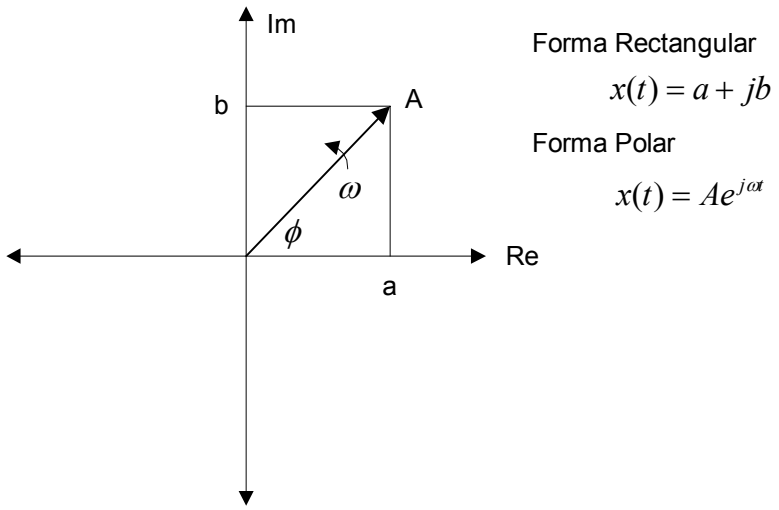
A continuación, se describen, las técnicas fundamentales para el análisis de señales en el dominio de la frecuencia, se inicia definiendo un modelo fasorial para representar señales, basándose en este modelo, se definen las series y transformada de Fourier para señales continuas, de las cuales se deriva el equivalente para señales digitales. Finalmente, se discute un algoritmo muy eficiente para el cálculo de la transformada discreta de Fourier.

1.2 El modelo fasorial

El modelo fasorial, es conveniente para modelar señales. Un fasor es un vector rotando en el plano complejo con una magnitud A y una velocidad de rotación de ω radianes/seg.

Como se muestra en la Figura 1, en cualquier punto, la señal puede ser representada por $x(t) = a + jb$ donde a es el valor real e la señal, y b es el valor complejo.

Figura 1. El modelo Fasorial.



De la notación rectangular, se deriva una forma más útil de representar la señal, en notación polar dada por, $x(t) = Ae^{j(\omega t)}$, donde $A = \sqrt{a^2 + b^2}$ y $\omega(t) = \phi = \tan^{-1} b/a$. Esta ecuación permite determinar la posición del fasor en cualquier instante de tiempo, en función de la frecuencia.

1.2.1 El modelo fasorial discreto

El modelo fasorial explicado anteriormente, se utiliza para representar señales que varían continuamente en el tiempo. En el procesamiento digital de señales, se deben representar señales discretas. Para poder utilizar el modelo fasorial, con señales discretas, es necesario definir dos nuevas variables, T_S que es el periodo de muestreo, y n que representa el número de muestras de la señal que se va a analizar. Con estas dos nuevas variables, se obtiene el modelo fasorial para señales discretas

$$x(n) = Ae^{j(n\omega T_S + \alpha)} \tag{1.1}$$

Donde: T_S = Periodo de muestreo

n = número de muestras

α = desplazamiento de fase

1.2.2 Modelado de señales senoidales

Con las ecuaciones recientemente definidas, es posible modelar señales senoidales, esto es de utilidad cuando se analizan señales en el dominio de la frecuencia utilizando técnicas como el análisis en series de fourier, que se describe más adelante. A continuación se demuestra como modelar señales senoidales.

Por la identidad de Euler, $e^{j\omega t}$ se puede expresar como

$$e^{j(\phi)} = \cos(\phi) + j\sin(\phi) \quad \text{y} \quad e^{-j(\phi)} = \cos(\phi) - j\sin(\phi) \tag{1.2}$$

donde ϕ = ($(\omega t + \alpha)$ ó $(n\omega T_S + \alpha)$ (1.3)

Aplicando esta identidad se puede expresar el seno y el coseno, de la siguiente manera:

$$\sin\phi = \frac{e^{j(\phi)} - e^{-j(\phi)}}{2j} \qquad \text{y} \qquad \cos\phi = \frac{e^{j(\phi)} + e^{-j(\phi)}}{2} \tag{1.4}$$

Con las ecuaciones (1.4), se puede representar cualquier señal senoidal, en términos fasoriales, por ejemplo la siguiente ecuación:

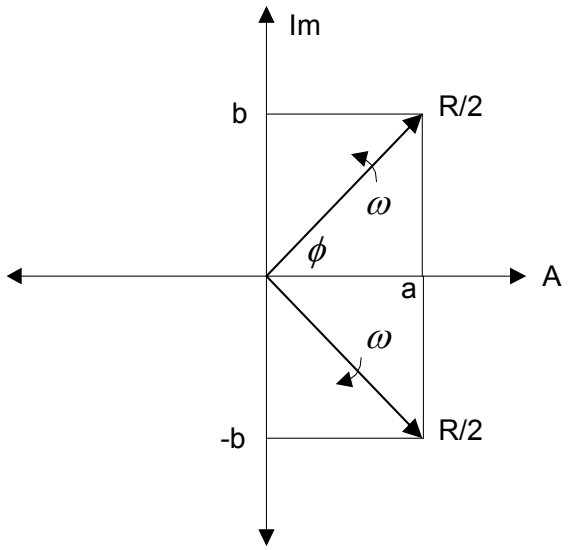
$$x(t) = A\cos(\omega t + \alpha) \tag{1.5}$$

se puede representar de la siguiente manera:

$$x(t) = \frac{A}{2}\left(e^{(j\omega t+\alpha)} + e^{-j(\omega t+\alpha)}\right) \tag{1.6}$$

La señal proporcionada por la ecuación 1.5, se puede representar por medio de dos fasores de igual magnitud rotando a la misma velocidad pero en dirección opuesta, esto se conoce como par conjugado y se muestra gráficamente en la figura 2.

Figura 2. Representación fasorial de $\cos\phi$



1.3 Series de Fourier

En 1782, el matemático francés, Jean Baptiste Joseph Fourier, demostró que si una señal periódica satisface algunas condiciones de carácter general, esta señal, puede ser representada por una sumatoria infinita de senos y cosenos. Esta sumatoria se conoce como series de Fourier.

Las series de Fourier son herramientas útiles para descomponer una señal periódica en componentes más simples. Específicamente una señal periódica con período T_p, puede ser expandida en una serie trigonométrica de senos y cosenos, si cumple con los siguientes requisitos:

1. $x(t)$ tiene un número finito de máximos y mínimos en T
2. $x(t)$ tiene un número finito de discontinuidades en T
3. Es necesario que $\int_0^T |x(t)|dt < \infty$

Si estas condiciones se cumplen, la serie de Fourier puede ser expresada, utilizando la notación fasorial descrita anteriormente, de la siguiente manera

$$x(t) = \sum_{k=-\infty}^{\infty} C_k e^{j(k\omega_0 t)} \tag{1.7}$$

Donde $$C_k = \frac{1}{T_p} \int_{-T_p/2}^{T_p/2} x(t) e^{-j(k\omega_0 t)} dt \tag{1.8}$$

1.4 Transformada de Fourier

Las series de Fourier son de utilidad para representar señales periódicas. Sin embargo, en el procesamiento digital de señales, normalmente se trabaja con señales que no son periódicas, por lo que se necesita un método para representar este tipo de señales. Esto se logra haciendo que un segmento de la señal que se desea representar, tenga un periodo infinito, al hacer esto se obtiene lo siguiente:

$$\text{Si } T_p \to \infty \text{ entonces } \frac{1}{T_p} = \frac{\omega}{2\pi} \to \frac{d\omega}{2\pi} \tag{1.9}$$

Además, la frecuencia variable se convierte en continua, $k\omega_0 \to \omega$ y el coeficiente discreto C_k también se convierte en continuo y esta dado por

$$C(\omega) = \frac{d\omega}{2\pi} \int_{-\infty}^{+\infty} x(t) e^{-j\omega t} dt \tag{1.10}$$

La ecuación (1.10) se puede normalizar dividiendo ambos lados por $d\omega / 2\pi$, al hacer esto, se llega al siguiente resultado:

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt \tag{1.11}$$

Esta ecuación, expresa la frecuencia ω en función del tiempo, substituyendo $X(\omega)$ en la ecuación (1-7), se obtiene:

$$x(t) = \frac{1}{2\pi} \int X(\omega) e^{j\omega t} d\omega \tag{1.12}$$

Las ecuaciones (1-11) y (1-12), permiten la conversión entre el dominio del tiempo y el dominio de la frecuencia. Estas ecuaciones se conocen como par de transformadas de Fourier.

1.5 Trasformada discreta de Fourier (DFT)

Las ecuaciones (1-11) y (1-12), únicamente se aplican a señales contínuas, para poder utilizar estas ecuaciones en un sistema de procesamiento digital, es necesario obtener un equivalente para señales discretas. Este se obtiene al sustituir la variable contínua t con las dos variables discretas nT_S , donde T_S es el período de muestreo y n el número de muestras tomadas. Al hacer esto, se obtiene lo siguiente.

$$x(\omega) = \sum_{n=-\infty}^{\infty} x(n) e^{-j\omega T_s n} \tag{1-13}$$

La integral de la ecuación (1-11), cambia debido a que se trata de valores discretos.

La ecuación discreta equivalente a la ecuación 1-12, es la siguiente

$$x(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\omega) e^{j(\omega T_S n)} d(\omega T_S) \qquad (1\text{-}14)$$

En este caso, la integral se mantiene, debido a que $X(\omega)$ es una función contínua. En tanto que los límites ahora son de $-\pi$ a π, porque el espectro de frecuencia se repite a sí mismo a intervalos de 2π.

La DFT, tal como se presenta en la ecuación (1-13) presenta dos problemas para ser implementada con señales reales, en primer lugar, no es posible calcular una sumatoria con un número infinito de entradas, por lo que es necesario reducirlas. En segundo lugar, se debe reducir el número de frecuencias que se van a calcular, ya que el tiempo del que se dispone para el cálculo es finito.

El primer problema, se resuelve fácilmente tomando una sección de los valores de entrada muestreados, esto se conoce como ventaneo. Para resolver el segundo problema, se debe recordar que, el espectro de frecuencia es repetitivo con respecto a ω_S, por lo que puede deducirse cuantas muestras *N* son requeridas para representar adecuadamente la señal. El número de frecuencias (fasores) que es necesario calcular, generalmente es el mismo que el número de muestras de entrada *N*.

Representando la diferencia entre fasores por la constante δ, se obtiene:

$$N\delta = \omega_S \quad \text{ó} \quad \delta = \omega_S / N$$

Con esta ecuación, es posible digitalizar la frecuencia de la DFT, de tal forma que el espectro se obtiene en términos de k en lugar de ω ya que $k\delta = \omega$:

$$X(k\delta) = \sum_{n=0}^{N-1} x(n) e^{-jk\delta T_S n} \tag{1-15}$$

De ecuaciones previas se sabe que:

$\omega_S = \dfrac{2\pi}{T_S}$ Además $\delta = \dfrac{\omega_S}{N}$ Por lo que se puede rescribir $\delta T_S = \dfrac{\omega_S}{N}\dfrac{2\pi}{N} = \dfrac{2\pi}{N}$

Substituyendo esta relación en la ecuación (1-15) se obtiene

$$X_n(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi kn}{N}} \tag{1-16}$$

El término $e^{-j\frac{2\pi}{N}}$, conocido como factor *twiddle*, generalmente se abrevia por W_N incorporando este factor, en la ecuación (1-16) resulta:

$$X_N(k) = \sum_{n=0}^{N-1} x(n) W_n^{kn} \tag{1-17}$$

La ecuación (1-17), permite obtener los componentes de frecuencia de una señal discreta, utilizando para ello un sistema digital. Sin embargo, el cálculo de esta ecuación, requiere una gran cantidad de multiplicaciones y adiciones. Como ejemplo, se muestra el desarrollo de una DFT de 8 puntos.

$$X_N(k) = \sum_{n=0}^{7} X_n(k) W_7^{kn} \text{ , para } k = 0, 1, 2,...,7 \qquad (1\text{-}18)$$

Expandiendo la serie, para cada valor de n, se obtiene

$$X_n(k) = x(0)W_7^{k0} + x(1)W_7^{k1} + ... + x(7)W_7^{k7} \qquad (1\text{-}19)$$

Cada uno de los 8 términos en la ecuación (1-19) consiste de una multiplicación compleja que debe sumarse a los términos restantes, esto implica 8 multiplicaciones y 7 sumas complejas. Además, cada uno de los términos requiere la expansión de 8 armónicas, ya que n toma valores de 0 a 7. Por lo tanto, para calcular el resultado final se necesitan 8 x 8 multiplicaciones complejas y 8 x 7 adiciones complejas.

Este resultado se puede generalizar, diciendo que una DFT de *N* puntos, requiere N^2 multiplicaciones complejas y $N(N-1)$ sumas complejas.

Para una DFT de 8 puntos, el número de multiplicaciones y sumas no es crítico. Sin embargo para una DFT de 1024 puntos que si tiene utilidad práctica, se requiere aproximadamente 1 millón de multiplicaciones y 1 millón de sumas complejas. Aun para los potentes sistemas de computación de la actualidad, este número de cálculos necesarios, es demasiado para la implementación de una aplicación en tiempo real.

1.6 Transformada rápida de Fourier (FFT)

Como se menciono en los párrafos anteriores, referentes a la DFT, el número de cálculos se vuelve excesivo a medida que el número de muestras *N* aumenta. La transformada rápida de Fourier (FFT), es un método que permite calcular de manera eficiente la DFT, reduciendo en gran medida la cantidad de cálculos necesarios.

La DFT es ineficiente, porque no aprovecha la simetría y periodicidad del factor *twiddle* W_N. Los algoritmos para el cálculo de la FFT, se valen de estas propiedades para mejorar la eficiencia en el cálculo.

La propiedad de simetría afirma que:

$$W_N^{k+N/2} = -W_N^k \tag{1-20}$$

En tanto que la propiedad de periodicidad dice que:

$$W_N^{k+N} = W_N^K \tag{1-21}$$

Dado que W_N es una función periódica, con un número limitado de valores diferentes, reduciendo el número de veces que este factor tiene que ser calculado, se puede acelerar el cálculo de la transformada. Una posibilidad es dividir la transformada en dos series, una consistente de secuencias pares y otra de impares, como se muestra a continuación:

$$X_N(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_N^{k(2r)} + \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_N^{k(2r+1)} \tag{1-22}$$

Se puede incrementar la cantidad de términos idénticos, manipulando los factores *twiddle* de la siguiente manera

$$W_N^{k(2r+1)} = W_N^{k(2r)} x W_N^k \tag{1-23}$$

Dado que W_N^k no depende del término r, se puede extraer de la sumatoria, con lo que se obtiene la siguiente ecuación:

$$X_N(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_N^{2rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_N^{2rk} \tag{1-24}$$

Manipulando nuevamente los factores *twiddle*, se llega a la ecuación de la FFT:

$$X_N(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_{N/2}^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_{N/2}^{rk} \tag{1-25}$$

El calculo de la ecuación (1-25), produce el mismo resultado que el cálculo directo de la DFT, sin embargo es más rápido. Cada una de las sumatorias, requieren $(N/2)^2$ multiplicaciones, y W_N^K se multiplica a los términos impares, por lo que se requieren $(N/2)^2 + (N/2)^2 + N$ multiplicaciones en total. En el caso específico de una DFT de 8 puntos, se requieren 36 multiplicaciones complejas, contra 64 que se requieren para un cálculo directo de la DFT.

Para un *N* de bajo valor, el ahorro de cálculos no es tan evidente, sin embargo para una DFT de 1024 puntos, únicamente se requieren 50,500 multiplicaciones, opuesto a más de 1,000,000 requerido en un cálculo directo.

Con estos ahorros en el cálculo es posible implementar la transformada de Fourier en tiempo real, con sistemas digitales.

A continuación se explica uno de los métodos para calcular la FFT, conocido como *radix*-2 (raiz 2).

1.6.1 Decimación en el tiempo

El proceso de dividir la DFT en dos, es conocido como división en el tiempo, porque la división se realiza en el dominio del tiempo. En el ejemplo previo, se dividió la serie en dos, una sola vez, sin embargo, no hay razón por la que no pueda seguirse dividiendo. Uno de los algoritmos más populares para el cálculo de la FFT, se conoce como *radix*-2 en este algoritmo, la señal se divide en dos, las señales resultantes se continúan dividiendo en dos, hasta que toda la serie queda dividida en DFT de 2 puntos.

Las matemáticas de un algoritmo radix-2 son más simples que las de la DFT, ya que únicamente se deben calcular DFTs de 2 puntos. Sin embargo, los factores *twiddle* producidos, son diferentes en cada etapa de la división. La meta de cualquier implementación práctica de la FFT es incorporar estos factores extras en las matemáticas, sin añadir complejidad computacional al algoritmo.

Para ilustrar mejor como trabaja el algoritmo *radix*-2, se utiliza como ejemplo, una FFT de 4 puntos, a pesar de ser un ejemplo simple, los mismos principios se aplican a todas las FFT calculadas con el algoritmo *radix*-2.

Se inicia con una DFT de 4 puntos, que se define de la siguiente manera:

$$X_4(k) = \sum_{0}^{3} x(n) W_4^{kn} \tag{1-26}$$

Decimando en el tiempo produce dos series de n=(0,2) y (1,3). Esto produce la ecuación:

$$X_4(k) = \sum_{r=0}^{1} x(2r) W_2^{rk} + W_4^k \sum_{r=0}^{1} x(2r+1) W_2^{rk} \tag{1-27}$$

Al expandir las dos sumatorias se obtiene:

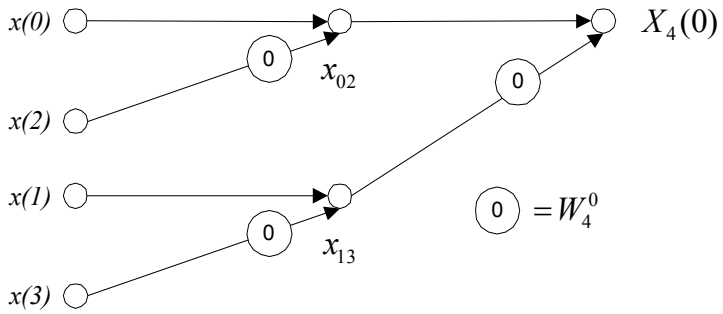
$$X_4(k) = \left[x(0) + x(2) W_2^k\right] + W_4^k \left[x(1) + x(3) W_2^k\right] \tag{1-28}$$

Se tienen dos factores *twiddle*, W_2^k y W_4^k. Sin embargo, ambos pueden ser relacionados como $W_2^k = W_4^k$. Por lo tanto, la ecuación (1-28) se convierte en:

$$X_4(k) = \left[x(0) + x(2) W_4^{2k}\right] + W_4^k \left[x(1) + x(3) W_4^{2k}\right] \tag{1-29}$$

Esta expansión no es muy complicada para una FFT de 4 puntos, sin embargo, al incrementar el número de puntos, también se incrementa la complejidad. Para expresar de mejor manera una FFT con muchos puntos, generalmente la expansión se expresa por medio de un diagrama de flujo, el cual se muestra a continuación.

Figura 3. Diagrama de Flujo de una FFT de 4 puntos para *k* = 0



El diagrama de flujo de la figura 3, muestra la expansión de la ecuación 1-29 cuando *k*=0. el número de círculos representa las potencias de W_4 requeridas en cada etapa. Cuando no hay ninguna multiplicación requerida, la línea no tiene factor multiplicador. Las entradas del diagrama de flujo están en el dominio del tiempo, y la salida a la derecha, en el domino de la frecuencia.

De la figura 3 puede verse que

$$X_4(0) = x_{02} + W_4^0(x_{13})$$

donde

$$x_{02} = x(0) + W_4^0 x(2)$$

y

$$x_{13} = x(1) + W_4^0 x(3)$$

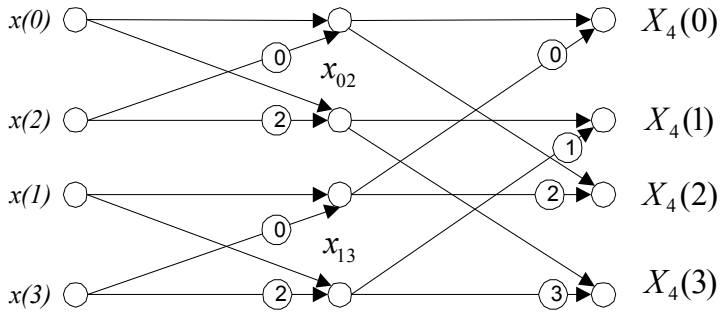
Esto resulta en

$$X_4(0) = x(0) + W_4^0 x(2) + W_4^0 x(1) + W_4^0 x(3)$$

Que es la expansión de la ecuación 1-29, para *k*=0.

En la figura 4, puede verse la expansión completa, para una FFT de 4 puntos. En esta figura, puede apreciarse la simetría de la expansión, la cual se extiende para cualquier tamaño. Esta simetría es conocida como mariposa.

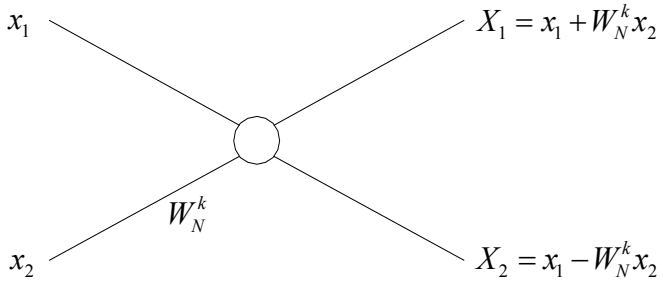
Figura 4. Diagrama de Flujo completo para una FFT de 4 puntos



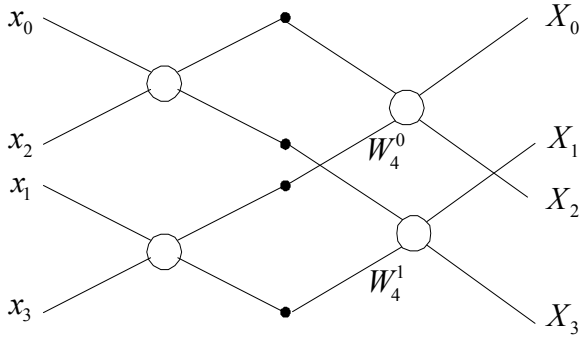
1.6.2 La mariposa

Una forma más típica de implementar la mariposa es como se muestra en la figura 5. Esta consiste de dos entradas, dos salidas y un multiplicador opcional. En esta misma figura, también se muestra el diagrama completo para una FFT de 4 puntos.

Figura 5. a)Mariposa típica. b) FFT de 4 puntos.



a) Mariposa Típica



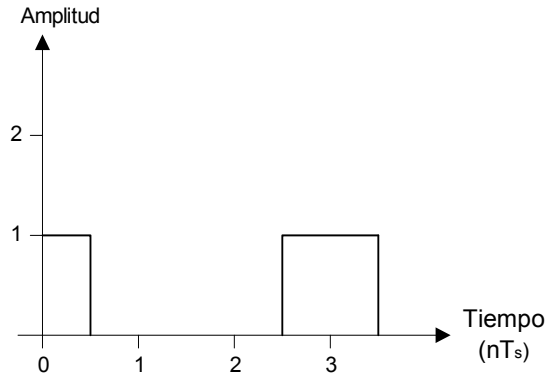
b) FFT de 4 puntos

1.7 Análisis espectral utilizando la FFT

A continuación se muestra con un ejemplo sencillo, como obtener el espectro de frecuencia de una señal, utilizando una FFT de 4 puntos. A pesar de ser un ejemplo que no tiene aplicación práctica, permite ilustrar el proceso de conversión.

Se tiene la siguiente señal en el dominio del tiempo

Figura 6. Señal en el dominio del tiempo.



Se muestrea la señal a 10 KHz, ya que $T=1/f$ esto implica tomar una muestra cada 1/10 KHz = 100 μs. Como se esta utilizando una FFT de 4 puntos, para realizar la conversión, se toman 4 muestras sucesivas de la entrada, obteniendo los siguientes valores de entrada para la FFT:

$$x_0 = 1 \quad x_1 = 0 \quad x_2 = 0 \quad x_3 = 1$$

Con estos valores, se calcula la FFT, como se muestra a continuación

$$X_0 = x_0 + x_2 + x_1 + x_3 = 1 + 0 + 0 + 1 = 2$$
$$X_1 = x_0 - x_2 - j(x_1 - x_3) = 1 - 0 - j(0 - 1) = 1 + j$$
$$X_2 = (x_0 + x_2) - (x_1 + x_3) = (1 + 0) - (0 + 1) = 0$$
$$X_3 = (x_0 - x_2) + j(x_1 - x_3) = (1 - 0) + j(0 - 1) = 1 - j$$

Al calcular la FFT, se obtienen 4 valores en el dominio de la frecuencia. Estos se pueden relacionar con frecuencias, recordando que al definir la DFT, se hizo la siguiente relación: $X(k) \equiv X(\omega)k$

Donde:

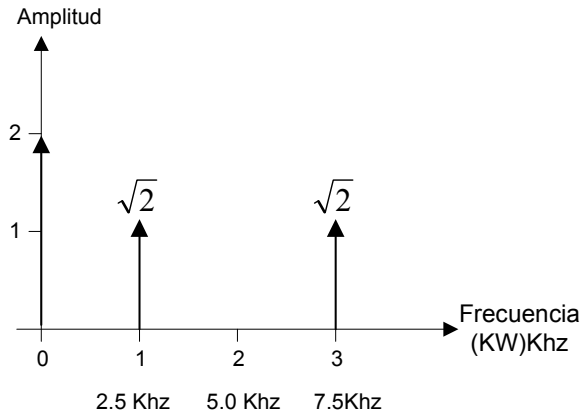
$$\omega = \frac{2\pi}{N} \quad \text{y} \quad f = \frac{\omega}{2\pi}$$

Dado que se esta trabajando con una señal discreta, ω esta dado en radianes por ciclo. Por lo tanto, f representa ciclos por muestra. Ya que se desea calcular frecuencia, se obtiene lo siguiente:

$$X(k) = \left(\frac{2\pi / N}{2\pi}\right)(k) = \left(\frac{1}{N}\right)(k) \ \text{Hz} \qquad (1\text{-}30)$$

De la relación 1-30, puede verse que la frecuencia de salida, se incrementa en múltiplos de ¼ de la frecuencia de muestreo. Por lo tanto, se dispone de las amplitudes de la señal, a las frecuencias de 0, 2.5, 5 y 7.5 Khz. Este resultado, se muestra gráficamente en la siguiente figura.

Figura 7. Espectro de frecuencia para la señal de ejemplo de la figura 6.



Se sabe que el espectro de una FFT de *N* puntos, tiene simetría alrededor de *N*/2, por lo tanto, todos los valores arriba de *N*/2 son simétricos a los valores previos, y pueden ser descartados. De este ejemplo, se observa que el dominio de la frecuencia es simétrico alrededor de *k*=2 y por lo tanto el valor *X*(3) contiene información redundante y debe ser descartado.

1.8 La FFT y los DSP

La estructura simple de la FFT, permite que sea fácilmente implementada en un chip DSP. En los siguientes capítulos, se describen los chips DSP, y la implementación de un algoritmo FFT en uno de estos chips, combinado con un programa de PC, con lo que se obtiene un analizador de espectro de señales de audio.

2. PROCESADORES DIGITALES DE SEÑAL (DSP)

2.1 Introducción

Los procesadores digitales de señal (DSP por sus siglas en inglés), son microprocesadores, cuyo hardware y conjunto de instrucciones están diseñados y optimizados para aplicaciones que ejecutan cálculos matemáticos a alta velocidad.

El desarrollo de los algoritmos para el procesamiento digital de señales, dio origen a la necesidad de contar con un microprocesador especializado para la ejecución de dichos algoritmos. A su vez, el surgimiento del primer DSP en la década de 1980, incrementó el número de dispositivos que incorporan capacidades de procesamiento digital de señales.

Entre las principales tareas de procesamiento digital de señales, que realizan los DSP, se pueden mencionar las siguientes: compresión de audio, filtrado, modulación y demodulación, servo control, procesamiento de audio, señalización en telecomunicaciones, reconocimiento de voz, síntesis de audio.

Estas tareas, requieren la ejecución de cálculos numéricos repetitivos a alta velocidad. Acceso rápido y eficiente a la memoria. Normalmente deben realizar el procesamiento en tiempo real.

Como requisito adicional, los DSP deben minimizar el costo, la potencia consumida, el uso de la memoria, y el tiempo requerido para desarrollar nuevos productos.

En términos financieros, el mayor mercado para los DSP, se encuentra en las aplicaciones como telefonía celular, localizadores y otros sistemas inalámbricos, modems y servo control de discos duros. Estas aplicaciones requieren alto desempeño, bajo costo y mucha eficiencia en el consumo de energía.

A continuación, se describe la arquitectura de los DSP, mencionando las diferencias con respecto a procesadores de propósito general. Se describe además el formato de datos utilizado por los DSP, así como una breve descripción de los principales lenguajes para programarlos.

2.2 Arquitectura de los DSP

La arquitectura de un microprocesador, incluye aquellos elementos que conforman la unidad central de proceso (CPU), tales como registros, la unidad aritmética lógica (ALU) buses internos. Incluye además, la organización de la memoria, el conjunto de instrucciones, las interrupciones, el acceso directo a memoria y los periféricos asociados. Esta información se presenta generalmente en forma de diagramas a bloques.

En el caso específico de un DSP, la arquitectura esta relacionada fundamentalmente con la de los microprocesadores de propósito general, pero tiene diferencias importantes, que responden a la necesidad de ejecutar tareas de procesamiento digital de señales. Estas diferencias se describen en los siguientes párrafos.

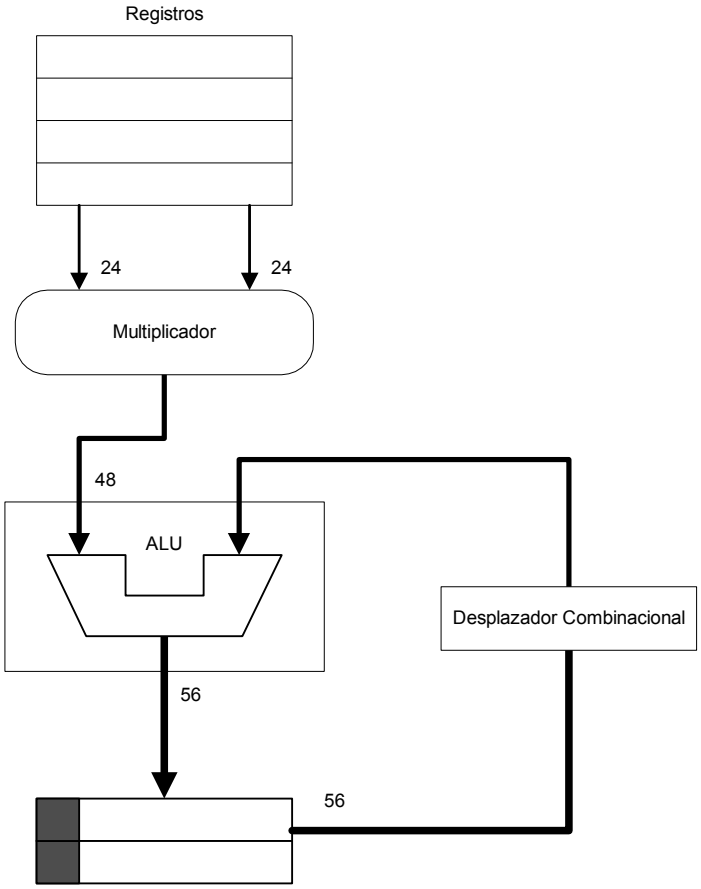
2.2.1 CPU

Con la finalidad de aumentar la velocidad de ejecución de instrucciones, y optimizar la implementación de algoritmos para procesamiento digital de señales, la CPU de los DSP, dispone de unidades computacionales específicas que pueden trabajar en paralelo, los componentes más importantes son, una unidad multiplicadora implementada en Hardware (conocida como unidad MAC), la unidad aritmética lógica (ALU), unidades generadoras de direcciones, un desplazador combinacional y soporte para diferentes tipos de datos.

Una de las principales características de los DSP, es la habilidad de realizar una o más operaciones producto-acumulación (conocida como MAC, por sus siglas en ingles) en un solo ciclo de instrucción. Esta operación se encuentra con mucha frecuencia en algoritmos para la implementación de filtros, correlación y transformadas de Fourier. Para realizar una operación MAC en un solo ciclo, el CPU de un DSP, incluye una unidad multiplicadora y registros acumuladores, para almacenar el producto de varias multiplicaciones. Algunos DSP más recientes, integran una o más unidades MAC, que permiten realizar dos o más operaciones en paralelo.

En los procesadores de propósito general, la operación de multiplicación se lleva a cabo, por medio de operaciones de desplazamiento y suma, lo cual consume varios ciclos de reloj. La adición de hardware para realizar operaciones MAC en un solo ciclo de reloj, es una de las principales diferencias entre los procesadores de propósito general y los DSP.

Figura 8. Unidades computacionales de un DSP típico



Al ejecutar repetidamente operaciones MAC, se puede dar una situación de desbordamiento, lo cual produce resultados erróneos. El desbordamiento se debe a la imposibilidad de representar el resultado de salida, con el número de bits disponibles. Para evitar esta situación, se sobredimensiona el número de bits de la ALU, añadiendo bits extras denominados bits de guarda. De igual manera se sobredimensionan algunos registros del CPU, para que puedan almacenar los valores acumulados.

Algunos DSP que no disponen de estos bits de guarda, suplen esta deficiencia, con la capacidad de desplazar el contenido del registro donde se almacena el producto, antes de sumarlo, sin necesidad de ciclos de instrucción adicionales. Una característica distintiva de lo DSP, es que estos desplazadores, están implementados mediante lógica combinacional, contrario a la estructura clásica de desplazamiento secuencial de otros procesadores. Esto permite realizar desplazamientos en un solo ciclo de instrucción, independiente del número de bits a desplazar.

Una característica adicional, que permite a los DSP aumentar la velocidad de ejecución de operaciones aritméticas, es la disponibilidad de una o más unidades generadores de direcciones de datos. Estas constan de una unidad aritmética específica, que opera en paralelo con el resto de las unidades funcionales y un conjunto de registros que proporcionan la dirección base y el desplazamiento necesario, para el cálculo de la nueva dirección.

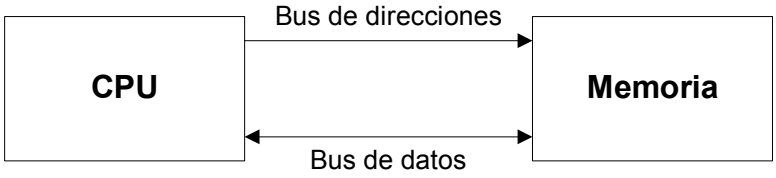
2.2.2 Organización de la memoria

Uno de los mayores cuellos de botella, al ejecutar algoritmos para DSP, es la transferencia de información desde y hacia la memoria. Esto incluye datos e instrucciones de programa. Existen dos modelos básicos en la organización de la memoria, la arquitectura Von Neumman y la arquitectura Harvard, ambas se describen a continuación.

2.2.2.1 Arquitectura Von Neumman

Como se observa en la figura 9, esta arquitectura consiste de una memoria y un bus, para transferir datos desde y hacia la CPU. En esta arquitectura multiplicar dos números requiere al menos tres ciclos de reloj, uno por cada número que se transfiere de la memoria hacia la CPU. Este es el diseño básico de los μP de propósito general.

Figura 9. Arquitectura Von Neumman

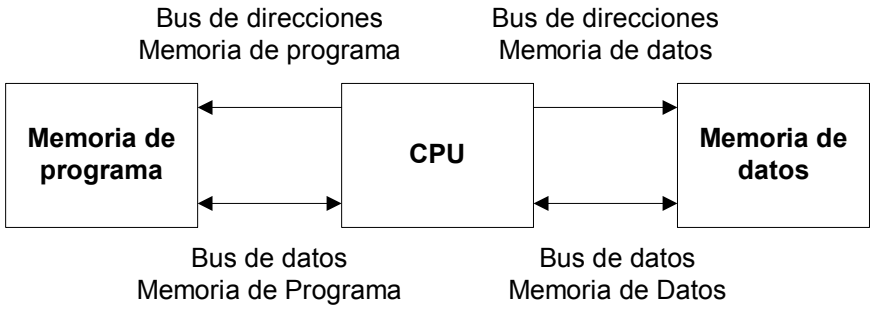


Esta arquitectura es buena para las operaciones que normalmente ejecuta un μP de propósito general, sin embargo no son adecuadas, para la ejecución de algoritmos para DSP, que requieren un ancho de banda mayor. Los DSP incluyen hardware especializado para ejecutar operaciones MAC de un solo ciclo, sin embargo, con este diseño una operación MAC requiere al menos 4 ciclos de reloj.

2.2.2.2 Arquitectura Harvard

Una arquitectura más apropiada para aplicaciones de DSP, es la arquitectura Harvard, nombrada así, por haber sido desarrollada en la Universidad de Harvard, esta arquitectura se muestra en la siguiente figura

Figura 10. Arquitectura Harvard



En esta arquitectura, se dispone de dos memorias independientes, una de programa y otra de datos. Esto permite la transferencia de una instrucción de programa y una de datos al mismo tiempo, lo cual incrementa la velocidad de acceso a la memoria.

La arquitectura Harvard original, dedica un bus para llamar instrucciones y otro bus para llamar operandos. Esto es inadecuado para operaciones de procesamiento digital de señales, que usualmente requieren de al menos 2 operandos. Debido a esto, los DSP que implementan la arquitectura Harvard, usualmente permiten que el bus de programa también pueda llamar operandos. Para mejorar aún más la eficiencia, también se incluye una memoria cache que se utilizara para almacenar instrucciones que serán rehusadas, dejando ambos buses libres para llamar operandos.

2.2.2.3 Arquitectura Von Neumman modificada

En su forma original, la Arquitectura Von Neumman, no permite múltiples accesos simultáneos a memoria. Sin embargo, algunos DSP utilizan una arquitectura Von Neumman modificada, que permite accesos simultáneos a memoria.

Esto se logra utilizando el reloj de memoria a una velocidad mayor que el ciclo de instrucción. En particular, si el reloj de memoria es 4 veces más rápido, cada ciclo de instrucción es dividido en 4 estados y se puede realizar un acceso a la memoria en cada estado, permitiendo un total de cuatro accesos a memoria, por ciclo de instrucción.

La arquitectura Von Neumman modificada, tiene la ventaja de tener un diseño más simple, que produce chips más pequeños, con menos consumo de energía, lo que se traslada en menor costo.

2.2.3 Conjunto de instrucciones

Los DSP se caracterizan por tener hardware especializado para la ejecución de tareas de procesamiento digital de señales. El conjunto de instrucciones debe hacer uso eficiente de este hardware. Lo anterior, resulta en un conjunto de instrucciones altamente especializado, irregular y complejo, que varía mucho entre diferentes DSP.

Normalmente, el conjunto de instrucciones se divide de acuerdo al tipo de operaciones, típicamente, en un DSP, se encuentran las siguientes categorías:

2.2.3.1 Computacionales

Este tipo de instrucciones, ejecutan cálculos con operandos. Por ejemplo operaciones multiplicación/acumulación (MAC), aritméticas y lógicas. Es posible en algunos DSP, ejecutar operaciones hasta con 3 operandos.

2.2.3.2 Carga y almacenamiento

Estas instrucciones, transfieren datos desde y hacia los registros internos de datos y la memoria externa.

2.2.3.3 Control de programa

Instrucciones que se encargan de controlar el flujo de programa. Incluye instrucciones que pueden ejecutar bucles, sin requerir ciclos extras para actualizar el contador del bucle.

2.2.3.4 Multifunción

Toman ventaja del paralelismo inherente del DSP, proveyendo combinaciones de transferencia de datos, lectura o escritura de memoria y ejecución de calculo, en un solo ciclo.

2.2.3.5 Misceláneas

Instrucciones adicionales, para manejo de las pilas de memoria, instrucciones para sincronización con sistemas externos, algunos DSP, también tienen instrucciones especiales, para ahorro de energía.

Otro requerimiento para el conjunto de instrucciones, es que deben minimizar el espacio requerido para almacenar programas. Esto debido a que muchos de los productos que utilizan DSP, son de bajo costo, por lo que tienen poca memoria. Para cumplir este requerimiento, se procura que las instrucciones sean cortas, restringiendo el número de registros que pueden ser utilizados por una instrucción en particular.

Adicionalmente los DSP normalmente tienen menos registros que los procesadores de propósito general.

2.2.4 Modos de direccionamiento

Otra de las características que permiten aumentar la velocidad de procesamiento aritmético en los DSP, es la existencia de unidades dedicadas para la generación de direcciones. Estas controlan la dirección enviada a las memorias de programa y de datos, especificando donde la información será leída o escrita.

Además de algunos modos de direccionamiento de propósito general, los DSP incluyen algunos modos de direccionamiento especializados para el procesamiento de señales.

2.2.4.1 Direccionamiento con autoincremento

Es de utilidad cuando se debe realizar un gran número de cálculos repetitivos, en datos almacenados secuencialmente en memoria.

2.2.4.2 Direccionamiento circular

Para implementar buffer circulares, que son de utilidad en cálculos de convolución y correlación como los utilizados por los filtros digitales.

2.2.4.3 Direccionamiento de bit en reversa

Se utiliza en algoritmos para el cálculo de la FFT.

2.2.5 Periféricos

Los DSP, no solo requieren un diseño favorable para la ejecución de operaciones matemáticas. También deben interconectarse con el mundo real. En una aplicación típica, un DSP debe ser capaz de recibir y transmitir datos en tiempo real, sin interrumpir las operaciones matemáticas internas.

Existen tres fuentes de datos con las que un DSP debe interactuar. Señales de entrada y salida, comunicación con sistemas de control general y comunicación con otros DSP del mismo tipo. Para poder comunicarse efectivamente con estas fuentes de datos, los DSP incluyen puertos especializados. A continuación se describen brevemente los más comunes.

2.2.5.1 Controlador DMA

Es un periférico programable, que permite la transferencia de bloques de datos hacia la memoria del DSP, sin interferir en la operación del CPU. Sirve para interconectar el DSP, con memorias externas y con otros periféricos como el puerto serial y el puerto *host*. La transferencia de datos, se puede sincronizar con interrupciones internas o externas.

2.2.5.2 Puertos seriales

Son puertos síncronos de alta velocidad, diseñados para manejo de señales de audio y telecomunicaciones a velocidades de hasta 10 Mbits/seg. Estos puertos, pueden ser interconectados fácilmente con un convertidor analógico a digital sin requerir lógica adicional. La temporización es flexible, el reloj serial puede ser generado por el DSP o por una fuente externa. También soporta relojes separados para transmisión y recepción.

Adicionalmente, estos puertos trabajan directamente con el controlador DMA, permitiendo comunicación directa con la memoria del DSP, sin interferir en la operación de este último.

2.2.5.3 Puerto *host*

Muchos sistemas de DSP, funcionan controlados por otros sistemas, por eso, generalmente se incluye un puerto *host*, diseñado para comunicación con procesadores de diferente tipo o con un bus estándar, por ejemplo el bus PCI o ISA de las PC. Este puerto, también trabaja con el controlador DMA, por lo que puede comunicarse directamente con la memoria del DSP sin interferir en el funcionamiento de este.

2.2.5.4 Puerto de comunicación

Es un tipo de puerto menos común, que únicamente se encuentra en los DSP diseñados para interactuar con otros DSP del mismo tipo. Físicamente es un híbrido entre puerto serial y paralelo. Internamente acepta palabras de 32 bits, externamente, tiene 4 u 8 bits. También funciona con el controlador DMA.

2.2.5.5 Temporizador

Otro periférico, que normalmente incluyen los DSP, es uno o más temporizadores. Estos se pueden utilizar para señalizar al DSP o al exterior, a intervalos específicos, o como contador de eventos externos. Tienen dos modos de señalización, interno y externo. Con reloj interno puede señalizar un convertidor ADC para iniciar una conversión, o puede generar una interrupción al controlador DMA para iniciar una transferencia de datos.

Con reloj externo, el temporizador puede contar eventos externos, e interrumpir el CPU, después de cierto número determinado de eventos.

2.2.5.6 Puertos especiales

Los DSP diseñados para aplicaciones muy concretas, disponen de periféricos específicos para tales aplicaciones. Los más comunes, son generadores de señal PWM, convertidores ADC y DAC y temporizadores para propósitos especiales.

2.3 Formato de datos

Por el formato utilizado para almacenar y manipular datos Los DSP, pueden ser divididos en dos categorías, de punto fijo y de punto flotante. A continuación se describen ambos.

2.3.1 Datos de punto fijo

Algunos DSP sólo son capaces de operar con números enteros, pero en las aplicaciones es común manejar números fraccionarios. El formato de punto fijo utiliza una representación similar a la de un número entero, salvo que se considera la existencia de un punto binario mediante el cual se escalan los valores enteros para de esta forma, obtener números fraccionarios. Este factor de escala es igual a 2^{-bp} donde bp es la posición del punto binario

El DSP realiza las operaciones de suma o multiplicación como si se tratase de números enteros, sin considerar este factor de escala. Es responsabilidad del programador interpretar la posición del punto binario.

Al desplazar el punto decimal a la izquierda, utilizando más bits para la parte fraccionaria, la precisión aumenta, pero disminuye el margen de valores de la representación. Puesto que el tamaño de la palabra de datos es fijo, la ubicación del punto binario requiere un compromiso, entre la precisión a obtener y el margen de valores a cubrir. Se debe utilizar el mayor número de bits para la parte fraccionaria (máxima precisión) que permiten representar todo el rango de valores que toma una variable.

Si durante el procesamiento un número en coma fija aumenta demasiado para poder ser representado con el número de bits disponibles para la parte entera, el programador debe realizar un escalado descendente del número mediante un desplazamiento a derechas perdiendo los bits de menor peso y por tanto disminuyendo la precisión. Si por el contrario el número en coma fija disminuye demasiado el número de bits utilizados en la parte fraccionaria puede ser insuficiente. El programador debe realizar un desplazamiento a la izquierda para aumentar la precisión.

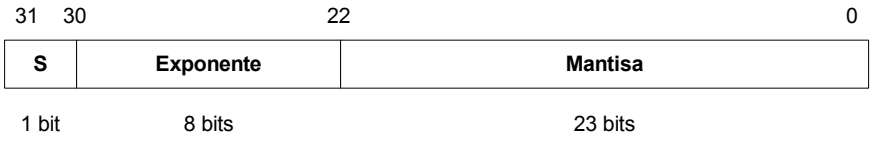
En ambos casos el programador debe tomar en consideración como se ha desplazado el punto binario restaurando todos los números de coma fraccionaria a una misma escala en una etapa posterior. Esto convierte la programación de aplicaciones en una tarea muy tediosa.

Pese a las dificultades en la programación, la mayor parte de los DSP fabricados en la actualidad, utilizan aritmética de punto fijo. Esto se debe principalmente a que tiene un bajo costo, ya que se requiere menos hardware para implementarlos. Además, el consumo de potencia es menor.

2.3.2 Datos de punto flotante

Los DSP que manejan datos de punto flotante, comúnmente representan cada número con un mínimo de 32 bits. Este formatos de 32 bit, esta estandarizado por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) en el estándar 754.1985. En la figura 11, se muestra el ejemplo de un formato numérico de punto flotante de 32 bits.

Figura 11. Representación de punto flotante



F= Número de punto flotante = $(-1)^S$ X M X $2^{(E-127)}$
Donde:
S=0 Para números positivos
S=1 Para números negativos
M= Mantisa formada por una fracción binaria de 23 bits
E= Número entre 0 y 255 representado por los 8 bits del exponente

Todos los DSP de punto flotante, también pueden manejar números de punto fijo, necesario para implementar contadores, lazos, y señales desde y hacia los convertidores ADC y DAC. El desempeño en el manejo de números de punto fijo suele ser más lento aunque existen DSP optimizados para el manejo de ambos formatos de datos.

Los DSP de puntos flotante tienen mejor precisión, y un rango dinámico (el radio entre el número más pequeño y más grande que puede ser representado) más elevado. Adicionalmente, la programación es más fácil, ya que el programador no debe preocuparse por errores debidos a saturación, redondeo o desbordamiento por sobrecapacidad.

La arquitectura de un DSP de punto flotante, es más complicada que la de uno de punto fijo. Los registros y buses de datos son de 32 bits. El multiplicador y la ALU deben tener alto desempeño para ejecutar aritmética de punto flotante. El conjunto de instrucciones es más grande, ya que debe incluir instrucciones para manejo de ambos formatos de datos. Todo esto, se traduce en un costo elevado y mayor consumo de potencia.

2.4 Programación de los DSP

Los DSP se programan en los mismos lenguajes utilizados en otras áreas de la ingeniería y la ciencia, los más usuales son Ensamblador ó C. En términos generales, los programas escritos en ensamblador se ejecutan más rápido, en tanto que los programas en C son más fáciles de escribir y mantener.

2.4.1 Programación en lenguaje ensamblador.

La forma más directa de escribir un programa para un microprocesador, es a través de unos y ceros, esto es conocido como código de máquina. Para los humanos es imposible escribir un programa con este código. El lenguaje ensamblador, que es un lenguaje de bajo nivel, que traslada el código de máquina a instrucciones nombradas según la operación que se ejecuta.

La programación en ensamblador, implica la manipulación directa de la electrónica digital: registros, localidades de memoria, bits de estatus, etc. Por ello, este lenguaje requiere un profundo conocimiento de la construcción interna del DSP que se esta programando, para poder así optimizar al máximo el código.

Los programas para DSP, se diferencian de las tareas tradicionales del software en dos aspectos importantes. El primero es que los programas usualmente son más cortos. El segundo, es que la velocidad de ejecución normalmente es un factor crítico de la aplicación. Estos dos factores, han propiciado un mayor uso de lenguaje ensamblador.

La principal desventaja del lenguaje ensamblador, es el tiempo requerido para escribir un programa. Además, el conjunto de instrucciones varía entre uno y otro fabricante, e incluso entre la línea de productos de un fabricante en particular.

En términos generales, programar en lenguaje ensamblador, es lo ideal cuando la velocidad de ejecución es crítica, y cuando el costo del producto final se desea mantener lo más reducido posible.

2.4.2 Programación en Lenguaje C

Los DSP, también pueden ser programados, utilizando lenguajes de alto nivel, en estos, se manejan variables abstractas, sin hacer referencia al hardware en particular. De los lenguajes de alto nivel, el más utilizado es C, aunque es posible hacerlo con otros lenguajes como Pascal o Basic, en la práctica es muy raramente usado.

Al programar con un lenguaje de alto nivel, un programa llamado compilador transforma el código de alto nivel, en código de máquina. El lenguaje de alto nivel aísla al programador del hardware. Esto hace que la programación sea más fácil, y que el código pueda ser transportado, entre diferentes tipos de DSP. El programador no necesita saber nada acerca de la construcción interna del DSP, ya que el compilador, hace este trabajo por él.

Sin embargo programar en C, tiene algunas desventajas, usualmente se requiere más memoria que en ensamblador, esto incrementa el costo del producto, además, por no tenerse control directo sobre el hardware, la velocidad de ejecución no está optimizada.

Pese a sus desventajas, C es la mejor opción cuando el tiempo de desarrollo es crítico, cuando los programas son muy extensos o complejos, y en casos en los que la programación será desarrollada por un grupo de personas.

También existe la posibilidad de escribir programas en C, y optimizar los segmentos críticos en velocidad, con ensamblador.

3. SELECCIÓN DE COMPONENTES DE HARDWARE

3.1 Introducción

Un analizador de espectro, es un dispositivo que despliega en forma gráfica, el contenido de frecuencias de una señal.

En el presente trabajo de graduación, se implementa un analizador de espectro, utilizando un sistema digital basado en DSP, para la conversión de una señal del dominio del tiempo, al dominio de la frecuencia y también un programa para PC que permita desplegar la información de manera gráfica y controlar el dispositivo.

Para la implementación del analizador de espectro, se requiere un sistema digital, que incluya todo el hardware necesario. En este capítulo se detalla el proceso de selección del sistema de hardware requerido, iniciando con las especificaciones y requerimientos mínimos del sistema, las opciones evaluadas y los criterios de selección aplicados.

3.2 Especificaciones del sistema

El primer paso, en el diseño de un sistema basado en DSP, es determinar las prestaciones del sistema, basado en las tareas que debe realizar.

Esto incluye los algoritmos de procesamiento digital que debe ejecutar, el tipo de señales a procesar y los dispositivos de entrada y salida a interconectar. El tipo de algoritmo a implementar determina la cantidad de memoria de programa requerida, en tanto que el tipo de señales a procesar, determina la cantidad de memoria de datos necesaria.

Los principales fabricantes de chips DSP, disponen de diferentes herramientas de hardware para la evaluación de sus dispositivos. Una de estas herramientas, es una tarjeta que incluye un sistema basado en DSP, con el hardware necesario para interactuar con señales analógicas y el software necesario para programar el dispositivo.

Para la implementación del analizador de espectro, se utilizara uno de estos sistemas de evaluación. Por ello será necesario considerar las características de hardware y software requeridas del sistema, para hacer una selección apropiada.

3.2.1. Algoritmo a ejecutar

Como ya se menciono, el objetivo del analizador de espectro, es convertir una señal analógica del dominio del tiempo, al dominio de la frecuencia. Para esto, se deben utilizar el análisis de Fourier, como se vio en el capítulo uno de este trabajo de graduación, la transformada de Fourier apropiada para ser implementada en un sistema digital, es la DFT. Para calcular la DFT, el algoritmo más eficiente es la FFT que será el utilizado en este proyecto.

Entre mayor sea la cantidad de muestras disponibles para el cálculo de la FFT, se obtiene mejor resolución de frecuencia, y por lo tanto una mejor identificación de las frecuencias presentes en una señal dada. Sin embargo, al incrementar el número de muestras, se incrementan los requerimientos del sistema, ya que se requiere más memoria interna y mayor potencia de cálculo. Dadas las limitaciones previstas en los sistemas de evaluación, se estima apropiado el cálculo de una FFT de 256 puntos.

3.2.2 Memoria interna

La memoria interna que incluya el DSP que se utilice, es importante, ya que de ello depende la eficiencia en la ejecución del algoritmo. Para implementar una FFT de 256 puntos, se requieren aproximadamente 1 Kb de palabras de memoria, 256 para almacenar la señal original, 512 palabras para almacenamiento en tiempo de ejecución, y 256 palabras para memoria de programa.

3.2.3 Sistema de adquisición de señales

Otro elemento importante, en el desarrollo del proyecto, es el sistema de adquisición de señales, específicamente la resolución del convertidor analógico a digital y la frecuencia de muestreo, que determina el rango de señales que pueden analizarse. Para poder muestrear apropiadamente señales de audio, se necesita una frecuencia de muestreo de 44 Khz.

3.2.4 Comunicación con PC

Para el despliegue gráfico del resultado de la FFT y también para controlar el dispositivo, se utilizara una PC, por lo que un requerimiento sobre el DSP que se utilizará, es que debe tener un puerto *host* para comunicación con la PC, este requerimiento no presenta inconvenientes, ya que generalmente, los sistemas de evaluación incluyen DSP con este tipo de puertos, y se comunican con la PC, a través del puerto serial, o del puerto paralelo.

3.2.5 Formato de datos

Como se vio en el capítulo dos, una forma de clasificar los DSP, es por el formato de datos, si es de punto flotante, o de punto fijo. Se ha decidido utilizar formato de punto flotante, por la precisión numérica que se obtiene y que es requerido en la implementación de una FFT, además de facilitar la programación.

3.2.6 Herramientas para desarrollo de software

Otro aspecto a considerar en el desarrollo de un sistema basado en DSP, es la cantidad y disponibilidad de herramientas para desarrollo de software. En este aspecto, como mínimo se requiere de un ensamblador para convertir código fuente en un archivo que pueda ser ejecutado en el DSP. También un depurador para verificar el funcionamiento apropiado del sistema y corregir los errores que pudieran presentarse en la programación.

3.2.7 Requerimientos adicionales

Además de los requerimientos anteriormente mencionados, existen otros aspectos que se deben considerar. El costo del sistema es importante, ya que de ello dependerá la viabilidad de implementar sistemas de este tipo. Otro aspecto que no se debe descuidar, es la velocidad de operación del DSP, porque esto determina que el sistema pueda operar en tiempo real. Este último aspecto no es tan crítico en el sistema propuesto, ya que la mayoría de DSPs disponibles en el mercado, pueden ejecutar una FFT de 256 puntos en tiempo real.

3.3 Principales fabricantes de chips dsp

En el anteproyecto del trabajo de graduación, se propuso la investigación de las principales familias de productos DSP, producidos por los mayores fabricantes de dispositivos DSP. En la actualidad, las empresas que dominan el mercado de chips DSP, son Texas Instruments™, Analog Devices™, Motorola™ y Lucent™.

Para el desarrollo de este capítulo, se investigaron estos fabricantes principales, buscando la descripción y detalles de los chips DSP que producen.

Motorola™ es uno de los principales fabricantes, sin embargo, sus productos están diseñados para mercados específicos, como la telefonía celular, servocontrol de discos duros y otras aplicaciones para electrónica de consumo.

Motorola™ Tienen disponible herramientas de evaluación, pero los precios son bastante elevados, ya que consiste de sistemas muy elaborados y destinados a empresas que se dedican a desarrollar productos, basados en estos chips. Adicionalmente, no produce dispositivos DSP de 32 bits, ya que sus políticas de diseño, están enfocadas a productos de 24 y 16 bits.

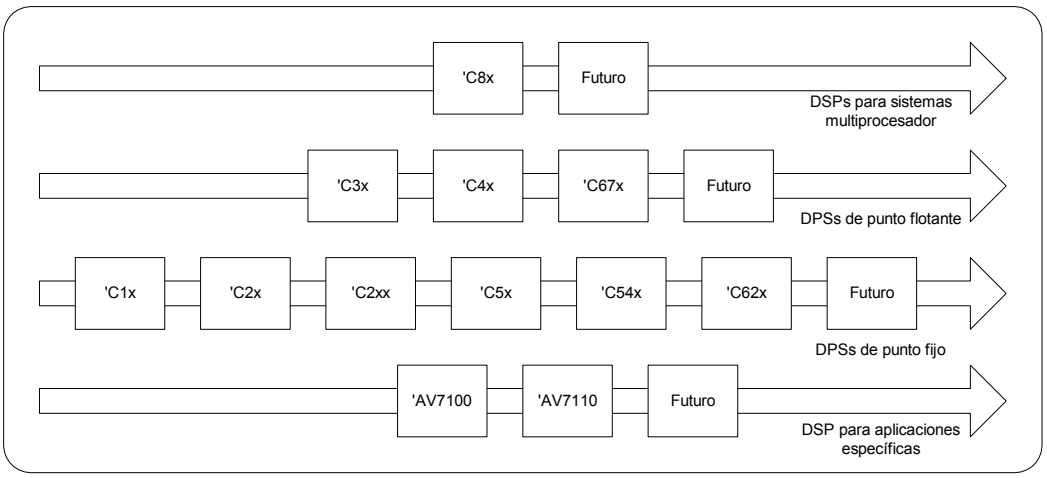
En el caso de Lucent™, no fue posible obtener información de los dispositivos que producen. En la actualidad. Lucent™ esta desarrollando junto a Motorola™, el núcleo para DSP denominado *StarCore*, sin embargo, esto no es aplicable para el presente proyecto.

Por lo anteriormente expuesto, las dos opciones consideradas para la implementación del proyecto, son los chips fabricados por Texas Instruments™ y Analog Devices™ Estas empresas en la actualidad, tienen el liderazgo en la producción de estos dispositivos, contando con una amplia gama de productos y herramientas para desarrollo. A continuación se describen las principales familias de productos producidos por estos dos fabricantes.

3.3.1 Texas Instruments™

Se considera a Texas Instruments™, como la empresa que produjo el primer chip DSP, el TMS32010 que fue introducido en el año 1982. En la actualidad, la empresa aún lidera el mercado de chips DSP y cuenta con un amplio portafolio de productos, tanto de punto fijo, como de punto flotante, con una gran variedad de opciones de memoria interna, periféricos asociados y velocidades de operación. En la figura 12, se presentan las principales familias de chips DSP producidos por esta empresa.

Figura 12. Principales familias de chips DSP producidos por Texas Instruments™



Fuente: TMS320 DSP Product Overview. Pag. 5

Como puede observarse en la figura 12, existen dispositivos de punto fijo, de punto flotante, dispositivos para sistemas multiprocesador, y algunos para aplicaciones específicas. Por los requerimientos de diseño, se concentrará la atención en los chips de punto flotante, que de acuerdo con el diagrama anterior son los siguientes, TMS320C3x, TMS320C4x, TMS320C67x

3.3.1.1 TMS320C3x

La familia TMS320C3x es la primera generación de dispositivos de punto flotante de 32 bits fabricados por Texas Instruments™. Los dispositivos 'C3x proveen una arquitectura de alto desempeño, combinada con la facilidad de uso. Esto permite el rápido desarrollo de productos. Estos dispositivos, se utilizan en una amplia variedad de áreas, incluyendo aplicaciones para automóviles, audio digital, automatización industrial y control, comunicaciones de datos y equipos para oficina, como periféricos multifunción, copiadoras e impresoras láser.

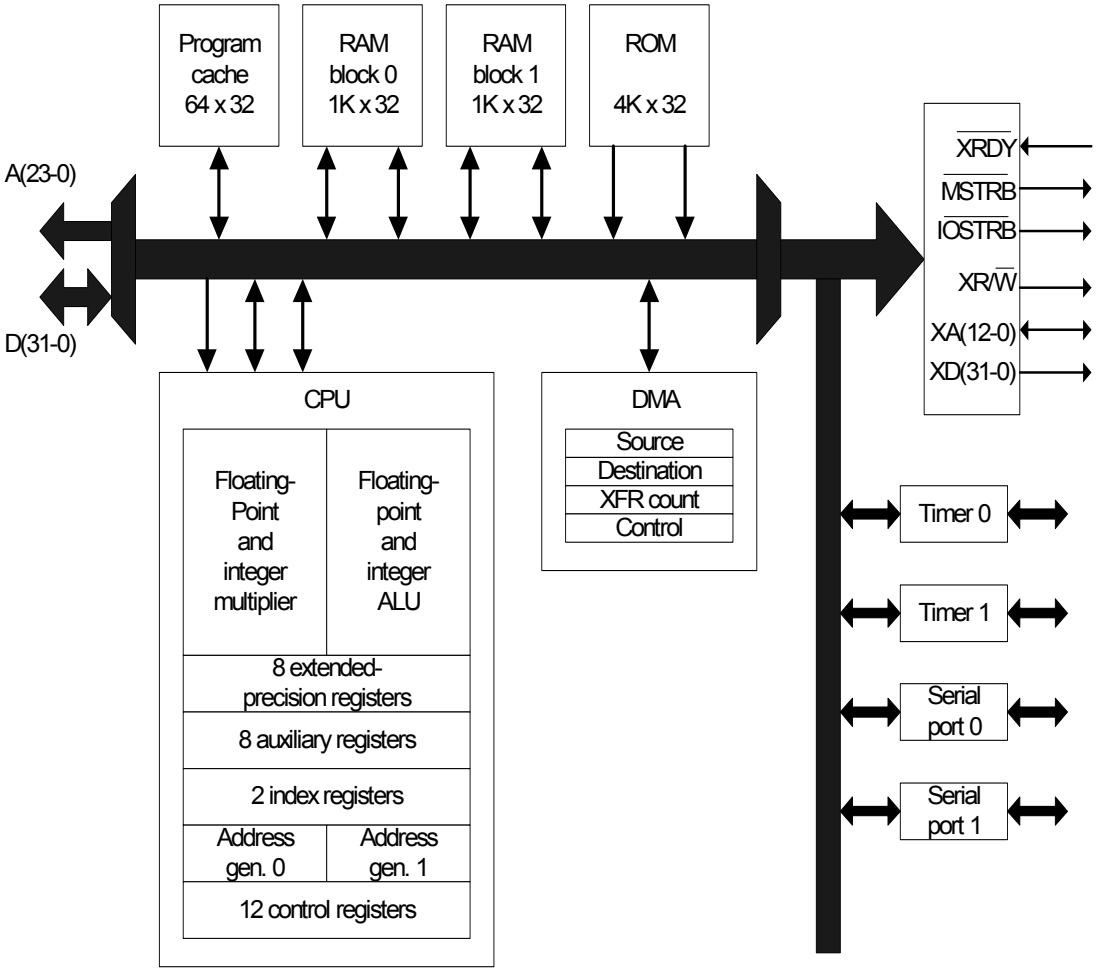
El CPU tiene un multiplicador y un acumulador independientes, y puede ejecutar hasta 60 MFPOPS (Millones de operaciones de punto flotante por segundo) y hasta 30 MIPS (Millones de operaciones enteras por segundo).

Dispone de un controlador de acceso a memoria (DMA), el cual tiene su propio bus de datos y opera en paralelo con el CPU. El DMA se programa para entrada y salida de bloques de datos, liberando el CPU de algunas operaciones aritméticas. Además, este controlador puede acceder cualquier ubicación del mapa de memoria, incluyendo memoria interna, externa y registros mapeados en memoria.

El espacio de memoria total, del 'C3x es de 16 millones de palabras de 32 bits. La memoria de datos, de programa y de E/S esta contenido en este espacio de memoria, lo que permite flexibilidad en la distribución de la memoria.

A continuación se muestra un diagrama a bloques de un dispositivo 'C3x típico. Las diferencias, entre los miembros de esta familia, están determinadas, por el número de periféricos asociados y la cantidad de memoria interna, en la tabla I, se presenta la relación entre los diferentes miembros de esta familia.

Figura 13. Diagrama a bloques del TMS320C30



Fuente: TMS320 DSP Product Overview, Pag. 45

Tabla I. Integrantes de la familia TMS320C3x

Device	Cycle Time (ns)	Memory On Chip		Memory Off Chip	Peripherals		
		RAM Words	ROM Words		On-Chip Timer	Serial Port	DMA
'C30	60	2K	4K	16M	2	2	1
'C30-40	50	2K	4K	16M	2	2	1
'C30-50	40	2K	4K	16M	2	2	1
'C31-40	50	2K	Boot	16M	2	1	1
'C31-50	40	2K	Boot	16M	2	1	1
'C31-60	33	2K	Boot	16M	2	1	1
'C32-40	50	512	Boot	16M	2	1	2
'C32-50	40	512	Boot	16M	2	1	2
'C32-60	33	512	Boot	16M	2	1	2

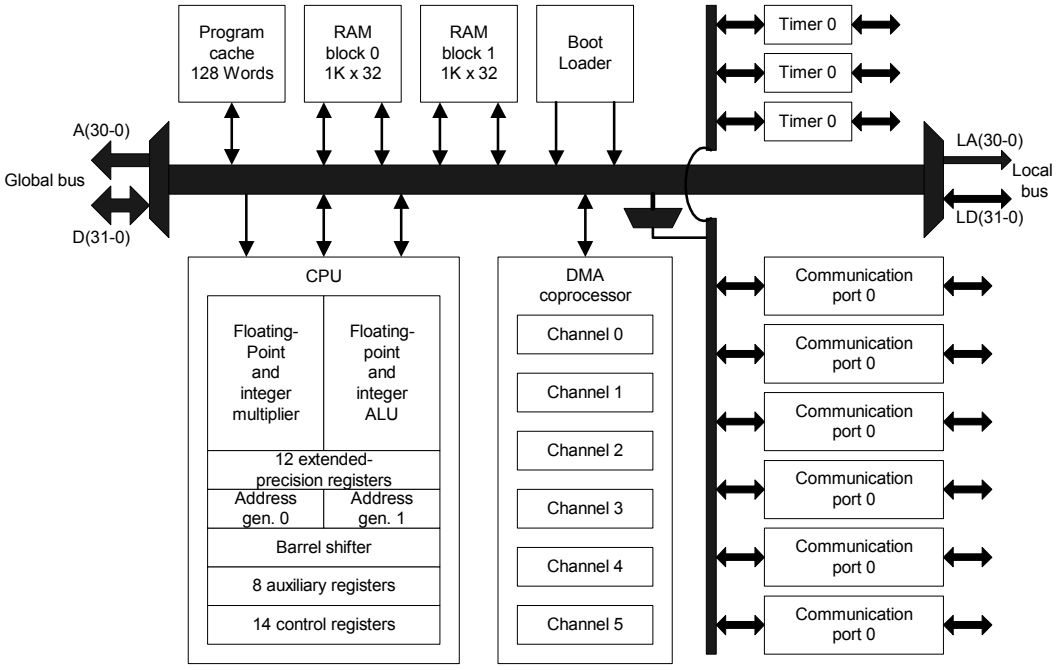
Fuente: TMS320 DSP Product Overview, Pag. 59

3.3.1.2 TMS320C4x

Los dispositivos TMS320C4x, son DSP de punto flotante de 32 bits, optimizados para procesamiento en paralelo. La familia 'C4x combina un CPU de alto desempeño y un controlador DMA con hasta 6 puertos de comunicación, para satisfacer las demandas de sistemas multiprocesadores. Las aplicaciones típicas para el C4x incluyen, gráficas tridimensionales, procesamiento de imágenes, dispositivos para redes y telecomunicaciones.

En la figura 14, se muestra un diagrama a bloques del 'C40 y en la tabla II se presentan las diferencias, entre los miembros de la familia TMS3204x

Figura 14. Diagrama a bloques del TMS320C40



Fuente: TMS320 DSP Product Overview, Pag. 63.

Tabla II. Diferentes miembros de la familia TMS320C4x

		Memory On-Chip			Peripherals			
Device	Cycle Time (ns)	RAM (Words)	ROM (Words)	Memory Off-Chip	Parallel Bus	Comm Ports	DMA Coprocessor	On-Chip Timer
‘C40-50	40	2Kx32	Boot	4Gx32	2	6	6/12 Chann.	2
‘C40-60	33	2Kx32	Boot	4Gx32	2	6	6/12 Chann.	2
‘C44-50	40	2Kx32	Boot	32Mx32	2	4	6/12 Chann.	2
‘C44-60	33	2Kx 32	Boot	32Mx32	2	4	6/12 Chann	2

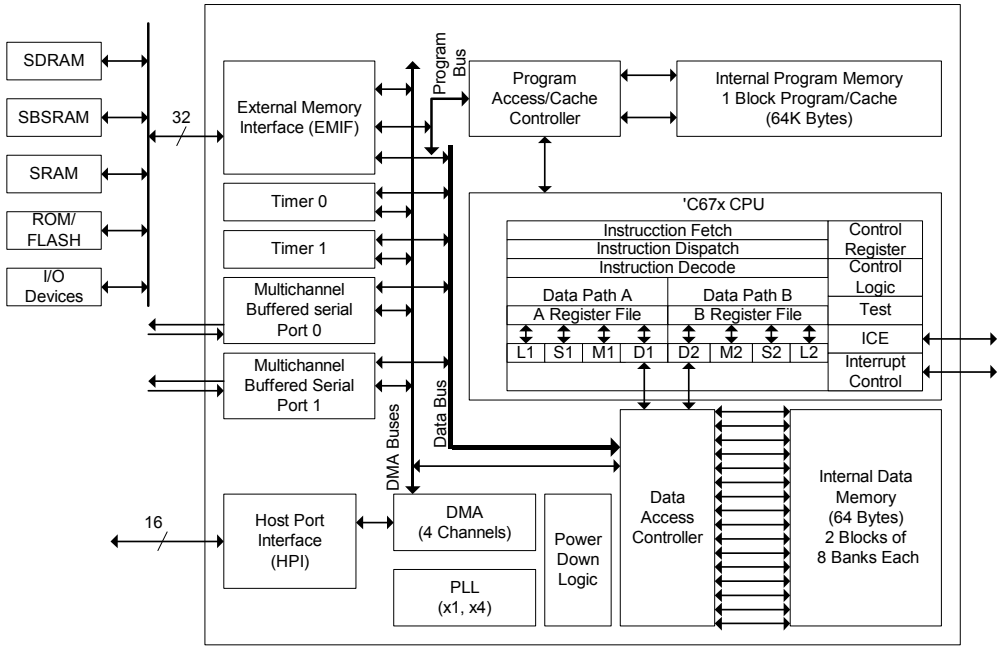
Fuente: TMS320 DSP Product Overview Pag. 75

3.3.1.3 TMS320C67x

Los dispositivos TMS320C6x son los primeros en utilizar la tecnología VelociTI^{TM} , una arquitectura avanzada que utiliza VLIW (Very Large Instruction Word). Esta arquitectura permite desempeños de hasta 1600 MIPS. El TMS320C67x es un DSP de punto flotante capaz de ejecutar 1 GFLOP por segundo. Esta familia es ideal para aplicaciones que requieren un alto desempeño.

Este procesador, tiene 32 registros de propósito general, de 32 bits. Además dispone de ocho unidades funcionales, altamente independientes, que proveen cuatro ALUs de punto flotante/punto fijo. Dos ALUs de punto fijo y dos multiplicadores de punto flotante. A continuación se presenta el diagrama a bloques de este DSP.

Figura 15. Diagrama a bloques del TMS320C6701



Fuente: TMS320C6701 Floating Point DSP, Data Sheet, Pág. 4

Tabla III. Principales integrantes de familia TMS320C67x

Device	Cycle Time	Data/ Program Memory	DMA	External Memory Inteface	Host/ Port	Timers 32 bits
'C6701	7	512K/512K	4	1	1	2
'C6701-150	6.7	512K/512K	4	1	1	2
'C6711-100	10	32K/32K	16 (EDMA)	1	1	2
'C6711B-100	10	32K/32K	16 (EDMA)	1	1	2
'C6712-100	10	4K/4K	16 (EDMA)	1	N/A	2
'C6713-150	6.7	4K/4K	16 (EDMA)	GFN PYP	1	2
'C6713-225	4.4	4K/4K	16 (EDMA)	GFN PYP	1	2

Fuente: http://dspvillage.ti.com

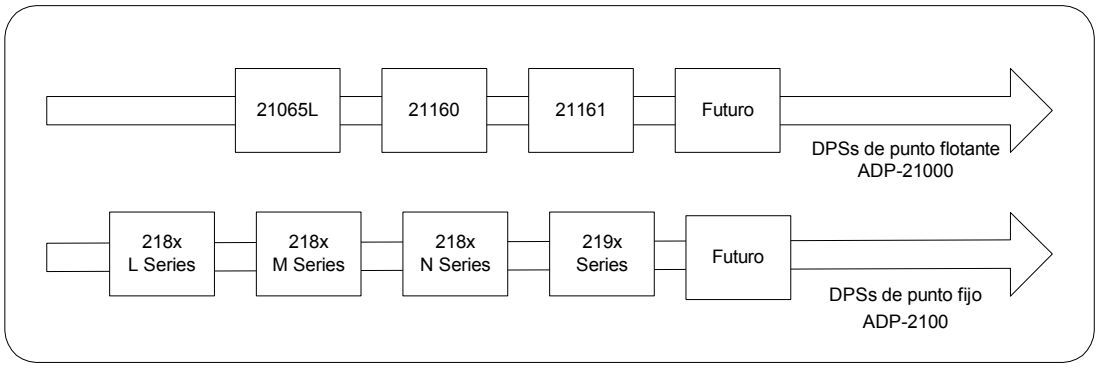
Cualquiera de los DSP examinados previamente, puede ser utilizado para la implementación del analizador de espectro. El 'C4x esta diseñado para sistemas multiprocesador, que no es una característica requerida, en tanto que el 'C6701 esta diseñado para sistemas de alto desempeño, y por lo tanto su precio es más alto, por lo que la opción ideal, que cumple con todos los requisitos es el TMS320C3x.

3.3.2 Analog Devices™

La empresa Analog Devices™, ingreso al mercado de chips DSP en el año de 1984, con la introducción del chip ADSP-2101, un DSP de punto fijo de 16 bits. Desde entonces, la variedad de productos a crecido para incluir además de otros DSP de punto fijo, una completa línea de productos de 32 bits de punto flotante, además de dispositivos para aplicaciones específicas.

El siguiente diagrama, muestra las principales familias de DSP producidas por Analog Devices™.

Figura 16. Principales familias producidas por Analog Devices™



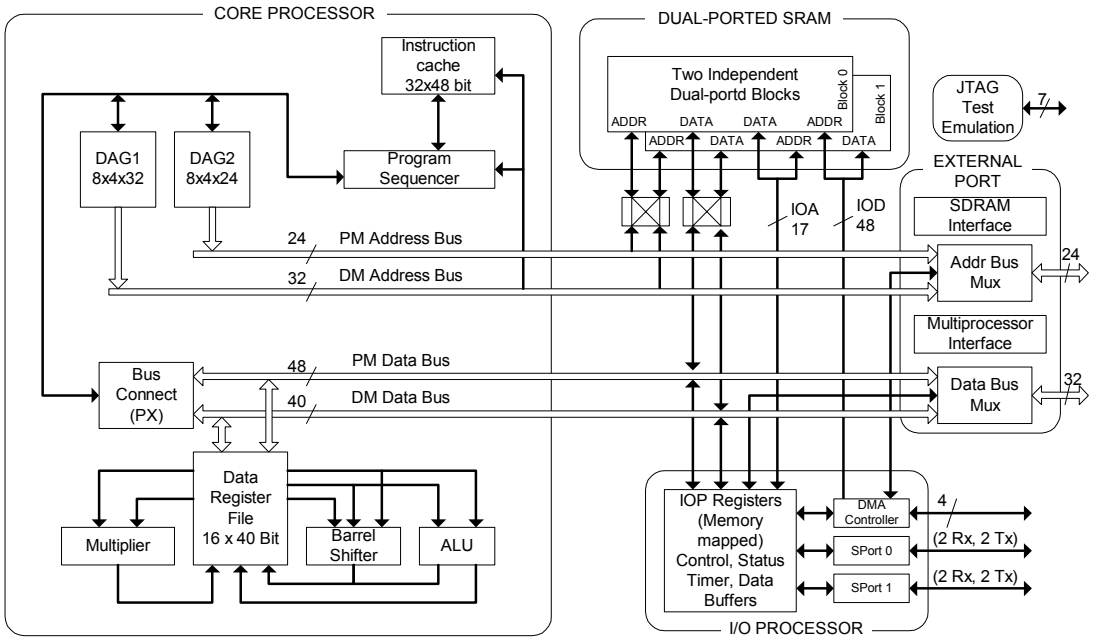
Como en el caso de Texas Instruments™, el interés se centra en seleccionar un DSP de punto flotante. Analog Devices™ recomienda para nuevos diseños, los miembros de la Familia ADSP-21000, los cuales se examinan a continuación.

3.3.2.1 ADSP-21000 SHARC

Los DSP de la familia ADSP-2100, presentan una arquitectura denominada SHARC (Super Harvard Architecture), optimizada para implementar una gran variedad de aplicaciones de procesamiento digital de señales en tiempo real. Estos DSPS de 32 bits, pueden ser programados eficientemente, tanto en punto flotante, como en punto fijo. La amplia variedad de aplicaciones de estos DSP incluye, audio profesional y de consumo, graficas 3D, videojuegos, videoconferencia, reconocimiento de voz e instrumentación.

En la siguiente figura se muestra un diagrama a bloques con uno de los DSP pertenecientes a esta familia, en el diagrama se destacan las diferentes unidades computacionales, y la interrelación entre las mismas. En la tabla IV, se muestran los principales componentes de esta familia, con las diferencias básicas entre las mismas.

Figura 17. Diagrama a bloques del ADSP-21065L



Fuente: Analog Devices™, DSP Microcomputer, ADSP-21065L Data Sheet, pag 1

Como en el caso de Texas Instruments™, cualquier miembro de la familia ADSP-21000 es apropiado para el desarrollo del proyecto, sin embargo la opción económicamente más accesible es el ADSP-21065L.

Tabla IV. Componentes de la familia ADSP-21000

Familia ADSP-21000			Peripherals		
Device	Cycle Time (ns)	Dual-ported SRAM	Serial Ports	Comm Ports	SPI*
21065L	15	544 K	2	-	-
21160	12.5	4 M	2	6	-
21161	10	1 M	4	2	1

Fuente: Analog Devices™ Select Guide

3.4 Sistemas de evaluación

Por lo discutido en la sección anterior, la selección de un sistema con los componentes de hardware necesarios para implementar el proyecto, se reduce a dos opciones. Una basada en el TMS320C3x de Texas Instruments™. La otra en el ADSP-21065L de Analog Devices™. Los dos fabricantes, tienen herramientas de evaluación para estos dos DSPs. A continuación se describen las características principales de ambos.

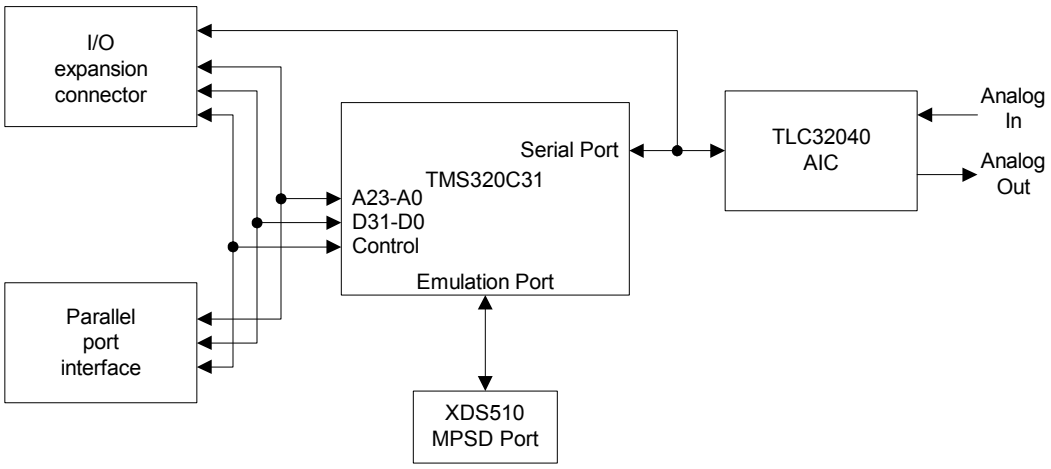
3.4.1 Texas Instruments™ TMS320C3x DSP *Starter Kit* (DSK)

Este *kit*, esta basado en el DSP de punto flotante TMS320C31, con un reloj interno de 50 Mhz. Tiene un ciclo de instrucción de 40 ns y puede ejecutar 50 MFLOPS y 25 MIPS. Cuenta con una interfase para puerto paralelo, que le permite la comunicación con una PC.

La adquisición de datos, es a través del chip TLC32040, el cual es un circuito de interfase analógica (AIC), que consiste de un convertido ADC y un DAC, con una tasa de muestreo variable hasta un máximo de 20000 muestras por segundo y una resolución de 14 bits.

En la figura 18, se muestra un diagrama a bloques del DSK. En dicho diagrama, puede verse la interconexión entre los principales componentes. También cabe destacar, que las salidas del bus de direcciones el bus de datos y de control, así como la salida del puerto serial, pueden ser enrutadas, hacia un conector de expansión, lo cual permite utilizar el DSK, como la base para un proyecto mayor.

Figura 18. Diagrama a bloques del TMS320C3x DSP *Starter Kit*.



Fuente: DSP Starter Kit User's Guide, Pág 1-3.

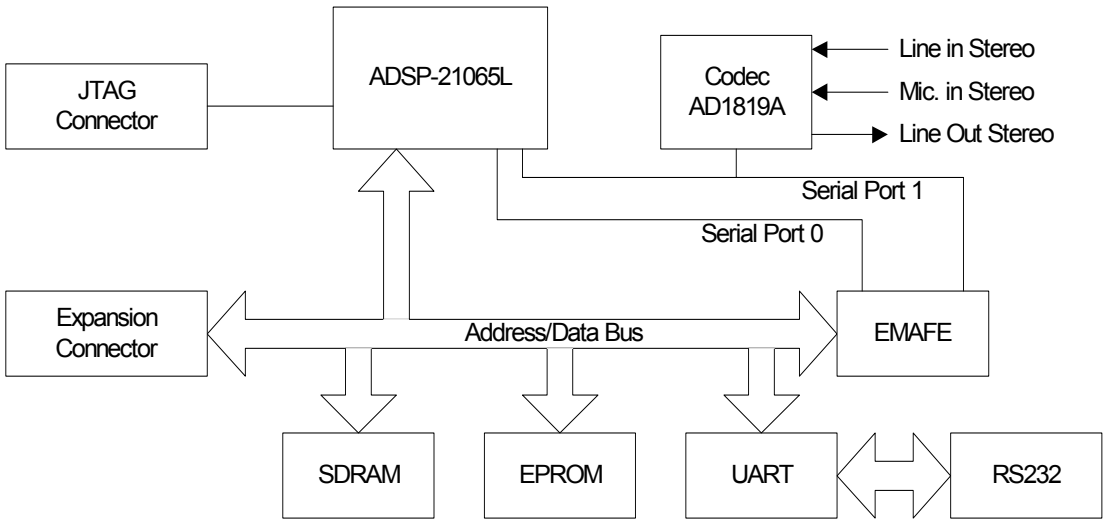
Para desarrollo de software, el DSK incluye, un ensamblador y un depurador. Ambas herramientas son sencillas, pero efectivas para la creación de programas para el DSK.

3.4.2 Analog Devices™ ADSP-21065L *EZ-Kit Lite*.

Este sistema, utiliza un ADSP-21065L, a 60 Mhz, con un ciclo de instrucción de 15 ns, y es capaz de ejecutar 180 MFLOPS y 66 MIPS. Su comunicación con una PC, es a través de una interfase RS-232. Para la conversión de señales analógica a digital y viceversa, el *EZ-Kit Lite*, utiliza un CODEC estereo de 16 bits, con una taza de muestreo variable hasta 48,000 muestras por segundo.

El diagrama a bloques del *EZ-Kit Lite*, se presenta en la figura 19. Puede advertirse en esta figura que este *kit* también cuenta con conectores de expansión. Adicionalmente, dispone de una memoria EPROM de 1 M x 8 bits, la cual puede ser utilizada en aplicaciones independientes de una PC.

Figura 19. Diagrama a bloques del ADSP21065L *EZ-Kit Lite*.



Fuente: ADSP-21065L EZ-Kit Lite Evaluation Manual, Pág 39.

Como herramienta para el desarrollo del software, el *EZ-Kit*, incluye una versión limitada, de una herramienta muy poderosa, el Visual DSP ++. Esta herramienta incluye, un entorno de desarrollo integrado (IDE), que es una interfase gráfica para la administración de proyectos. Un depurador, y herramientas para la generación de código, esto incluye un compilador de C, un ensamblador, y librerías para tiempo de ejecución.

3.5 Criterios Para Selección del Sistema a utilizar

Hasta el momento, se ha realizado una selección preliminar. De este proceso previo de selección, resultaron los sistemas de evaluación que se han descrito en los párrafos precedentes. Los criterios aplicados, se basaron, principalmente en: formato de datos, velocidad de ejecución, herramientas para desarrollo de software y costo relativo del producto.

Los dos sistemas seleccionados pueden ser utilizados para la implementación del proyecto de graduación. Por lo tanto, es necesario considerar factores adicionales, para realizar la selección final de uno de los dos sistemas. Específicamente se considera con mayor detalle el costo aproximado de cada uno de los sistemas y las facilidades proporcionadas para el desarrollo del software.

En la tabla V se presenta un resumen comparativo, de las principales características de los dos sistemas. En dicha tabla, también se enumeran las ventajas y desventajas principales de cada uno de los sistemas.

3.6 Selección del sistema

En base a los resultados presentados en la tabla V, el sistema elegido para la implementación del proyecto, es el TMS320C3x DSP *Starter kit*, (DSK) por las razones que se detallan a continuación

3.6.1 Costo

El costo del sistema, es considerablemente menor, que el de su contraparte, de Analog Devices™. Esto se debe principalmente a que es un sistema con prestaciones más limitadas, que sin embargo, son suficientes para desarrollar el proyecto. El *EZ-Kit Lite*, es un sistema más complejo, con mayor capacidad de procesamiento, pero con muchos componentes extras que no son necesarios para este proyecto y que contribuyen a elevar el precio.

3.6.2 Herramientas de desarrollo

El objetivo principal, de las herramientas de desarrollo, es disminuir el tiempo requerido para desarrollar un producto. El uso de herramientas complejas y avanzadas, se vuelve crítico para empresas que diseñan y producen dispositivos basados en DSP. Para los propósitos de este trabajo de graduación, lo importante es contar con una herramienta sencilla, de fácil aprendizaje, que permita desarrollar el software en el menor tiempo posible.

Por lo expuesto en el párrafo anterior, la mejor opción, es las herramientas incluidas en el DSK. Esto debido a que la versión del ensamblador y el depurador son sencillos y optimizados para el desarrollo de aplicaciones pequeñas.

Las herramientas de desarrollo proporcionadas por Analog Devices™ en el *EZ-Kit*, son más completas, pero también más complejas y no son necesarias para la implementación del proyecto.

Una ventaja a favor del *EZ-Kit*, es el hecho de incluir un compilador de C. Para personas con experiencia en la programación de C, esta herramienta puede facilitar enormemente el desarrollo de programas, sin embargo, en el caso del presente proyecto, no es una característica necesaria, dado que el proyecto no es tan complejo.

3.6.3 Documentación

Un tercer factor, favorable al DSK, es la existencia de abundante documentación que soporta al dispositivo. Esto es debido principalmente a que el TMS320C3x es una familia de dispositivos que tiene bastante tiempo en el mercado, y que ha sido muy popular para aplicaciones de propósito general. En contraposición, el ADSP-21065L, es un dispositivo relativamente reciente, con una marcada tendencia a aplicaciones de procesamiento de audio por lo tanto, la documentación disponible es limitada y generalmente enfocada al desarrollo de aplicaciones de audio.

3.6.4 Hardware

En conjunto, los componentes utilizados por el DSK, son más apropiados para la implementación del sistema propuesto. La mayor limitante de dicho sistema, es el AIC, que únicamente permite analizar señales de hasta 10 Khz. Sin embargo, como se observa en la figura 18, la salida del puerto serial, que se interconecta directamente con el AIC, puede ser enrutada hacia el

conector de expansión, por lo que si se desea, se puede sustituir el AIC, por un componente más avanzado, que permita mayores frecuencias de muestreo.

3.6.5 Conclusión

En la tabla V, se listan las desventajas de cada uno de los sistemas propuestos. Por las razones que se describieron recientemente, las desventajas del DSK pueden ser superadas, o no resultan determinantes. En tanto que las desventajas del *EZ-Kit Lite* son más severas para el desarrollo de este proyecto.

En general, el sistema presentado por Analog Devices™, es superior, en desempeño, en prestaciones, en herramientas disponibles para desarrollo de software, sin embargo, para el desarrollo del proyecto de graduación, la mejor opción, es el DSK, de Texas Instruments™, el cual satisface los requerimientos del proyecto, con menor costo y tiempo de desarrollo.

Tabla V. Comparación entre el DSK de Texas Instruments y el *EZ-kit* de Analog Devices

	DSP	Convertidor ADC	Software para Desarrollo	Costo Aproximado	Documentación Disponible	Ventajas	Desventajas
DSK Texas Instruments	TMS320C31 Clock – 50 Mhz Cycle Time – 40 ns 50 MFLOPS 25 MIPS 2 K Words RAM 64 Word Cache	AIC TLC32040 - Tasa de muestreo Variable, hasta 20,000 Hz - Rango Dinámico 14 bits	Ensamblador (Incluye directivas para vinculación) Depurador	$ 100.00	Manuales de usuario Guías de aplicación Reportes de aplicación Tutoriales Ejemplos	- Cumple con requisitos mínimos al mejor precio. - Abndante documentación disponible. - Facilidad de programación - Disponibilidad de herramientas de terceras partes	- AIC limitado a frecuencias máximas de 10 KHZ. - Software de desarrollo limitado. - Menor capacidad de procesamiento.
EZ-Kit Analog Devices	ADSP-21065L Clock – 60 Mhz Cycle Time – 15 ns 198 MFLOPS 60 MIPS 16 K RAM (Limitado por software a 2.5 K)	AD1819A Full Duplex, Stereo Codec. - Tasa de muestreo variable hasta 48,000 Hz - Rango Dinamico 16 bits	Visual DSP++ limitado, que incluye: IDE Compilador C Ensamblador Vinculador Librerías en tiempo de ejecución.	$ 300.00	Manual de usuario Reportes de aplicaciones (mayormente aplicaciones de procesamiento de audio).	- Mayor capacidad de procesamiento - Herramientas de desarrollo más completas. - Mayor rango de procesamiento (cubre completamente el espectro de frecuencias de audio).	- Costo elevado - Características extras, no necesarias. - Mayor complejidad para desarrollo. - Menor documentación y soporte disponible.

4. COMPONENTES DE HARDWARE Y SOFTWARE DEL DSK

4.1 Introducción

En este capítulo, se describen los principales componentes de hardware, y herramientas para desarrollo de software, que incluye el sistema DSK. Inicialmente, se detalla la arquitectura del DSP TMS320C31, que es el elemento central del sistema. A continuación, se describen las principales características del circuito de interfase analógica (AIC). También se describen las herramientas para el desarrollo de software y el proceso necesario para programar el DSK. Finalmente, se detalla el funcionamiento global del DSK y la manera en que este se comunica con la PC.

4.2 Arquitectura del DSP TMS320C31

El DSP TMS320C31, es un procesador muy rápido, que puede ejecutar 16.7 millones de instrucciones por segundo y tiene un ciclo de instrucción de 60 ns. Además dispone de un amplio espacio de memoria y maneja operaciones aritméticas de punto flotante.

La arquitectura del TMS320C31 esta diseñada para cumplir con los requerimientos de sistemas de procesamiento numérico intensivo, como es el caso de los algoritmos para procesamiento digital de señales.

El TMS320C31 logra la rapidez en el procesamiento de señales al combinar la precisión y rango dinámico de las unidades de punto flotante, memoria integrada en el chip, la capacidad de ejecutar varias instrucciones en paralelo y un controlador DMA. Estas características se ilustraran con mayor detalle en los siguientes párrafos.

Los diferentes componentes de la arquitectura del TMS320C31, se ilustran en la figura 20 y a continuación se explica más detalladamente cada bloque.

4.2.1 CPU

La arquitectura del CPU, esta basada en registros. Consiste en un multiplicador de números enteros y de punto flotante, una unidad aritmética lógica (ALU), un desplazador de 32 bits, buses internos, dos unidades auxiliares aritméticas de registro (ARAUs) y el archivo de registros del CPU.

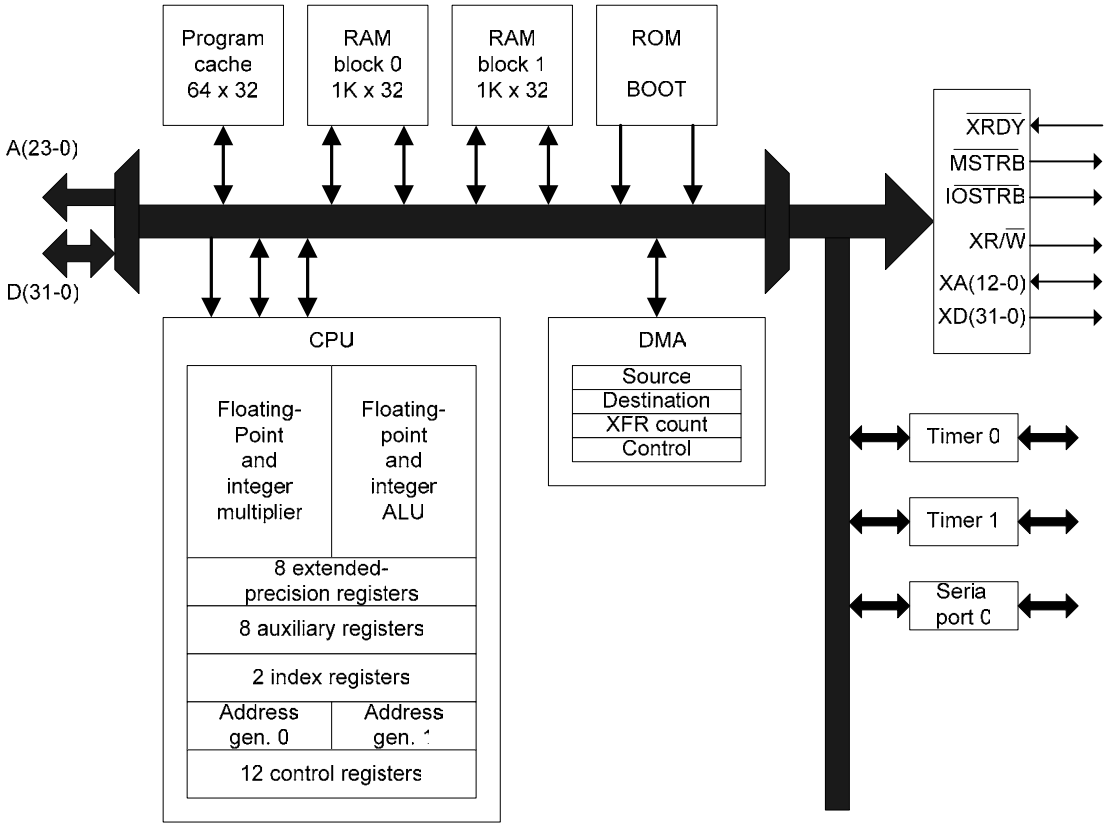
4.2.1.1 Multiplicador de números enteros y de punto flotante:

El multiplicador, ejecuta multiplicaciones de un solo ciclo, en enteros de 24 bits y números de punto flotante de 32 bits. La implementación que hace el 'C3x de la aritmética de punto flotante, permite ejecutar operaciones de punto fijo o punto flotante, a velocidades de hasta 33 ns por ciclo de instrucción. Al ejecutar multiplicaciones de punto flotante, los operandos son números de punto flotante de 32 bits, y el resultado es un número de punto flotante de 40 bits. En multiplicaciones enteras, los operandos tiene 24 bits y el resultado es de 32 bits.

4.2.1.2 Unidad aritmética lógica (ALU)

La ALU ejecuta operaciones en un solo ciclo, sobre enteros de 32 bits, y números de punto flotante de 40 bits. El resultado de las operaciones es mantenido en formato entero de 32 bits o de punto flotante de 40 bits. La ALU también cuenta con un desplazador, este es utilizado para desplazar hasta 32 bits hacia la izquierda o derecha en un solo ciclo.

Figura 20: Arquitectura del TMS320C31



Cuatro buses internos, trasladan dos operandos de la memoria, y dos operandos del archivo de registros. Esto permite ejecutar multiplicaciones en paralelo, además sumas y restas de cuatro operandos enteros o de punto flotante, en un solo ciclo.

4.2.1.3 Unidades auxiliares para aritmética de registros (ARAUs)

El ‘C31 dispone de dos unidades aritméticas para la aritmética de registros (ARAUs), las cuales pueden generar dos direcciones en un solo ciclo. Las ARAUs operan en paralelo con el multiplicador y la ALU. Estas unidades son de mucha utilidad para direccionamientos con desplazamiento o indexados por registro.

4.2.1.4 Archivo primario de registros del CPU

El ‘C31 cuenta con 28 registros en un archivo de registros multipuerto que esta estrechamente acoplado al CPU.

La ALU y el multiplicador, operan sobre los registros primarios y pueden ser utilizados como registros de propósito general. Los registros también tienen algunas funciones especiales. Por ejemplo, los ocho registros de precisión extendida, son apropiados para mantener resultados de punto flotante de precisión extendida. Los ocho registros auxiliares soportan una gran variedad de modos de direccionamiento indirecto y además pueden ser usados como registros de 32 bits de propósito general, o como registros lógicos. Los restantes registros proveen algunas funciones, como direccionamiento, administración de la pila de memoria, información sobre el estado del CPU, interrupciones o repetición de bloques.

4.2.2 Organización de la memoria

El espacio total de memoria del ‘C31 es de 16 Millones de palabras de 32 bits. Esta memoria contiene programas, datos y espacios de E/S, lo que permite el almacenamiento de tablas, coeficientes, código de programa, o datos

tanto en RAM como en ROM. De esta manera, se maximiza el uso de la memoria, y se dispone de flexibilidad para la distribución del espacio de memoria.

4.2.2.1 Memoria RAM y cache

En la figura 20, puede verse la forma como esta organizada la memoria del 'C31. Cada uno de los bloques de RAM contienen 1K de palabras de 32 bits y se pueden accesar dos veces en un solo ciclo. El *boot loader* permite cargar programas y datos, desde memorias de 8, 16 o 32 bits, o desde el puerto serial.

La separación de los buses de datos, programa y DMA permite la ejecución en paralelo de llamadas a programa, lectura y escritura de datos y operaciones DMA. Por ejemplo, el CPU puede acceder a dos datos almacenados en uno de los bloques de RAM y ejecutar una llamada a código en memoria externa, en paralelo con el controlador DMA cargando otro bloque de RAM, todo esto en un solo ciclo.

Además de la memoria RAM, el 'C3x dispone de una memoria cache de instrucciones, la cual es utilizada para almacenar secciones de código que se repiten frecuentemente, lo cual reduce el número de accesos a memoria externa. Esto permite almacenar código en memoria externa, la cual usualmente es más lenta, pero más económica. Una ventaja adicional, es que los buses externos quedan libres y pueden ser utilizados por el DMA, llamadas a memorias externas, o para otros dispositivos en el sistema.

4.2.2.2 Modos de direccionamiento de memoria

El ‘C31, soporta un conjunto de instrucciones de propósito general, así como instrucciones optimizadas para la ejecución de instrucciones de aritmética intensiva, optimizadas para la ejecución de algoritmos para procesamiento digital de señales.

Existen cuatro grupos de direccionamiento en el ‘C3x. Cada grupo usa dos o más de diferentes tipos de direccionamiento. A continuación se listan los modos de direccionamiento con sus tipos de direccionamiento.

- **Modos de direccionamiento general**
  Registro – El operando es un registro del CPU
  Inmediato corto – El operando es un valor inmediato de 16 o 24 bits.
  Directo – El operando es el contenido de una dirección de 24 bits, formada al concatenar un operando de 16 bits con los 8 bits del registro DP.
  Indirecto – Un registro auxiliar, indica la dirección del operando
- **Modos de direccionamiento para instrucciones de 3 operandos**
  Registro – El operando es un registro del CPU
  Indirecto – Un registro auxiliar, indica la dirección del operando.
- **Modos de direccionamiento para instrucciones en paralelo**
  Registro – El operando es un registro de precisión extendida.
  Indirecto – Un registro auxiliar, indica la dirección del operando.
- **Modos de direccionamiento para instrucciones de bifurcación**
  Registro – El operando es un registro del CPU
  PC-relativo – Un desplazamiento de 16 bits con signo, o un desplazamiento de 24 bits es sumado al contenido del registro PC.

4.2.3 Operación del bus interno

Una buena parte de la rapidez del ‘C3x, se logra a través de sus buses internos que operan en paralelo. Buses separados permiten llamadas a programa, accesos a datos y accesos DMA en paralelo. En la figura 20 se muestra la forma en la que los buses conectan los diferentes componentes del ‘C3x.

El contador de programa (PC) esta conectado al bus de direcciones de programa de 24 bits (PADDR). El registro de instrucción (IR) esta conectado al bus de datos de programa (PDATA) de 32 bits. Estos buses pueden llamar una sola palabra de instrucción en cada ciclo de máquina.

Los buses de dirección de datos (DADDR1 y DADDR2) y el bus de datos (DDATA), efectuar dos accesos a memoria en cada ciclo de máquina. El bus DDATA, traslada datos hacia el CPU a través de los buses CPU1 y CPU2. los buses CPU1 y CPU2 pueden trasladar dos operandos de la memoria de datos a el multiplicador, ALU y archivo de registros en un ciclo de máquina. Adicionalmente, los buses internos del CPU (REG1 y REG2), pueden llevar dos datos del archivo de registros al multiplicador y ALU cada ciclo de máquina.

El controlador DMA, se comunica a través de un bus de direcciones de 24 bits (DMAADDR) y un bus de datos de 32 bits (DMADATA). Estos buses, le permiten al DMA, ejecutar accesos a memoria en paralelo con los accesos a memoria que ejecutan los buses de datos y de programa.

4.2.4 Interrupciones

El ‘C31 soporta cuatro interrupciones externas (INT3-INT0), algunas interrupciones internas y una señal externa de *RESET*. Estas interrupciones se utilizan para interrumpir el DMA o el CPU.

Dos banderas externas de E/S (XF0 y XF1) se pueden configurar a través de software, como pines de entrada o salida. Estos pines también son utilizados para operación entrelazada con otros sistemas.

4.2.5 Periféricos

Todos los periféricos del ‘C31 son controlados a través de registros mapeados en memoria, en un bus dedicado. El bus de periféricos, esta compuesto de un bus de datos de 32 bits y un bus de direcciones de 24 bits. Los periféricos que tiene el ‘C31 son dos temporizadores, un puerto serial y un controlador DMA.

4.2.5.1 Temporizadores

Los dos temporizadores son contadores de propósito general, con dos modos de señalización y pueden operar con reloj interno o externo. Pueden ser utilizados para señalizar al ‘C31 o a dispositivos externos. Cada temporizador tiene una terminal de E/S, que puede ser utilizado como reloj de entrada al temporizador, como señal de salida controlada por el temporizador, o como terminal E/S de propósito general.

4.2.5.2 Puerto serial

Es un puerto bidireccional, que puede ser configurado para transferencias de 8, 16, 24 o 32 bits de datos por palabra. El reloj del puerto serial puede ser originado interna o externamente. Los pines son configurables como pines E/S de propósito general. El puerto serial también puede configurarse como temporizador. Dispone además de un modo de negociación que le permite comunicarse sobre cualquier puerto serial, y la velocidad de operación se ajustara de acuerdo al sistema con el que se está comunicando.

4.2.5.3 Acceso directo a memoria (DMA)

El controlador DMA integrado, puede leer o escribir a cualquier ubicación en memoria, sin interferir con la operación del CPU. El 'C31 puede interconectarse con memorias externas más lentas, sin reducir el rendimiento del CPU. El controlador DMA contiene un generador de direcciones propio, registros de origen y destino y un contador de transferencia.

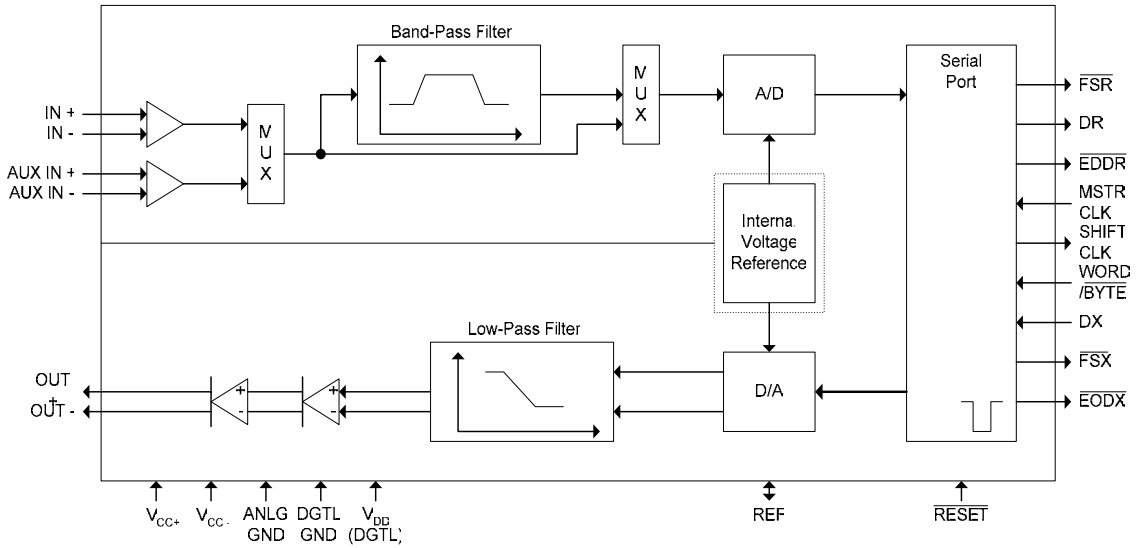
Para minimizar los conflictos entre el CPU y el DMA se dispone de buses dedicados para direcciones y datos. Dado que el DMA y la CPU tienen buses separados, pueden operar independientemente uno del otro. Sin embargo, cuando la CPU y el DMA intentan acceder el mismo recurso interno o externo, se puede exceder el ancho de banda disponible y es necesario establecer prioridades. El 'C31 asigna la mayor prioridad al CPU.

4.3 Circuito de interfase analógica (AIC)

La tarjeta DSK incluye para la adquisición y conversión de señales, del circuito integrado TLC32040. Este es un sistema de entrada y salida, que incluye un convertidor analógico a digital y digital a analógico en un solo chip. Incluye además un filtro pasabanda antialias de entrada, un ADC con resolución de 14 bits, un puerto serial configurable en cuatro modos compatibles con microprocesador, un DAC con 14 bits de resolución y un filtro pasabajos de reconstrucción de salida. Este dispositivo también ofrece diversas combinaciones de frecuencias para el reloj de entrada y tazas de muestreo, que pueden ser cambiadas por software.

En la figura 21 se muestra un diagrama funcional a bloques del TLC32040. En este diagrama, pueden observarse los componentes internos y la relación entre estos, así como las terminales disponibles.

Figura 21. Diagrama funcional a bloques del TLC32040



4.4 Software para programación del DSP

El propósito principal de un sistema como el DSK es experimentar, y utilizar el DSP para procesamiento digital de señales. Por lo tanto, provee de la capacidad de crear y cargar software para ser ejecutado en la tarjeta. Las herramientas para la creación de código que incluye el DSP son un ensamblador y un depurador. En conjunto, estas herramientas permiten desarrollar, probar y depurar código en lenguaje ensamblador para el sistema DSK. En esta sección se examinan las principales características del ensamblador y el depurador, así como el proceso general para el desarrollo de código.

4.4.1 Ensamblador DSK

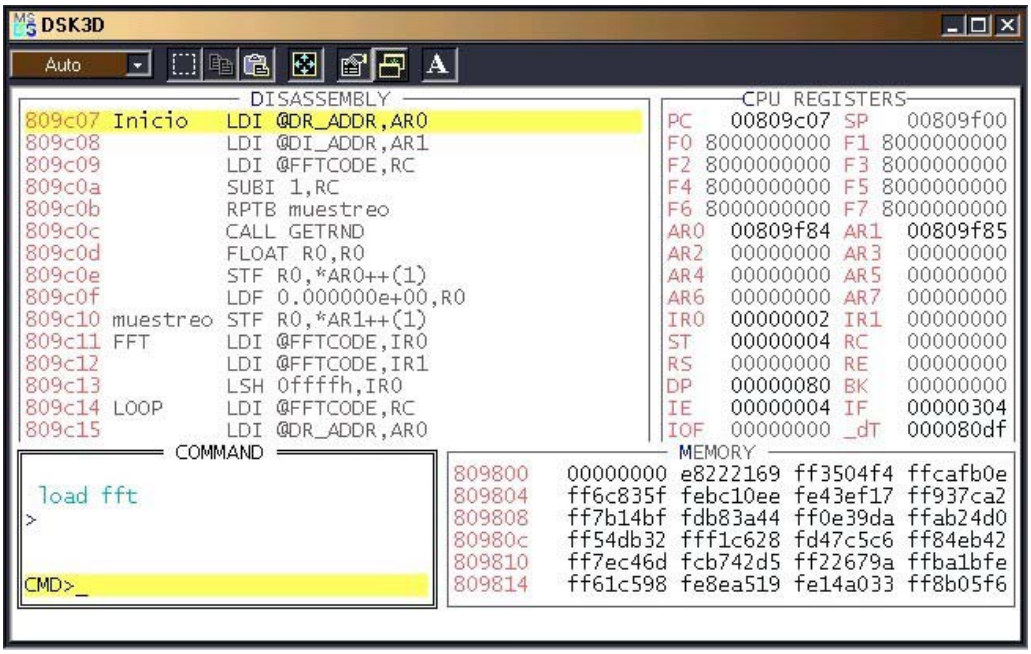
El ensamblador DSK es una herramienta simple y fácil de utilizar. Únicamente incorpora las características más necesarias. Este ensamblador se diferencia de otros más avanzados, ya que no pasa por una fase de vinculación para crear un archivo de salida. En lugar de ello, usa directivas especiales para ensamblar código a direcciones absolutas durante la fase de ensamblado. Como resultado, se pueden crear programas pequeños de manera fácil y rápida. Además, es posible crear programas más grandes combinando archivos individuales con algunas directivas de compilación especiales para este propósito.

4.4.2 El depurador

El depurador es una herramienta de fácil uso y aprendizaje. Contiene un entorno amigable, orientado a ventanas, lo que reduce el tiempo de aprendizaje y elimina la necesidad de memorizar comandos complejos.

El depurador puede cargar y ejecutar código paso a paso, establecer puntos de detención, además la capacidad de parar el programa cuando se desea. En la figura 22, se muestras diferencias características de la pantalla del depurador.

Figura 22. Pantalla del programa depurador.

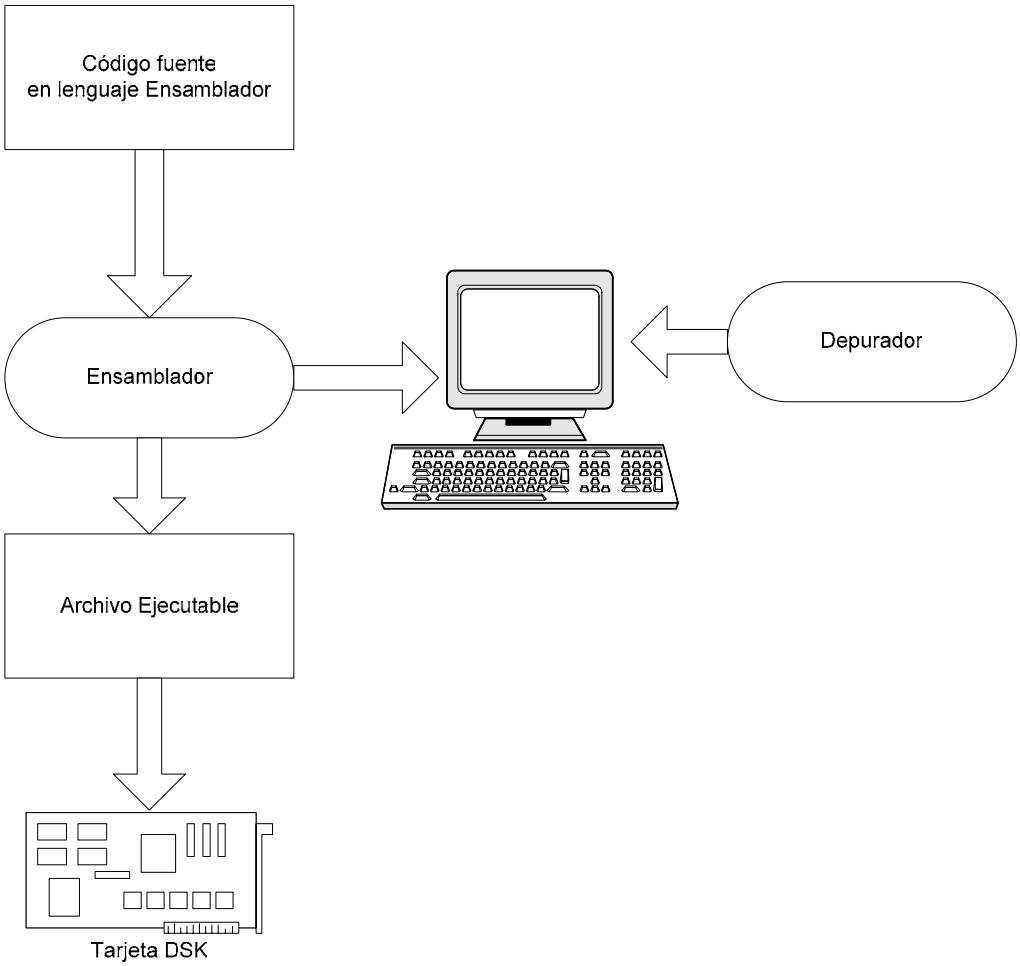


Como puede observarse en la figura 22 el depurador consiste de diferentes ventanas. La ventana *disassembly* permite ver el código en lenguaje ensamblador, esta ventana permite establecer *breakpoints*, observar que instrucción se esta ejecutando (al realizar una ejecución paso a paso). La ventana *register*, muestra todos los registros del DSP, al realizar una ejecución paso a paso, es posible observar la forma como se alteran los registros en respuesta a las instrucciones. La ventana *command* es de utilidad para ejecutar comandos del depurador, por ejemplo comandos para ejecutar n instrucciones, o para invocar el ensamblador. Finalmente, la ventana de memoria, permite visualizar el contenido de la memoria.

4.5 Procedimiento para elaboración de programas

En la figura 23 se presenta un diagrama de flujo, del proceso para escribir un programa que pueda ser ejecutado en el sistema DSK.

Figura 23. Proceso para escribir programas para el sistema DSK



El propósito principal del proceso de desarrollo, es producir un módulo que pueda ser ejecutado en el DSK.

El ensamblador, traslada un archivo fuente en lenguaje ensamblador, al lenguaje de archivo de objetos para el TMS320C31.

El depurador se utiliza para verificar, corregir y depurar la funcionalidad del código.

4.6 Funcionamiento general del sistema DSK

El hardware y software del DSK, están diseñados para proveer una plataforma de desarrollo de bajo costo, de aplicaciones de procesamiento digital de señales en tiempo real. Adicionalmente, se puede verificar el código y su tiempo de ejecución. El DSK dispone además de conectores de expansión, que permiten crear tarjetas hijas, para la expansión del sistema.

4.6.1 Interfase de hardware

El DSK se inicializa al responder a un comando de reinicio, y cargando un *kernel* de comunicaciones. El *kernel* de comunicación, provee las entradas y salidas necesarias para interconectar el DSK y el sistema *host*. Las comunicaciones del *host*, ocurren a través del bus paralelo del 'C31, mientras que las E/S analógicas, son manejadas por el TLC32040 y enviadas al puerto serial del 'C31.

4.6.2 Interfase de hardware para *host*.

La interfase del *host*, conecta el bus paralelo del 'C31 con el puerto paralelo de la PC *host*. Consiste de tres dispositivos, un arreglo lógico

programable (TICPAL22V10Z), y dos *transceiver* para buses octales de tres estados y alta velocidad (74ACT245).

El arreglo lógico programable (PAL), determina cuando el 'C31 esta accesando la interfase *host*, utilizando las señales STROBE, A23, A22, A21 y A20, para decodificar la dirección del 'C31. Los *transceivers* de bus, sirven de *buffer* de datos entre el puerto paralelo de la PC y el bus paralelo del 'C31.

La interfase de *host*, soporta dos tipos de transferencias, un modo de 8 bits bidireccional., que permite transferencias rápidas en puertos paralelo que soportan transferencias bidireccionales. Los puertos de impresión unidireccionales soportan transferencias de 8 bits desde el *host* hacia el 'C31, y de 4 bits, desde el 'C31 hacia el *host*.

4.6.3 Interfase de hardware con el AIC TLC32040

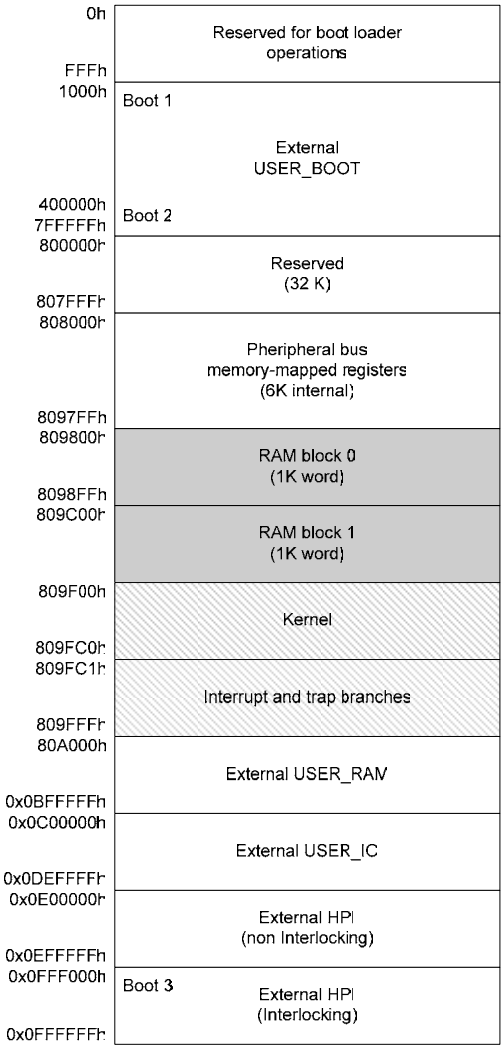
El DSK conecta el TLC32040 al puerto serial del 'C31, a través de un *header* y resistencias de aislamiento de 100 Ω. Los *header* permiten desconectar el AIC y usar el puerto serial del 'C31 en una tarjeta hija. Dos pines adicionales del 'C31 controlan las señales de reinicio y señalización del AIC.

Las entradas y salidas analógicas del AIC, se conectan a través de un conector RCA. Estas señales son compatibles con niveles de línea (+/- 3 V Pico), y pueden ser conectadas a entradas o salidas de audio con éste nivel.

4.6.4 Mapa de memoria del DSK

Dado que las comunicaciones del *host* ocurren a través del puerto paralelo del 'C31, el PAL decodifica las direcciones del 'C31 para determinar cuando esta accesando la interfase de *host*, de acuerdo al mapa de memoria que se muestra a continuación.

Figura 24. Mapa de memoria del DSK



Fuente: User's Manual, TMS320C3x DSK. Pag4-7

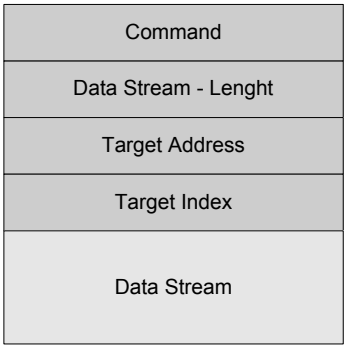
4.6.5 Kernel de comunicaciones del DSK

Luego de una señal de reinicio, el *host* carga un *kernel* de comunicaciones al ‘C31, este *kernel* provee un conjunto de rutinas de bajo nivel, que le permiten al *host* y al ‘C31 intercambiar información y ejecutar funciones de depuración.

4.6.5.1 Paquetes de datos

El *host* y el ‘C31 se comunican intercambiando paquetes de datos, en la figura 25 se muestra la estructura de paquetes de datos. El encabezado de estos paquetes, típicamente consiste de cuatro campos, comando, longitud de datos, dirección destino e índice de destino. Finalmente se agregan los datos a transmitir.

Figura 25. Formato de paquetes de datos del ‘C31.



4.6.5.2 Comandos

Cuando el ‘C31 recibe una petición de interrupción desde el *host*, el ‘C31 guarda el estado actual de la CPU, y luego recibe un paquete. A continuación, el *kernel* de comunicaciones examina el encabezado y extrae la información del

comando. Estos comando, proveen de rutinas de bajo nivel para la comunicación entre el *host* y el sistema. En la siguiente tabla, se presentan los comandos más importantes, ya la función que ejecutan.

Tabla VI Comandos principales del *kernel* de comunicaciones y su función

Comando	Funcion
XWRIT	Escribe un bloque de datos desde el *host*, hacia el DSK. Este comando, toma la longitud de la cadena de datos del *host* y lo escribe en la ubicación de memoria del 'C31 indicada por el campo de dirección de destino. La dirección de destino es incrementada por el índice de destino luego de cada operación de escritura.
XREAD	Lee un bloque de datos del DSK en el *host*. Este comando lee una cadena de datos de la ubicación de memoria del 'C31 indicada por el campo de dirección de destino y lo envía al *host*. El índice de destino incrementa la dirección de destino, después de cada operación de lectura.
XCTXT	Obtiene el contexto del 'C31
XRUNF	Restaura el contexto del CPU y ejecuta código hasta que encuentra un *breakpoint*. Este comando es utilizado para propósitos de depuración.
XSTEP	Restaura el contexto del CPU, ejecuta una instrucción y nuevamente salva el contexto del CPU. Este comando es utilizado para propósitos de depuración.
XALT	Salva el contexto del CPU y espera por nuevos comandos. También es utilizado en depuración del código.

4.7 Software para comunicación con PC

El DSK incluye varios archivos de código fuente, para la manipulación del puerto paralelo y ejecución de las funciones necesarias para inicialización y comunicación de una PC con el ‘C31. Este código, esta programado en el lenguaje C++. En la siguiente tabla, se resumen las rutinas disponibles en estos archivos con la función que ejecuta cada una de las rutinas.

Las rutinas descritas en la Tabla VII, se utilizan en programas en C, para diferentes tareas de comunicación entre la PC y el DSP. Esto se ve con más detalle, en el desarrollo de la aplicación de interfase para el usuario, descrita en el capitulo 6.

Tabla VII. Rutinas para interfase de PC con el DSK.

Archivo Fuente	Función	Descripción
Target.cpp Contiene rutinas para manipular la transmisión de información entre la PC y el 'C31.	Getmem	Esta rutina lee un bloque de datos de la memoria del 'C31
	Putmem	Esta rutina, escribe un bloque de datos en el 'C31.
	SSTEP_CPU	Ejecuta una sola instrucción en el CPU
	RUN_CPU	Ejecuta instrucciones iniciando en la dirección indicada por el contador de programa, y finaliza al encontrar un *breakpoint* si alguno a sido establecido.
	HALT_CPU	Detiene la ejecución de instrucciones en el CPU.
Driver.cpp Contiene rutinas a nivel de *driver*. Estas rutinas manipulan los circuitos de interfase del hardware para restablecer, enviar y recibir datos a través de puertos paralelos de impresión, unidireccionales o bidireccionales.	DSK_reset	Esta rutina inicia el DSK.
	input_rdy	Esta rutina, indica que el DSK esta listo para recibir datos.
	recv_long_byte	Recibe un valor de 32 bits, en cuatro transferencia de 8 bits.
	Recv_long	Recibe un valor de 32 bits, en ocho transferencias de 4 bits
	Xmit_long	Transmite un valor de 32 bits, en cuatro transferencias de datos de 8 bits.
	Xmit_byte	Transmite un valor de 8 bits, en una sola transferencia de datos.
object.cpp Contiene rutinas de alto nivel, basadas en las rutinas de bajo nivel descritas anteriormente. Estas rutinas permiten cargar programas o datos de archivos dsk3a o en formato COFF (*Common Object File Format*), que mueven datos binarios desde la PC hacia el DSK e inicializan el sistema DSK: Estas rutinas asumen la existencia de un *kernel* de comunicaciones válido en la memoria del 'C31.	Load_File	A la función se le pasa un argumento denominado *TASK*, que le indica que tarea debe ejecutar. La tarea más común que ejecuta esta función es cargar un archivo ejecutable en el DSK.
	Init_Communication	Esta función intenta comunicarse con el DSK asumiendo la existencia de un *kernel* de comunicaciones válido. Si esto falla, el DSK se reinicia e intenta cargar el *kernel* hasta n veces. Una vez inicializadas las comunicaciones, la función Load_File carga la aplicación que se desea ejecutar.

5. DESARROLLO DEL SOFTWARE PARA EL SISTEMA DSK

5.1 Introducción

En el presente capítulo, se describe el proceso seguido, para el desarrollo de la primera parte del software, que consiste en el código que es ejecutado en el DSP. La principal función de este código, es la implementación del algoritmo FFT, para obtener los componentes de frecuencia de la señal bajo análisis. Adicionalmente, este código debe programar de manera apropiada el AIC, para la correcta adquisición y conversión de señales.

5.2 Especificaciones del software

El punto de partida, para el desarrollo del software, son sus especificaciones generales. Estas especificaciones, se definieron con mayor detalle en el capitulo 3 en el que se describe el proceso de selección del sistema apropiado para la implementación del proyecto de graduación.

Las especificaciones para el software son las siguientes:

El software para el sistema DSK, consiste en código ejecutable en el DSP, que desarrolle un algoritmo para el cálculo de una FFT de 256 puntos, con datos obtenidos de una señal que es adquirida a través del puerto serial y digitalizada por el AIC. Esta operación debe ser continua, previendo que el control se hace en una PC.

La FFT debe ser de 256 puntos, esto determina la cantidad de memoria de datos necesaria. Debe considerarse espacio para la tabla de factores *twiddle*, donde se almacena la señal digitalizada, y donde se almacena el resultado final.

Otra especificación para el software, es que debe manipular una señal de audio, para ello, es necesaria la comunicación del DSP, con el circuito AIC, este ultimo, también es programable. Por lo tanto, es necesario desarrollar el código correspondiente para programar el AIC a diferentes tazas de muestreo y la manera como transfiere los datos al DSP.

El software en el DSP, no funciona independientemente, por lo cual debe preverse, la manera en la que se comunica con el sistema que lo controla, en este caso, una PC. Además, debe considerarse el formato que tendrá el resultado de la FFT, y de que manera lo interpreta la PC.

En la PC, el resultado leído desde el DSP, se debe almacenar temporalmente en un espacio de memoria para su visualización. Una forma conveniente de hacerlo, es mediante un arreglo de caracteres. Las variables de tipo carácter, utilizan 1 byte de memoria, por lo tanto, con el objetivo de minimizar el calculo en la PC, previo a trasladar el valor calculado, se debe formatear para que ocupe 1 byte (una palabra de memoria del DSP, contiene 4 bytes).

5.3 Diagrama de flujo

En base a las especificaciones generales, se elabora un diagrama de flujo, del funcionamiento deseado del dispositivo. Esta es una aproximación general, que sin embargo es de mucha utilidad para guiar el proceso de

desarrollo del software y al mismo tiempo, dividirlo en tareas más pequeñas y menos complejas.

En la figura 26, se muestra el diagrama a bloques, que incluye las principales tareas que debe ejecutar el sistema.

Para el desarrollo del software, es conveniente dividirlo en dos bloques principales, uno para la programación del AIC y el otro, la programación del algoritmo para la FFT. A continuación se describe con mayor detalle la programación de cada uno de estos bloques.

5.4 Programación del AIC y adquisición de señales

La programación del AIC, requiere cuatro pasos. Inicialmente, es necesario enviar una señal de reinicio al AIC, a continuación debe inicializarse el temporizador del DSP, ya que este suministra el reloj maestro para la operación del AIC, también es necesario programar el puerto serial del DSP, para que se comunique adecuadamente con el AIC. Finalmente, se debe programar el código para que el AIC funcione apropiadamente.

Con el AIC, debidamente inicializado y programado, se procede a desarrollar el código necesario para la adquisición de datos.

El procedimiento anteriormente descrito, se puede resumir en el diagrama de flujo de la figura 27

Figura 26. Diagrama de flujo del software para el DSK

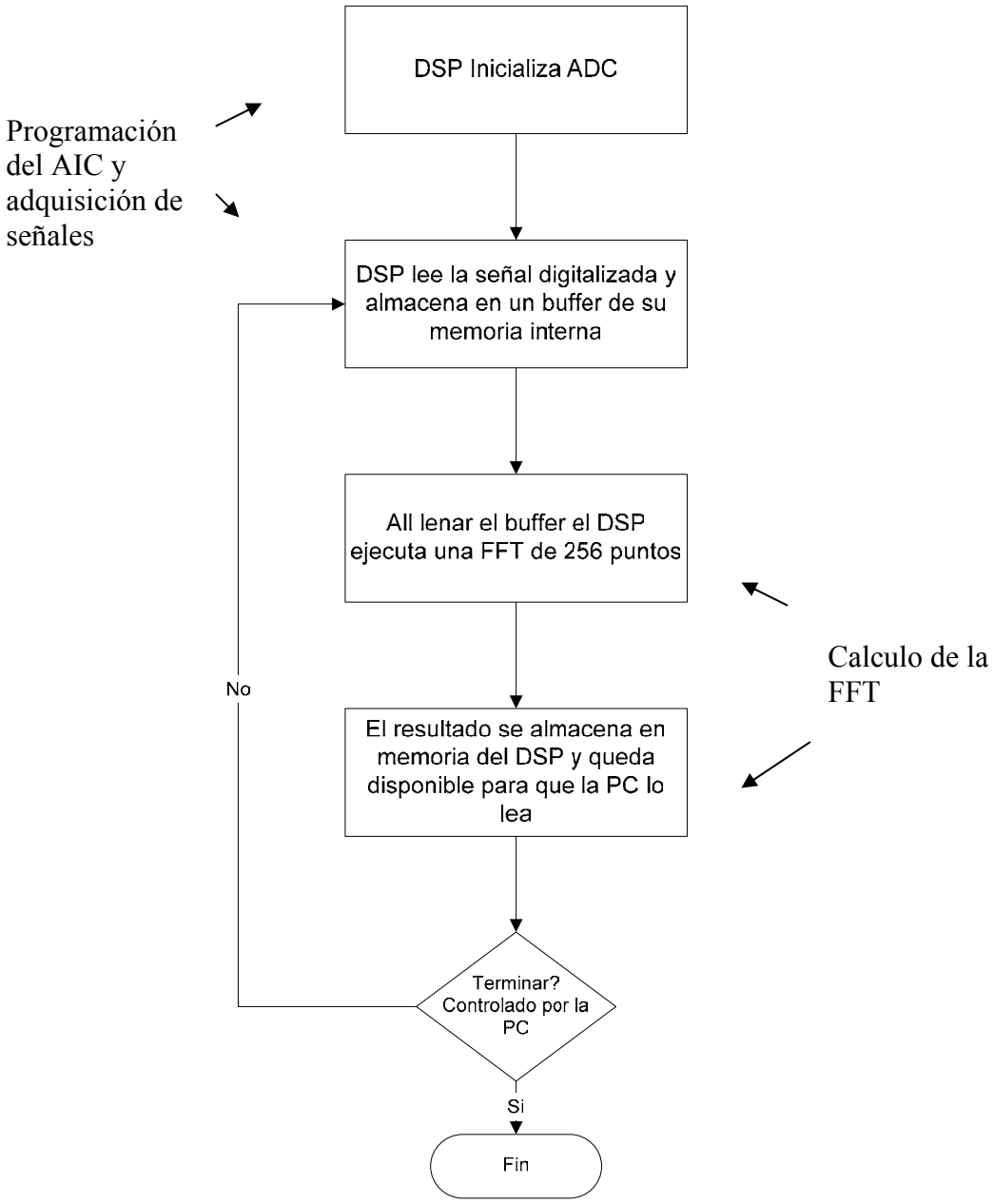
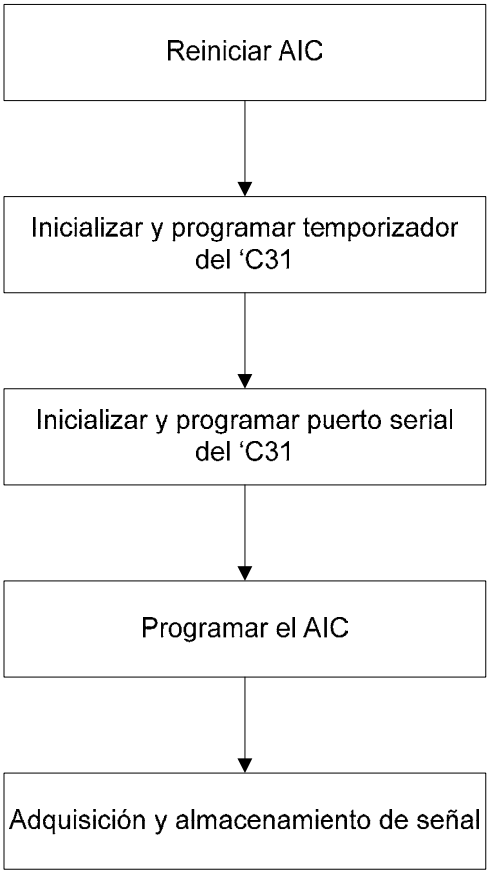


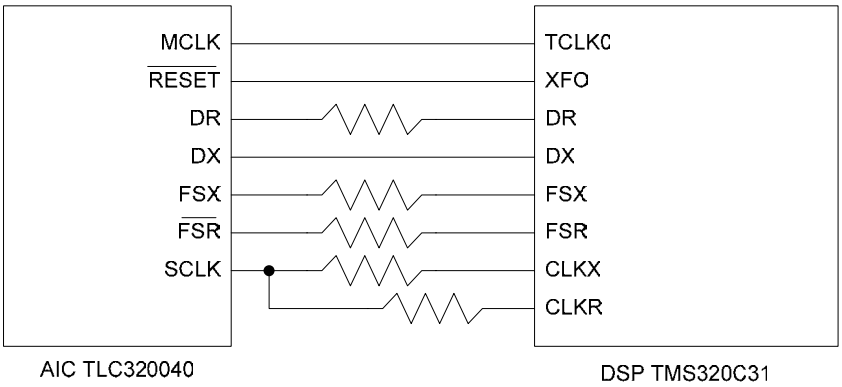
Figura 27. Diagrama a bloques de código para la inicialización y programación del AIC.



En la figura 28, se muestra un diagrama de la conexión entre el DSP y el AIC. En el diagrama se muestran las terminales de cada uno de los dispositivos que están conectadas entre si. Esto es necesario para determinar la forma en la que deberá programarse el AIC.

A continuación, se describe con más detalle, cada uno de los procesos que se observan en el diagrama de flujo de la figura 27.

Figura 28. Conexión entre el DSP y el AIC



Fuente: DSP Starter Kit User Guide, figura 4-1, Pag 4-3

5.4.1 Reiniciar AIC

Para programar el AIC, el primer paso, es enviarle una señal de reinicio. Al hacerlo, se lleva al AIC a un estado conocido, y es posible continuar con su programación.

El AIC, se reinicia enviando un pulso negativo a la terminal RESET, esta terminal, esta conectada directamente a la terminal XF0 del DSP. La terminal XF0 del DSP, es controlada por el registro IOF (Bandera de Entrada/Salida), por lo tanto, enviando una palabra de control a este registro, se puede reiniciar el AIC. El código para ejecutar esta operación es el siguiente:

```
rpts  40 ;la siguiente instrucción se ejecuta 40 veces
ldi   2h, IOF  ;envia un 0 a la terminal XF0 (el valor 2h, programa el pin
                ;XFO como salida y escribe un 0 en el)
ldi   6h, IOF ;escribe un 1 en la terminar XF0 completando el reinicio
                ;del AIC
```

5.4.2 Inicializar temporizador del ‘C31

Para su operación, el AIC, necesita un pulso de reloj externo. Este es necesario para proveer la señalización interna del AIC. El ‘C31 dispone de dos temporizadores, uno de los cuales debe programarse, para proveer al AIC, del pulso de reloj que necesita.

Los temporizadores del ‘C31, se programan a través de tres registros especiales, un registro de control global que determina el modo de operación del AIC, un registro de periodo y un registro contador que se utilizan para especificar la frecuencia a la cual operara el temporizador. Estos registros están mapeados en memoria, por lo que para su acceso, es necesario conocer la dirección de memoria en la que se encuentran.

La inicialización y programación del temporizador se realiza enviando las palabras de control apropiadas, a los registros mencionados en el párrafo anterior. Una selección adecuada de estas palabras de control, debe tomar en cuenta los requerimientos del dispositivo que utilizara la señal de reloj, en este caso el AIC.

Para el proyecto que se esta desarrollando, las palabras de control apropiadas se muestran en la tabla VIII de la siguiente página.

El código completo, para inicializar y programar el temporizador, puede observarse en el Apéndice I.

Tabla VIII. Palabras de control, necesarias para programar el Temporizador 0 del 'C31

Registro del temporizador	Palabra de control	Resultado
Control global	2C1h	Configura la terminal TLCK0 del DSP, como terminal de temporizador, reinicia el registro contador, reinicia el temporizador y especifica que se utilizara el reloj interno del DSP, para generar l pulso de reloj a tráves de la terminal TLCK0
Periodo	1h	Establece la frecuencia del temporizador en 12.5 MHz*
Contador	0h	Establece la frecuencia del temporizador en 12.5 MHz*

* Nota. De acuerdo con la hoja de especificaciones del AIC, la máxima frecuencia para el reloj maestro, es de 10 MHz, por lo que el valor de 12.5 MHz, esta por encima de ese valor. Sin embargo se opto por utilizar un valor más allá del límite, para alcanzar tasas de muestreo más altas que permitieran cubrir todo el espectro de frecuencias de audio.

5.4.3 Inicializar puerto serial

El siguiente paso, en el proceso de programación del AIC, es inicializar y programar adecuadamente el puerto serial del DSP. Esto es necesario, ya que a través de este puerto, el DSP se comunica con el AIC. En el caso particular del programa que se esta desarrollando, el DSP se comunica con el AIC para controlar su operación y a la vez, recibe del AIC la señal digitalizada que este esta procesando.

La operación del puerto serial del DSP, se controla a través de seis registros. De estos registros, únicamente es necesario considerar tres, el registro de control global del puerto serial, el registro de control transmisión (que controla los pines FSX/DX/CLKX) y el registro de control de recepción (que controla los pines FSR/DR/CLKR).

De manera similar a la programación del temporizador del DSP, el aspecto clave en la programación del puerto serial, es la selección de palabras de control adecuadas.

En la tabla IX se resumen las palabras de control seleccionadas para la programación del puerto serial.

Para programar el puerto serial, es necesario escribir un 0 en el registro de control global y a continuación cargar las palabras de control listadas en la tabla IX, el resultado final, es que el puerto serial, esta inicializado y listo para establecer comunicación con el AIC.

Tabla IX Palabra de control para programación del puerto serial

Registro de puerto serial	Palabra de control	Resultado
Control Global	0E970300h	Configura el puerto serial para velocidad de transmisión variable, transmisión y recepción de datos de 16 bits y habilita las interrupciones de entrada y salida (XINT y RINT)
Control de transmisión	111h	Configura los pines FSX/DX/CLKX como pines de puerto serial
Control de recepción	111h	Configura los pines FSX/DX/CLKX como pines de puerto serial

5.4.4 Programar AIC

Una vez que se han inicializado correctamente, el temporizador y el puerto serial del DSP, se procede a programar el AIC para que opere a determinada taza de muestreo. Esto se logra a través de cuatro registros del AIC, dos en la sección de transmisión y dos en la de recepción. Estos registros son denominados A y B.

La frecuencia de conversión esta dada por la siguiente ecuación:

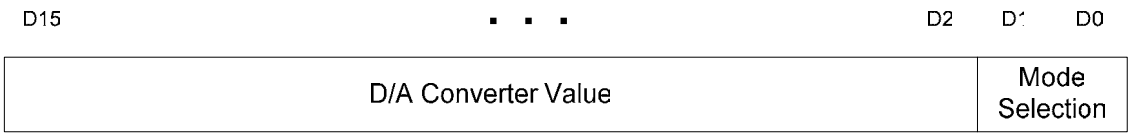
$$\text{frecuencia de conversión} = \frac{MCLK}{2 \times A \times B} \quad (5.1)$$

Para inicializar los registros *A* y *B*, es necesario enviar una comunicación primaria, seguida de una secundaria. La comunicación primaria carga valores en la sección de conversión D/A y determina el modo de comunicación. En tanto que la comunicación secundaria carga información en los registros de la sección A/D. Los valores para los registros *A* y *B*, se cargan durante la comunicación secundaria.

5.4.4.1 Comunicación primaria

La comunicación primaria, contiene un valor de datos en los 14 bits más significativos y utiliza los 2 bits menos significativos para la selección de modo. El formato de esta comunicación primaria se muestra en la figura 29.

Figura 29. Formato de datos de la comunicación primara



Fuente: DSP Starter Kit User's Guide, Pag 4-17

5.4.4.2 Comunicación secundaria

La comunicación secundaria, se da, cuando en la comunicación primaria, los dos bits menos significativos son 1. Esta comunicación secundaria programa el AIC cargando los registros A, A', B y un registro de control que se

utiliza para habilitar o deshabilitar algunas funciones del AIC, como el filtro pasabanda de entrada.

El formato de datos de una comunicación secundaria, se muestra en la figura 30.

Figura 30. Formato de datos de comunicación secundaria

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	TA register value (unsigned)					X	X	RA register value (unsigned)					0	0
X	TA' register value (signed 2s complement)						X	RA' register value (signed 2s complement)						0	1
X	TB register value (unsigned)						X	RB register value (unsigned)						1	0
X	X	X	X	X	X	X	X	Control Register						1	1

Fuente: DSP Starter Kit User's Guide, pag. 418

En base a lo expuesto previamente, la programación del AIC, implica la selección de los valores apropiados para los registros A y B y el registro de control. Ya que estos valores determinan la frecuencia de muestreo, y controlan algunas funciones del AIC, es necesario modificarlos en el transcurso del programa.

Los valores seleccionados para la inicialización del AIC se resumen en la tabla X.

Tabla X. Valores iniciales para los registros A, B y de control del AIC

Registro	Valor	Comentario
A	(TA<<9)+(RA<<2)+0	TA=10, RA=14, en esta ecuación, TA se desplaza 9 posiciones a la izquierda y RA 2 posiciones, para ubicarlas como lo muestra el formato de datos de la figura 30
B	(TB<<9)+(RB<<2)+2	TB = 10, RB = 14, TB se desplaza 9 posiciones a la izquierda y RB 2 posiciones, para ubicarlas como lo muestra el formato de datos de la figura 30
C	00000011b	Con esta palabra de control todas las opciones adicionales del AIC, están deshabilitadas

En base a los valores seleccionados para los contadores de los registros A y B y utilizando la ecuación 5.1, se obtiene una taza de muestreo inicial de 44.642 Khz.

5.4.5 Adquisición y almacenamiento de datos

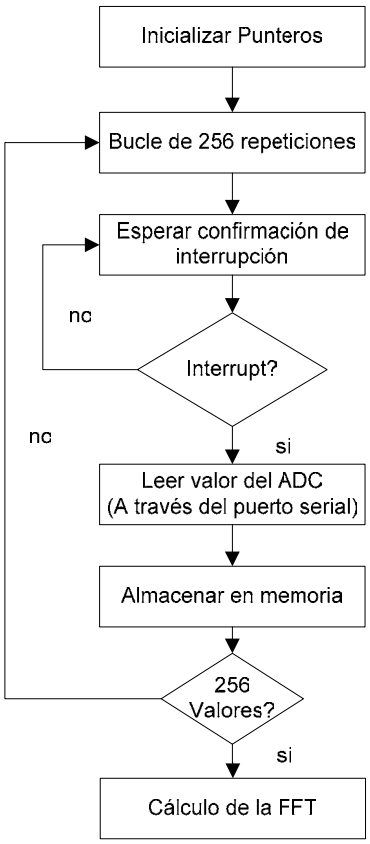
El propósito del código a programar en el DSP, es el de ejecutar un algoritmo FFT de 256 puntos, para ello, es necesario digitalizar la señal de entrada e ir almacenando 256 muestras a la vez.

La señal es convertida de analógica a digital por el AIC. Este a su vez, trasmite la señal hacia el DSP a través del puerto serial. El proceso de transmisión se realiza en base a interrupciones, cuando el DSP recibe un nuevo dato en el puerto serial, genera una interrupción, lee el dato y lo almacena en memoria

En la Fig. 31 se muestra un diagrama a bloques del proceso necesario para leer y almacenar los datos que posteriormente serán utilizados en el cálculo de la FFT.

El código necesario para implementar el diagrama de flujo descrito en la figura 31, se encuentra en el apéndice I.

Figura 31. Diagrama de flujo del proceso de adquisición y almacenamiento de la señal.



5.5 Programación del algoritmo para el cálculo de la FFT

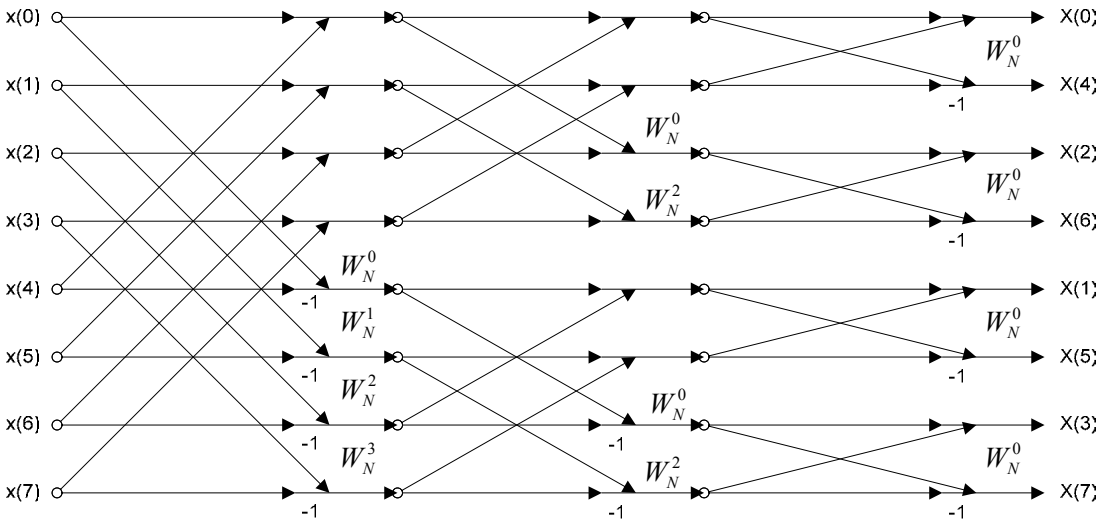
La parte principal del código a ejecutar en el DSP, es para el cálculo de la Transformada Discreta de Fourier mediante la FFT. Este cálculo convierte la señal del dominio del tiempo al de la frecuencia.

Existen diversas maneras de implementar un algoritmo para la FFT, en el capítulo 1, se describe una de las principales, utilizando la técnica denominada Radix-2, para el desarrollo de este proyecto, se utiliza esta técnica.

También existen dos variantes principales, decimado en el tiempo, o en la frecuencia, la diferencia fundamental entre ambas, es en la forma como se aplican los factores *twiddle*. En este proyecto se utiliza decimado en frecuencia, ya que facilita la visualización del proceso de cálculo de la FFT.

En la figura 32, se muestra un diagrama del proceso general para implementar una FFT de 8 puntos, con la técnica radix-2 y decimado en frecuencia. En la implementación de la FFT de 256 puntos, se utiliza un diagrama similar que incluye los 256 puntos.

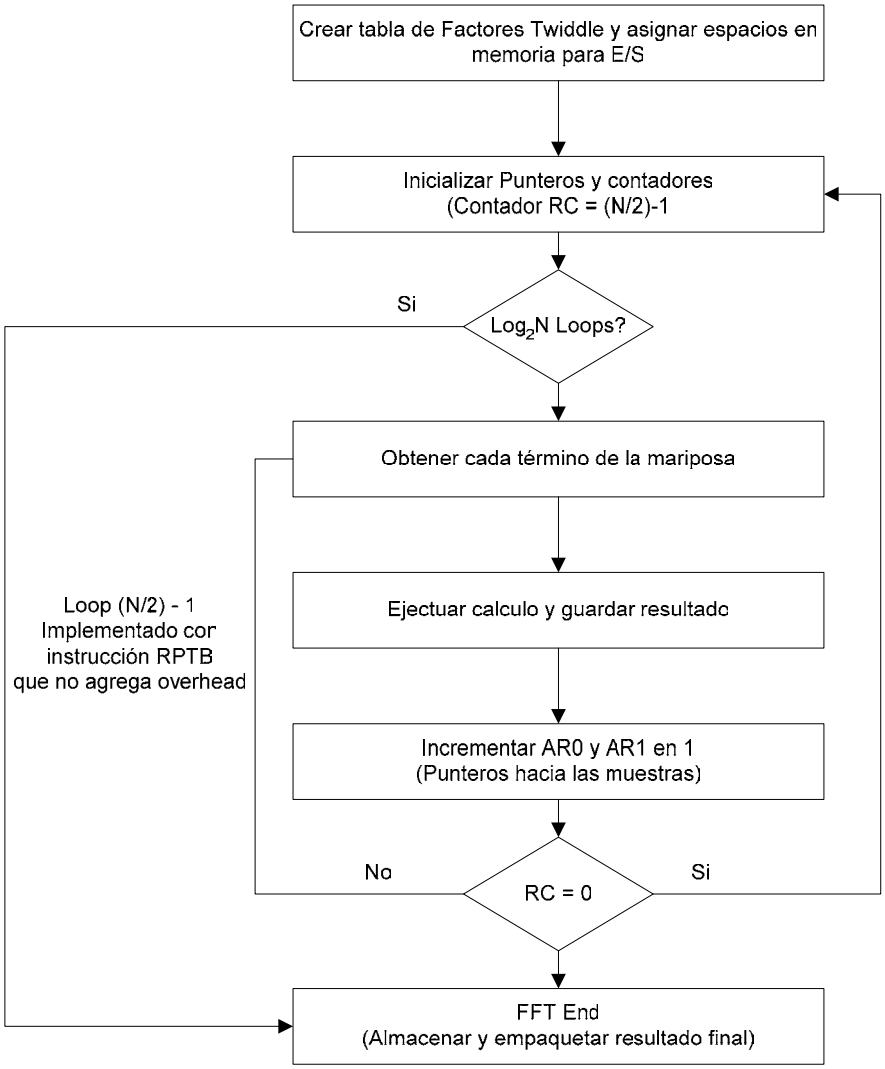
Figura 32. Ejemplo de una FFT radix-2 de 8 puntos, con decimado en frecuencia.



Como en el caso de la programación del AIC, el primer paso para la implementación del algoritmo FFT, es elaborar un diagrama de flujo, que muestres las principales tareas que deben ejecutarse, con el fin de facilitar el

proceso de desarrollo al dividirlo en tareas más pequeñas y menos complejas. Este diagrama, se presenta en la figura 33

Figura 33. Diagrama de flujo, del software para el cálculo de la FFT



En el diagrama de la figura 33, pueden identificarse tres bloques principales, el primero es la creación de la tabla con factores *twiddle*, el segundo bloque es el calculo de la mariposa y el tercero, la operatoria final para obtener el resultado de la FFT.

A continuación se describe con mayor detalle, los pasos necesarios para lo programación de cada uno de los tres bloques principales, mencionados en el párrafo anterior.

5.5.1 Creación de tabla con factores twiddle

Los factores *twiddle*, están dados por la ecuación $e^{-j\frac{2\pi}{N}}$ que se puede expresar como, $\cos(2\pi / N) + jsen(2\pi / N)$, por lo tanto, para crear una tabla de factores *twiddle*, se debe crear una tabla con los componentes reales y otra con los componentes imaginarios.

Para esté propósito, el ensamblador que se incluye como parte de las herramientas de software del DSK, contiene un analizador de expresiones, que soporta algunas funciones integradas de C. Haciendo uso de estas funciones, es posible elaborar una tabla con los factores *twiddle* requeridos.

El código necesario para crear una tabla con factores *twiddle*, se describe a continuación

```
        .start  "TWIDDLES"                ;0X809800
           .sect   "TWIDDLES"                      ;
TWR                                       ; 0x809800
        .loop   N2                        ;
                                    ;
        .float  cos(br($-TWR,N)*PIN)      ;
        .endloop                          ;
TWI                                       ;
        .loop   N2                        ;N2=N/2, N=256
        .float -1*sin(br($-TWI,N)*PIN)    ;PIN = π / N
        .endloop                          ;
```

La directiva *.start* y *.sect*, definen una sección de código que se ensambla en la dirección de memoria 0x809800. Esta sección se denomina *TWIDDLE*S, porque contiene la tabla de factores *twiddle*

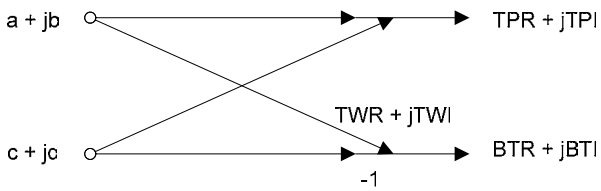
Con las directivas *.loop* y *.endloop*, es posible realizar un bucle que ejecuta una cantidad determinada de iteraciones. En este caso, se realizan 128 iteraciones, de tal manera que se obtiene una tabla con factores *twiddle*, con los 128 factores necesarios.

Las ecuaciones para el cálculo del seno y el coseno, integran como parte de la ecuación, la función br($-TWx, N), esta función, se utiliza para almacenar el resultado del cálculo en orden de bit en reversa.

5.5.2 Calculo de la mariposa

El elemento central en el cálculo de una FFT, es calcular la mariposa. En la figura 34 se muestra una mariposa típica, y la manera de obtener el resultado.

Figura 34. Mariposa típica en el calculo de una FFT radix-2 con decimado en frecuencia



TPR = (a+c)
TPI = (b+d)
BTR = [(a-c)TWR - (b-d)TWI]
BTI = [(a-c)TWI + (b-d)TWR]
TWR = cos(x) (Factor "Twiddle" Parte Real)
TWI = sen(x) (Factor "Twiddle" Parte imaginaria)

El código para el cálculo de la mariposa, se escribe directamente del diagrama mostrado en la figura 34 y considerando el orden de las iteraciones como lo muestra la figura 32.

En primer lugar se inicializan los registros que apuntaran a la dirección donde se encuentran almacenados los 256 puntos y los factores *twiddle*, se ejecuta el cálculo iterando por cada una de las 128 mariposas iniciales.

Una porción del fragmento para el cálculo de las mariposas se muestra a continuación

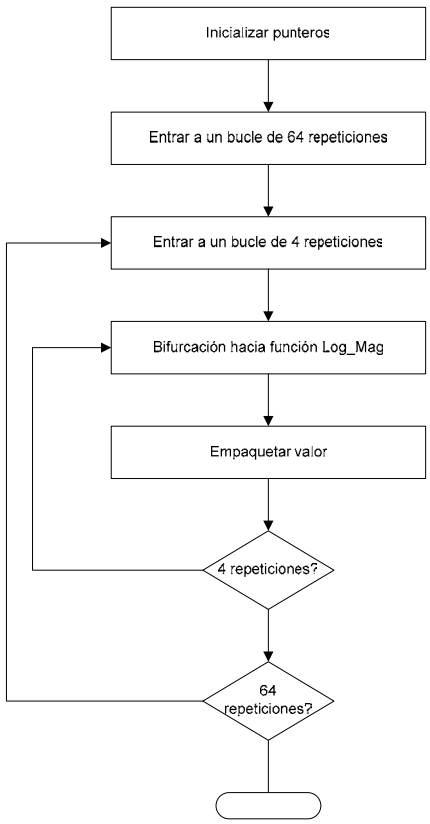
```
ldf   *+AR0(IR1)  ,R0                    ; Carga c
   ||    ldf   *AR0          ,R1         ; Carga a
         ldf   *+AR1(IR1)  ,R2            ; carga d
   ||    ldf   *AR1          ,R3         ; carga b
         ldf   *AR2++(IR0) ,R4            ; Cargar TWR
   ||    ldf   *AR3++(IR0) ,R5            ; Cargar TWI

         addf3 R0,R1,R6            ; (a+c)
         addf3 *+AR1(IR1),R3,R7    ; (b+d)
   ||    stf   R6,*AR0                    ;

         subf3 R0,R1,R6            ; R6=(a-c)
         stf   R7,*AR1             ;
         subf3 R2,R3,R7            ; R7=(b-d)  (R3 free)

         mpyf3 R6,R4,R1            ; R1 = (a-c)*TWR = REAL_1
         mpyf3 R7,R5,R3            ; R3 = (b-d)*TWI = REAL_2
         subf  R3,R1                      ; R1 = BTR = [(a-c)TWR - (b-d)TWI]
         stf   R1,*+AR0(IR1)       ; Almacenar BTR
         nop   *++AR0
         mpyf3 R6,R5,R1            ; R1 = (a-c)*TWI = IMAG_1
         mpyf3 R7,R4,R3            ; R3 = (b-d)*TWR = IMAG_2
         addf  R3,R1                      ; R1 = BTI = [(a-c)TWI + (b-d)TWR]
         stf   R1,*+AR1(IR1)       ; Almacenar BTI
```

5.5.3 Cálculo final

En el diagrama de flujo de la figura 33, puede verse que el bucle que calcula 128 mariposas se ejecuta 8 veces. Al finalizar estos 8 bucles, se obtiene el resultado de la FFT. Sin embargo aún es necesario ejecutar algunas operaciones sobre estos datos, ya que en este momento, lo que se tiene en memoria, es un número complejo, con sus partes real e imaginaria, almacenadas separadamente en la memoria del DSP.

El procedimiento para el cálculo final, se ilustra con el diagrama de flujo que se muestra en la figura 35.

Figura 35. Diagrama de flujo de cálculo final de la FFT



El primero de los bloques en el diagrama de la figura 35, inicializa los registros de direccionamiento, para que apunten hacia donde se encuentra almacenada la información y hacia la ubicación donde se guardara el resultado final. También se inician algunos contadores, para los lazos que se ejecutaran a continuación.

Seguidamente se entra en un bucle, en el que se llama a la función Log_Mag. Esta función, ejecuta dos operaciones sobre los datos almacenados. En primer lugar se aplica una función de ventana, para mejorar la visualización del espectro de frecuencia. En segundo lugar, se calcula la magnitud logarítmica del resultado y se formatea de tal manera que la rutina devuelve el valor calculado, contenido en los 8 bits menos significativos (LSB).

5.5.3.1 Función de ventana

La función de ventana es necesaria, ya que el cálculo de la FFT asume la existencia de una señal periódica. Sin embargo en la practica el periodo de muestreo no encaja exactamente en el período de la señal que se esta analizando. Esto produce discontinuidades en los extremos de la señal muestreada. El efecto de estas discontinuidades es producir un espectro de frecuencia disperso (el aparecimiento de frecuencias espurias).

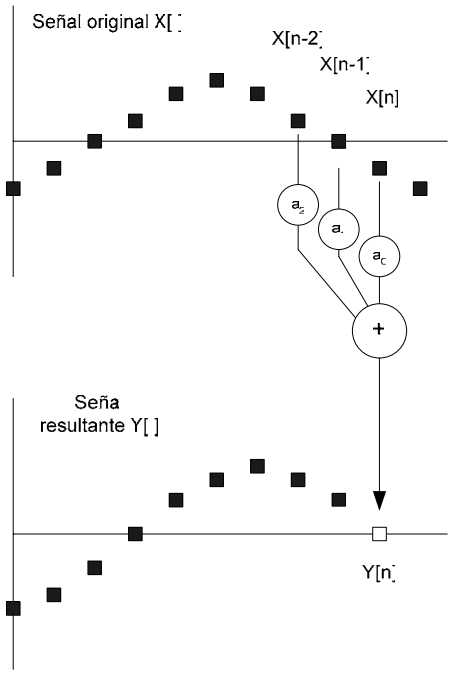
Para minimizar el efecto de las discontinuidades en los extremos se utilizan ventanas espectrales. Estas ventanas suavizan los extremos de la señal mejorando el espectro de frecuencia.

Existen varias ventanas espectrales, que se pueden aplicar, para este proyecto se seleccionó la ventana Hanning, que tiene la particularidad de que sus coeficientes se pueden calcular con mucha facilidad, siendo -5, 1, -5.

Para aplicar esta ventana, es necesario multiplicar tres valores de la señal original, por cada uno de los coeficientes. Esto es equivalente a ejecutar la convolución de la señal original con la función de ventana.

En la figura 36 se observa la manera en la que se calcula la función de ventana, en esta figura, puede verse, que se multiplican tres valores de la señal original por cada uno de los coeficientes y se suman para producir un solo valor. Esto se ejecuta en un lazo, que recorre todas las muestras almacenadas. Cada resultado se eleva al cuadrado y a continuación se suma el cuadrado de las parte real y de la parte imaginaria.

Figura 36. Ejemplo del calculo efectuado por la función de ventana.



5.5.3.2 Calculo del logaritmo

A continuación se calcula el logaritmo del valor obtenido en el paso previo. El resultado obtenido al aplicar el logaritmo, se formatea

adecuadamente, de tal forma que la rutina devuelve el valor del logaritmo, contenido en los 8 bits menos significativos.

5.5.3.3 Empaquetar resultado

Como se menciona en el párrafo anterior, la función Log_Mag, devuelve el valor del logaritmo, contenido en los 8 bits menos significativos, a continuación se empaquetan cuatro resultados (de 8 bits cada uno), en una palabra de memoria.

El procedimiento para empaquetar el resultado, consiste en 4 llamadas consecutivas a la función Log_Mag. Después de cada llamada, el resultado se desplaza hacia la izquierda, en la primera oportunidad 24 bits, a continuación 16 y finalmente 8. Luego de la última llamada, no es necesario ejecutar el desplazamiento, únicamente se concatena con los resultados previos y a continuación se almacena el resultado final. Esta operación se repite 64 veces, y al terminar se dispone del resultado de la FFT, que esta listo para ser llamado por el programa de control para desplegar la FFT.

5.6 Programa definitivo

El siguiente paso en el desarrollo del software consiste en integrar los bloques descritos en las secciones previas. Para efectuar la integración es necesario determinar de qué manera se distribuirá la memoria, definir y asignar nombres a las constantes que serán utilizadas y finalmente escribir el programa utilizando un editor de texto.

En la figura 24, se muestra un mapa, de la memoria del DSK. En dicho mapa, puede observarse el espacio disponible para código y datos. Se dispone de 2K espacios de memoria de 32 bits, dividido en dos memorias físicamente separadas, RAM0 que inicia en la posición 89800h y RAM1, que inicia en la posición 809C00h y termina en la posición 809FFFh.

De acuerdo con el mapa de memoria del DSK, los últimos 256 espacios de la RAM1, están reservados para el núcleo de comunicaciones (Kernel), y las tablas de interrupciones, por lo tanto, debe evitarse sobrescribir esta área.

Para el desarrollo del programa, se utilizara el área de la RAM0, para datos, y el área de la RAM1, para código. Los datos incluyen, espacio necesario para los factores *twiddle*, y para los *buffers* que almacenaran los valores reales e imaginarios, y el resultado final.

En la figura 37, se muestra un diagrama, con la distribución de los datos y el código en la memoria interna del 'C31.

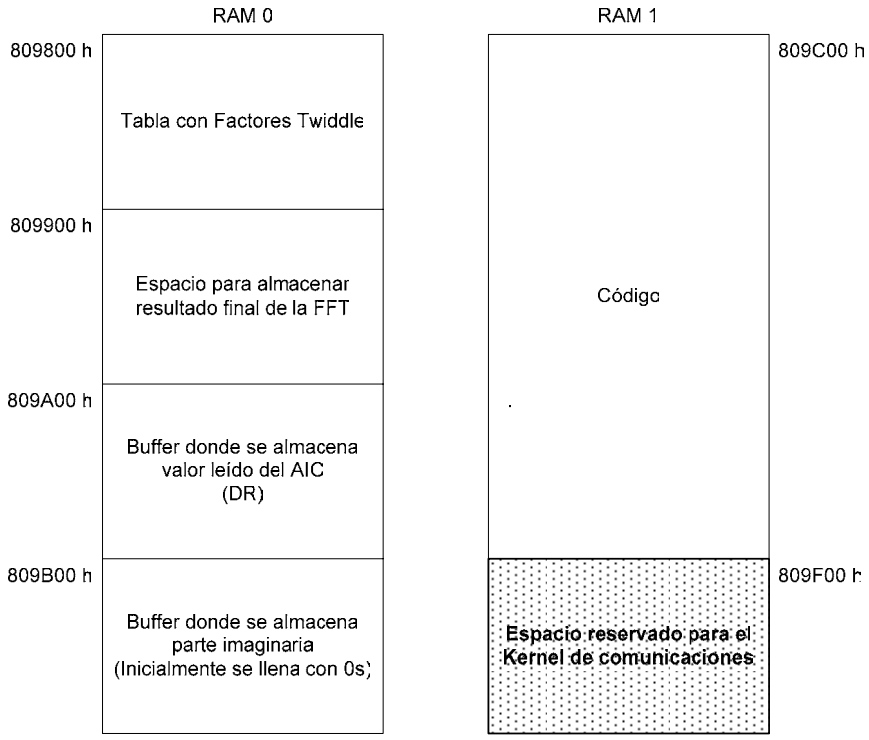
Una vez definida la distribución de memoria, se procede a escribir el código, utilizando para ello un editor de texto.

Para definir constantes, se utilizan dos directivas de compilación. La directiva *.set*, que asocia un símbolo, con un valor. También se utiliza la directiva *.word* que escribe el valor asignado en una palabra de memoria, de la sección en la que se este compilando el código.

De esta manera, con la directiva *.set*, se definen símbolos, que son usados en algunas expresiones, por ejemplo PI, define el valor de *pi*, y se utiliza en el cálculo de los factores *twiddle*. Con la directiva *.word*, se definen valores

que estarán ubicados en un lugar específico en memoria, por ejemplo el valor de las palabras de control que se enviaran al AIC.

Figura 37. Distribución de datos y código en la memoria interna del 'C31



Para ubicar el código en secciones, se utilizan dos directivas, la directiva *.start*, seguida de *.sect*. La primera directiva define el inicio de una sección en una dirección determinada. La segunda directiva define el nombre de la sección y ensambla el código correspondiente. Por ejemplo, para definir el espacio donde se creara la tabla de factores *twiddle*, se puede realizar de la siguiente manera

```
.start    "TWIDDLES", 0x809800      ; Define el punto de inicio en 0x809800
.sect     "TWIDDLES"                ; Define el nombre de la sección.
```

En el programa que se esta desarrollando, deben definirse secciones para el código de inicialización, la tabla de factores *twiddle*, el código de la FFT, y finalmente una sección donde se instalan los vectores de interrupción.

5.7 Ensamblado

Una vez que se ha escrito el programa, en un editor de texto, se procede al proceso de ensamblado, para ello, se utiliza el ensamblador que es incluido como parte de las herramientas de desarrollo de software del DSP.

Para iniciar la operación del ensamblador, es necesario abrir una sesión de comandos de DOS, y dirigirse hacia el directorio donde se encuentra el ensamblador. A continuación se utiliza el siguiente comando

```
C:\DSKTOOLS\dsk3a fft256.txt
```

Donde *fft256.txt*, es el archivo que se va a ensamblar

Durante el proceso de ensamblado, es posible que aparezcan mensajes de error. El ensamblador muestra un mensaje en el que indica el tipo de error y la línea donde encontró el error. Estos errores, pueden ocurrir por diferentes causas, generalmente son errores en la sintaxis de las instrucciones, o en símbolos inconsistentes.

Una vez que se han corregido los errores, el ensamblador produce un archivo ejecutable, en este caso llamado *fft256.dsk* que puede ser cargado en el DSP.

5.8 Depuración

El ensamblador puede detectar errores de sintaxis, sin embargo, esto no garantiza que el programa funcione como se espera. Para cerciorarse del correcto funcionamiento del programa, es necesario utilizar el depurador que se incluye como parte de las herramientas de software. En la figura 22, capítulo 4 se muestra la pantalla del depurador.

Con el depurador, se pueden corregir errores en el código y verificar que la funcionalidad de este, es la que se espera. Se puede ejecutar el código paso a paso, verificando la manera como se alteran los registros y la memoria, en respuesta a cada instrucción.

La depuración del código se hace en dos partes, una el código de inicialización, y la otra, el código para el cálculo de la FFT

5.8.1 Código de inicialización

En esta sección del código, se verifica que los registros del puerto serial y del temporizador, estén direccionados apropiadamente, y se escribe en ellos la palabra de control adecuada. Este código también inicializa el AIC, sin embargo, debido a que este procedimiento utiliza interrupciones, no es posible verificar su funcionalidad utilizando el depurador.

5.8.2 Código para el cálculo de la FFT

Para depurar el código de la FFT, se utilizo el procedimiento de simular la presencia del AIC, generando números aleatorios. A continuación se ejecuta el programa paso a paso, verificando el direccionamiento adecuado hacia los

datos, que el resultado se almacena en el espacio asignado y que el flujo del programa se realiza de manera correcta.

Una vez finalizado el proceso de depuración, se dispone del programa que es ejecutado en el DSP. Para obtener la funcionalidad completa, es necesario escribir un programa que permita la comunicación de una PC, con el sistema DSK, el desarrollo de este programa, esta descrito en el siguiente capítulo.

6. SOFTWARE PARA PC

6.1 Introducción

En este capítulo, se describe el proceso necesario para desarrollar el software de interfase, para el usuario, que permite interactuar con el DSP y visualizar la gráfica del analizador de espectro.

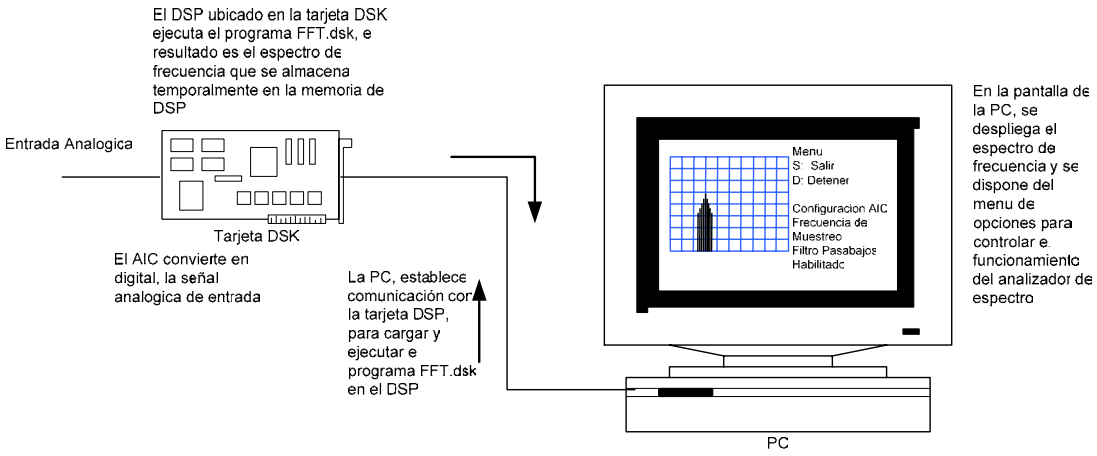
6.2 Especificaciones

El software para PC, debe cumplir con dos objetivos primordiales. En primer lugar, establecer comunicación con el sistema DSK para cargar y ejecutar el programa que calcula la FFT en el DSP, y debe mantener la comunicación, para poder leer el resultado de la FFT y trasladarlo a la PC a través del puerto paralelo. En segundo lugar, el programa debe proporcionar la gráfica del espectro de frecuencia, tomando como datos, el resultado de la FFT y proporcionar una interfase gráfica, para que el usuario pueda controlar la operación del analizador de espectro.

El programa esta desarrollado en lenguaje C, debido principalmente a que el sistema DSK proporciona varios programas en C, para la comunicación entre la PC y el sistema DSK.

En la figura 38 se ilustra la funcionalidad requerida en el software para PC.

Figura 38. Especificaciones del Software para PC



6.3 Desarrollo del software

6.3.1 Algoritmo

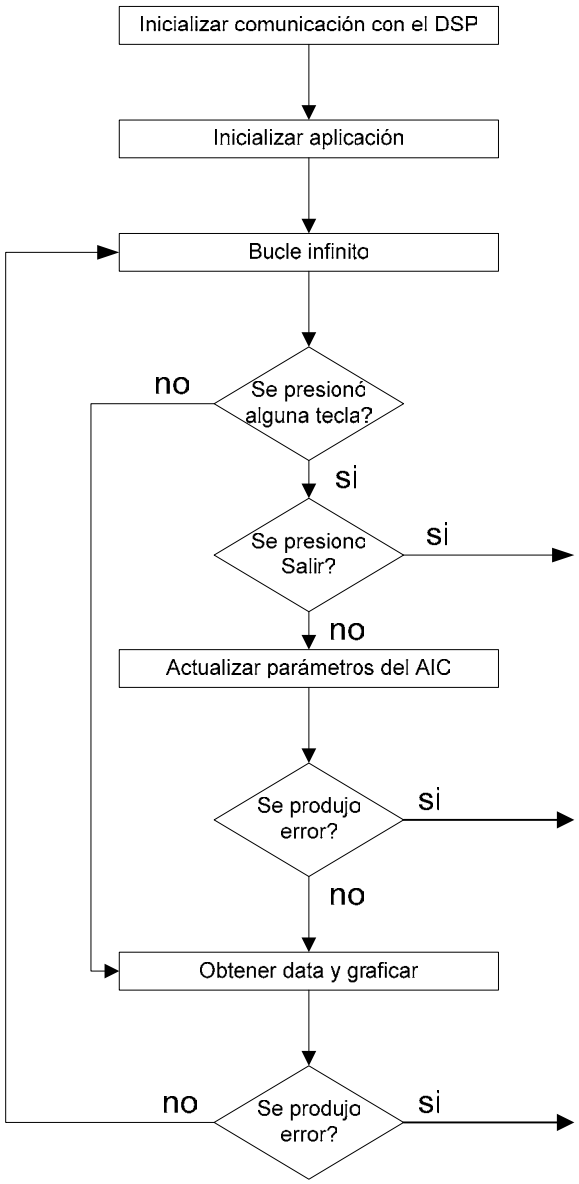
El primer paso en el desarrollo del programa, es elaborar un algoritmo, que muestre la secuencia de pasos a seguir, para lograr la funcionalidad requerida.

En la figura 39 se muestra el algoritmo para el programa a desarrollar, este algoritmo sirve como base, para explicar el proceso de desarrollo del software.

Para la implementación de este algoritmo, es necesario utilizar algunos programas incluidos como parte de las herramientas de programación del DSK. Estos programas, son aplicaciones en C, que contienen funciones necesarias para la comunicación entre la PC y el DSP.

En la tabla VII ubicada en el capítulo 4, se especifican las funciones más importantes, con una breve descripción de las mismas.

Figura 39. Algoritmo para desarrollo del software.



En la tabla XI se describen las funciones utilizadas para la programación del analizador de espectro

Tabla XI. Funciones en C utilizadas en el programa para PC.

Función	Archivo de origen
Scan_Command_Line	Object.cpp
Init_Communication	Object.cpp
HALT_CPU	Target.cpp
Load_File	Dsk_coff.cpp
RUN_CPU	Target.cpp
HPI_STRB	Driver.cpp
HPI_ACK	Driver.cpp
Putmem	Target.cpp
Getmem	Target.cpp

6.3.2 Inicializar comunicación con el DSP

La primera tarea que debe ejecutar el programa, es cargar en el DSP el *kernel* de comunicaciones. Este *kernel*, soporta diferentes comandos que permiten a la PC, leer y escribir datos en la memoria del DSP.

Adicionalmente se debe cargar el código que será ejecutado por el DSP en este caso la aplicación **.dsk*, programada previamente, y que esta descrita en el capítulo 5.

Para inicializar la comunicación con el DSP, se utilizan 4 de las funciones, incluidas en el software del DSK.

A continuación se muestra el fragmento del código necesario para inicializar las comunicaciones.

```
Scan_Command_line(DSK_EXE);
//-------------------------------------------------
for(;;)
{ if (Init_Communication(10000) == NO_ERR) break;
  if (kbhit()) exit(0);
}
HALT_CPU;
if ((err = Load_File(DSK_APP, LOAD)) != NO_ERR)
{  printf("%s %s\n", DSK_APP, Error_Strg(err));
   exit(0);
}
if ((err = Load_File(DSK_APP, SLOAD)) != NO_ERR)
{  printf("%s %s\n", DSK_APP, Error_Strg(err));
}
RUN_CPU();      //Ejecutar aplicación FFT256.DSK, en el DSP
```

La función *Scan_Command_line* lee argumentos de la línea de comandos de la aplicación EXE en base a lo cual establece algunas variables globales. Estas variables globales incluyen la selección del puerto paralelo, el ancho de banda de la transferencia entre el puerto paralelo y el DSP.

Seguidamente se ejecuta la función *Init_Communication*. Esta función determina si existe un *kernel* de comunicaciones válido en el DSP, si no lo encuentra, carga el *kernel* de comunicaciones. La función se utiliza dentro de un bucle infinito, del cual se sale si no se produce ningún error al cargar el *kernel*. De producirse un error, se genera un mensaje, que indica las posibles causas de error e indica que debe presionarse una tecla para salir de la aplicación.

Los errores al cargar el *kernel*, pueden producirse si la tarjeta no esta conectada, o el puerto paralelo no se esta comunicando apropiadamente con el DSP. El mensaje de error, que se produce, da indicaciones sobre las posibles causas y la manera de solucionar el problema.

Una vez inicializado con éxito, las comunicaciones, se ejecuta la función *HALT_CPU*, que detiene la operación del DSP. Esto se hace con el fin de prepara al DSP, para cargar un nuevo programa.

El siguiente paso en el proceso, es cargar la aplicación DSK. Esto se hace con la función *Load_File*. Esta función también se ejecuta dentro de una sentencia *if* que determina si se produce un error en la operación. Al producirse un error, el programa termina y se genera un mensaje de error, que indica las causas del fallo. La principal causa por la que se puede dar un error, es que el archivo DSK, no exista. Otra razón para que se produzca un error, es por problemas de comunicación con el DSP.

De no producirse error en la carga de la aplicación, se ejecuta la siguiente sentencia, que llama a la función *RUN_CPU*. Esta función, inicia la operación de la aplicación DSK.

6.3.3 Inicialización de la aplicación

Adicionalmente a la inicialización de la comunicación entre la PC y el DSP, es necesario inicializar la aplicación. Este es un proceso que consiste en la llamada a dos funciones diseñadas para este software. Una de las funciones inicializa el modo gráfico de DOS en la PC, la otra despliega la pantalla del analizador de espectro y el menú de opciones.

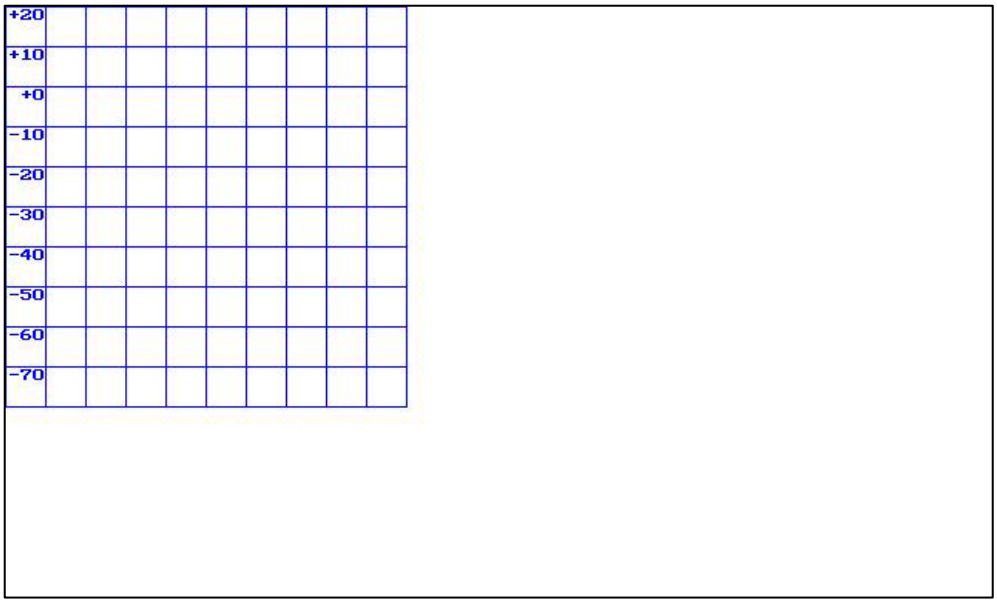
El código necesario se reduce a dos líneas, que se muestran a continuación

```
init_graphics();
menu();
```

En primer lugar se llama a la función *init_graphics*, esta función inicializa el modo gráfico de DOS. Esto es necesario para soportar la función de graficación, utilizada para desplegar el analizador de espectro.

Además de inicializar el modo gráfico, esta función despliega la pantalla del analizador de espectro, esta pantalla puede verse en la figura 40.

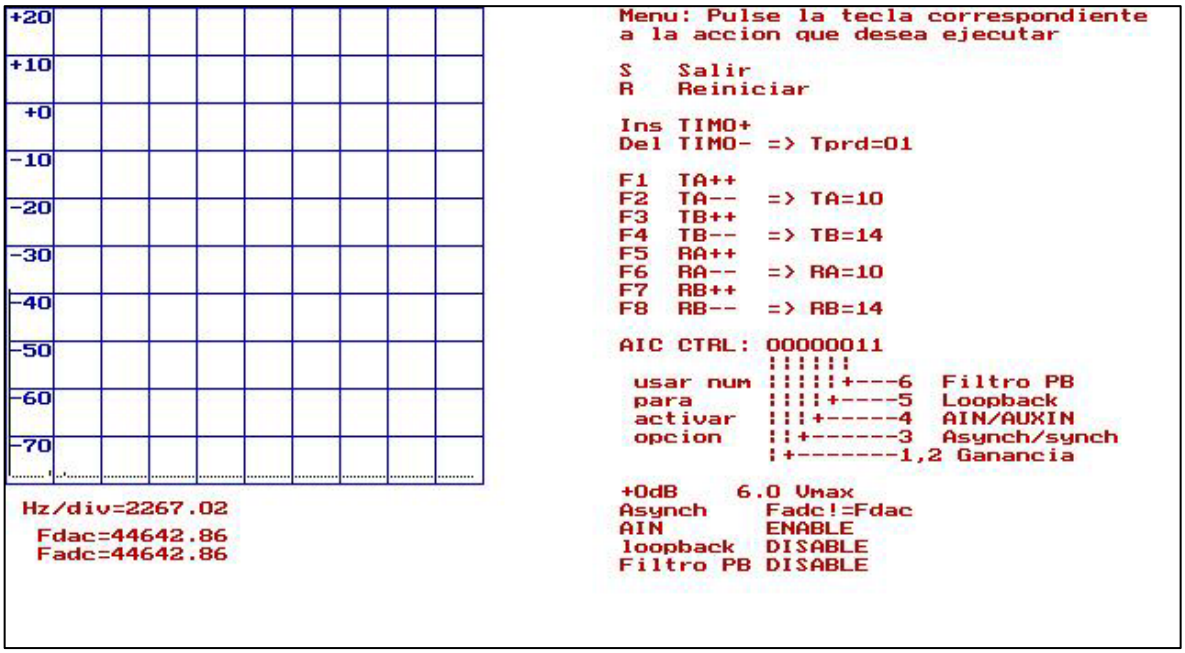
Figura 40 Pantalla del analizador de espectro sin menú de opciones



A continuación se llama a la función *menu*. Esta función cumple con dos propósitos. El primer es desplegar en pantalla el menú de opciones para la operación del analizador de espectro. El segundo es verificar si alguna tecla a sido presionada y ejecutar la acción correspondiente.

En esta sección de código, la llamada a la función menú es con el primer propósito mencionado en el párrafo previo, esto es, desplegar las diferentes opciones para el usuario. La pantalla luego del llamado de estas dos funciones se muestra en la figura 41.

Figura 41. Pantalla del analizador de espectro con menú de opciones.



Las diferentes opciones del menú se accedan a través del teclado de la pc, a continuación se describe cada opción

Tabla XII. Opciones de la pantalla de menú del analizador de espectro

Tecla	Resultado
S	Salir de la aplicación
R	Reiniciar la aplicación
Ins	Incrementar la frecuencia de muestreo DAC
Del	Reducir la frecuencia de muestreo del DAC
F1-F8	Modificar la frecuencia de muestreo del DAC y del ADC
1,2	Aumentar o reducir la ganancia de la señal de entrada al DAC
3	Definir la operación del AIC como sincronía o asincrona
4	Activar la entrada auxiliar
5	Enviar la señal de salida del AIC, de regreso a la entrada
6	Activar o desactivar el filtro pasabanda del AIC

La pantalla de visualización del analizador de espectro, es el área definida por la cuadricula azul, es un grafico de frecuencia vrs amplitud. La amplitud (en *db*) de los componentes de frecuencia se muestra en las líneas horizontales, en tanto que en las líneas verticales se muestra la frecuencia de la señal.

Las divisiones verticales, permiten determinar la frecuencia de las señales que se están analizando. La escala vertical varía según la frecuencia de muestreo del AIC. En la pantalla de ejemplo de la figura 40, cada línea vertical esta espaciada por 2,267.02 Hz, este valor puede leerse en la esquina inferior izquierda de la pantalla del analizador de espectro y corresponde a una frecuencia de muestreo de 44 Khz. Para frecuencias de muestreo menores, el valor de cada división vertical disminuye, permitiendo analizar mejor las señales a baja frecuencia.

6.3.4 Actualización de parámetros y graficación

Una vez realizada la inicialización, tanto de la comunicación entre la PC y el DSP, como de la aplicación en la PC, se entra a un bucle infinito. En este bucle la PC interactúa con el DSP. En primer lugar verifica si se ha presionado alguna tecla y realiza la acción correspondiente, de lo contrario, lee la memoria del DSP y grafica el resultado. El bucle se termina, si se presiona la letra S (que ejecuta la acción de cerrar la aplicación), o si ocurre un error en la comunicación entre la PC y el DSP.

A continuación se muestra el código necesario para implementar la graficación del espectro de frecuencia de las señales.

```
for (;;)
{
    HPI_STRB(0);
    reset=0;
    for (;;)
    {
      if (kbhit())
      {
          reset=menu();
          nparam=1;
      }
      if (HPI_ACK()) break;
     delay(1);
    }
    if (reset) break;
    if (nparam)
    {
      if (putmem(T0_prd   ,1,&T0_prdv)!=NO_ERR) break;
      if (putmem(T0_count,1,    &Zero)!=NO_ERR) break;
      if (putmem(T1_prd   ,1,&T0_prdv)!=NO_ERR) break;
      if (putmem(T1_count,1,    &Zero)!=NO_ERR) break;
      if (TB>TA)
      {
         aic = A_REG;
         if (putmem(A_BOX,1,&aic)!=NO_ERR) break;
         aic = B_REG;
         if (putmem(B_BOX,1,&aic)!=NO_ERR) break;
      }
      else
      {
         aic = B_REG;
         if (putmem(A_BOX,1,&aic)!=NO_ERR) break;
         aic = A_REG;
         if (putmem(B_BOX,1,&aic)!=NO_ERR) break;
      }
      aic = C_REG;
      if (putmem(C_BOX,1,&aic)!=NO_ERR) break;
    }
    nparam = 0;
    if (getmem(MEMDSP, FFTSize/8,(ulong *)buf_0)!=NO_ERR) break;
    putmem(MSG_DSP,1,&MSG);
    draw_vect();
```

El segmento de software, consiste de un bucle infinito. De este bucle se puede salir por una de dos razones. La primera es al presionar la letra S, que ejecuta la acción de salir y terminar programa. La otra forma de salir del programa, es si existe un error de comunicación entre la PC y el DSP.

La primera sentencia que se ejecuta en este bucle infinito, es un llamado a la función *HPI_STRB*. El objetivo de esta función, es mantener la comunicación entre la PC y el DSP. Esto es necesario ya que cuando transcurre cierto tiempo en que el DSP no recibe información, este puede terminar la operación. Para evitarlo, se utiliza esta función que simula una petición de información al DSP.

La función *HPI_STRB*, se utiliza junto a la función *HPI_ACK* que recibe la confirmación del DSP. En este programa, la llamada a las dos funciones esta separada por unas líneas de código que determinan si se ha presionado una tecla. Esta separación es necesaria, para asegurarse que la PC determine si se presiono una tecla.

Después del llamado a la función *HPI_STRB*, se entra a un bucle infinito. En este bucle, la PC detecta si se presiono una tecla. Esto se hace llamando a la función *kbhit* (esta es una función integrada de C, que devuelve el valor de la tecla presionada).

Cuando una tecla a sido presionada, se ejecuta una nueva llamada a la función menú, esta función determina que tecla fue leída, y ejecuta la acción correspondiente, adicionalmente la variable *nparam* (que indica la presencia de nuevos parámetros), adquiere el valor de 1.

Si no se ha presionado ninguna tecla, se llama a la función *HPI_ACK* y a continuación se sale del bucle infinito.

A continuación se ejecuta una sentencia condicional, que se ejecuta cuando la variable *nparam* es igual a uno. Esto como se vio recientemente, ocurre cuando a sido presionada una tecla.

El cuerpo de esta sentencia condicional, traslada hacia el DSP, los nuevos parámetros de operación del AIC. Esto se logra a través de la función *putmem*.

La función *putmem*, escribe datos en una ubicación determinada del DSP. A manera de ejemplo la siguiente sentencia *putmem(T0_prd, 1, &T0_Prdv)* escribe en la ubicación de memoria *T0_prd* del DSP, el valor contenido en la variable *T0_prdv* definida en C. El numero uno, indica que únicamente se escribe una posición de memoria del DSP.

Mediante el procedimiento descrito en el párrafo anterior, se cargan los nuevos parámetros de operación del AIC. La función *putmem*, también se llama junto a una sentencia condicional, de esta manera, la operación se interrumpe si se produce un error en la comunicación con el DSP.

El siguiente paso, es leer desde el DSP, la información de la FFT, esto se hace mediante la siguiente sentencia

```
if (getmem(MEMDSP, FFTSize/8,(ulong *)buf_0)!=NO_ERR) break;
```

La función *getmem*, ejecuta la opuesto a la función *putmem*, ya que su función es leer un bloque de memoria del DSP y trasladarlo a la PC. *MEMDPS* indica la dirección inicial del bloque a leer, *FFTSize/8*, determina que es un

bloque de 32 posiciones de memoria el que se va a leer y *buf_0* indica que el resultado se almacena en un arreglo de caracteres denominado *buf_0*.

La razón por la que se lee un bloque de 32 posiciones, es porque como se recordará, en el programa que calcula la FFT, la información se empaqueta guardando cuatro resultados por posición de memoria. De esta manera, el resultado final de la FFT, esta contenido en 64 posiciones de memoria. Sin embargo, de la FFT, solo 128 valores tienen información útil, ya que los otros 128, corresponden al espectro negativo y es información redundante.

Como en el caso de *putmem*, la función *getmem*, se llama dentro de una sentencia *if*, ya que de producirse un error, la aplicación genera el mensaje de error y termina.

Luego de ejecutar la operación de lectura de la memoria del DSP, se envía un mensaje al DSP (llamando la función *putmem*), que le indica al DSP que debe reiniciar su operación, calculando un nuevo conjunto de valores de la FFT.

Finalmente se llama a la función *draw_vect*, que es la encargada de graficar la señal. La función *draw_vect*, ejecuta un bucle en el que recorre los 256 valores leídos y grafica una línea correspondiente a cada valor.

El programa completo, de la interfase de usuario, puede verse en el apéndice II, de este trabajo de graduación.

6.4 Compilación

La mejor manera de escribir el programa, es en el editor de alguna herramienta para desarrollo de software. Para este proyecto, se utilizó el entorno de programación, Turbo C++, versión 3.0. En un editor especializado, se tiene la ventaja de que se utilizan diferentes colores, para representar variables, palabras reservadas, comentarios. De esta manera, se facilita el proceso de escribir el programa.

Una vez escrito el programa, es necesario compilarlo. Para el proceso de compilación, es necesario tomar en cuenta, los diferentes archivos que contienen las funciones que incluye el software del DSK. Esto se logra a través de archivos de proyecto.

Un archivo de proyecto, incluye:

- Todos los archivos necesarios para el proyecto
- Donde encontrar los archivos en el disco
- Los archivos de encabezado de cada modulo fuente
- Que opciones de compilación se aplicaran, al crear cada parte del programa.
- Donde colocar el programa resultante
- El tamaño del código, tamaño de los datos y número de líneas de cada compilación.

En el archivo de proyecto creado para el programa de la PC, se incluyen los siguientes archivos:

- Fft05.cpp Archivo principal
- driver.cpp Drivers de la impresora de bajo nivel
- target.cpp Comandos a nivel del DSK
- object.cpp Rutinas para cargar aplicaciones al DSK
- dsk_coff.cpp Utilerias para carga de archivos DSK y COFF
- errormsg.cpp Mensajes de retorno de la mayoría de funciones
- symbols.cpp Tabla de símbolos (necesarias para dsk_coff)
- textwin.cpp Funciones de ventana de texto a nivel de DOS
- egavga.obj drivers para gráficos EGA y VGA

El archivo *fft05*, es el archivo cuyo desarrollo se escribe en este capítulo y constituye el programa principal.

El proceso de compilación recorre todos los archivos. Si encuentra algún error en cualquiera de los archivos, el proceso de compilación se detiene. En este caso es necesario corregir todos los archivos. Cuando el proceso de compilación logra ejecutarse sin errores, se puede proceder a la etapa de depuración

6.5 Depuración

El proceso de depuración se utiliza para corregir errores en el código. El entorno de desarrollo Turbo C++, incluye un depurador bastante completo, que permite ejecutar secciones específicas de código, establecer puntos de detención. También es posible analizar el contenido de variables y sus direcciones de memoria.

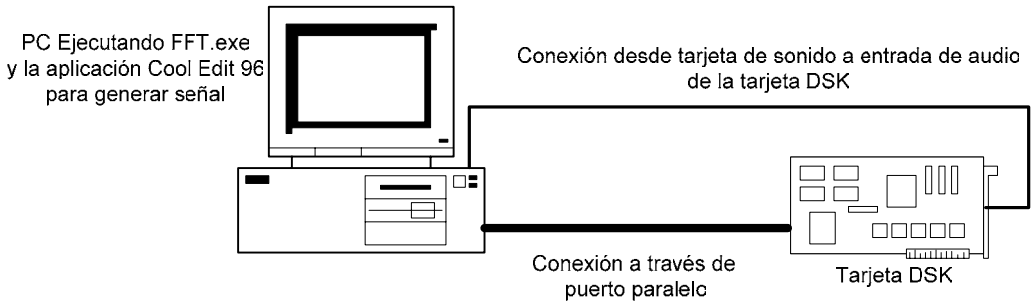
Durante el proceso de depuración, se corrigieron algunos problemas con la ubicación del despliegue en pantalla.

Otro aspecto importante, fue comprobar la correcta funcionalidad del software que se ejecuta en el DSK.

6.5.1 Pruebas

A continuación se describen diferentes pruebas que se realizaron, para verifica el funcionamiento del programa, para ello se utilizo la configuración mostrada en la figura 42.

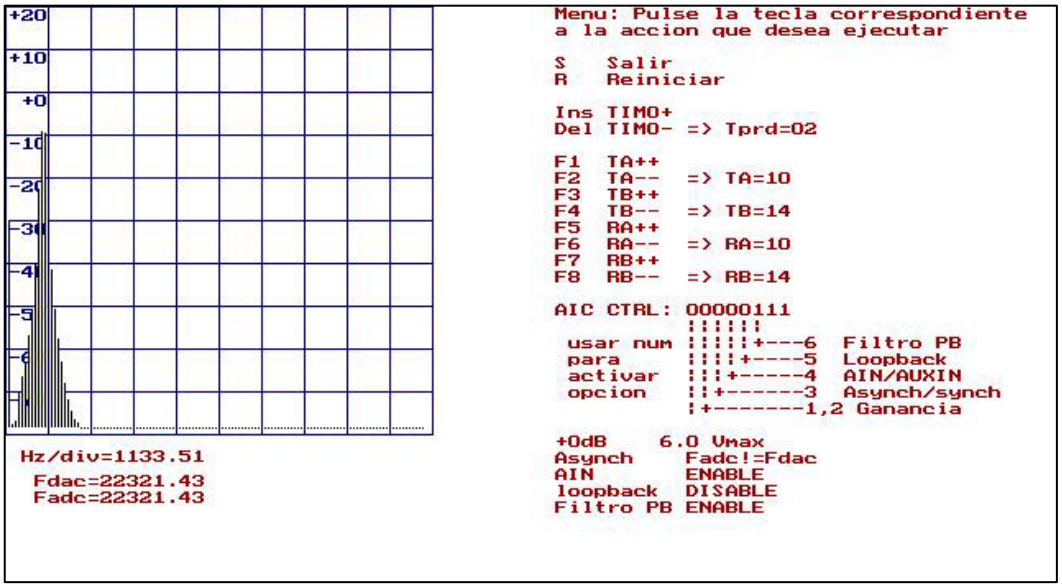
Figura 42 Configuración para ejecutar pruebas de medición de señales.



La Pc a la que se conecta el DSK, es la misma que se utilizo como generador de señales, a través del programa *Cool Edit 96*. Este software es un editor de audio, que tiene la opción de generar diferentes señales. Las señales generadas con el programa *Cool Edit*, fueron enviadas a través de la tarjeta de sonido de la PC, a la entrada de audio del DSK.

En las figuras siguientes, se muestran los resultados obtenidos, al enviar diferentes señales al DSP

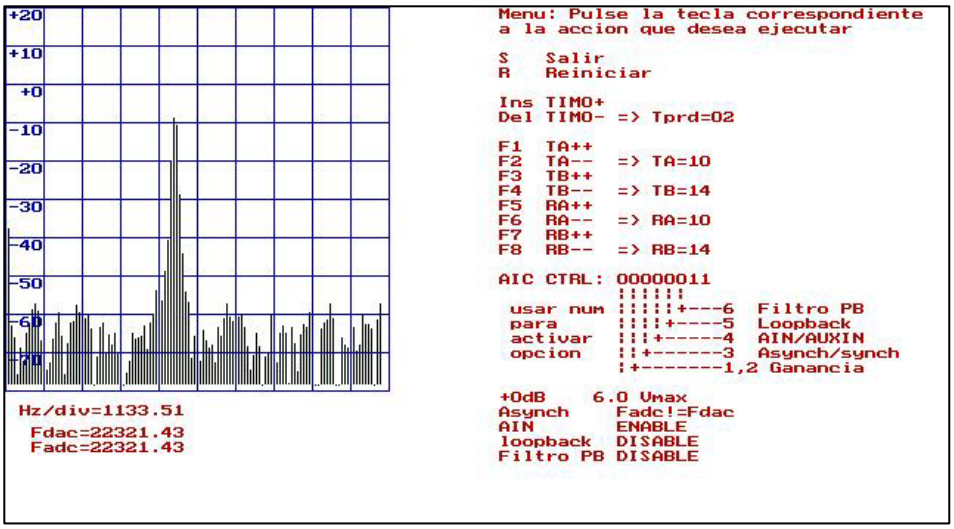
Figura 43. Espectro de frecuencia para una onda senoidal de 1 Khz



En la figura 43 puede observarse el espectro de frecuencia de una señal senoidal de 1 Khz, el espectro teórico esperado para este tipo de señal, es una línea (un componente de frecuencia) a 1 Khz. El resultado obtenido es una línea a aproximadamente 1 Khz (cada línea vertical de la pantalla de visualización tiene 1.13 Khz).

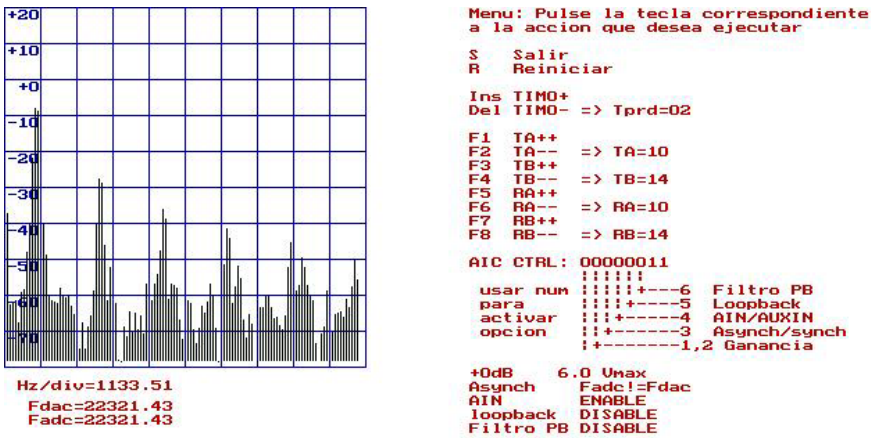
En la figura 44 se observa el resultado obtenido para una señal senoidal de 5 Khz, como es de esperarse, este espectro muestra un componente fundamental a 5 khz.

Figura 44. Espectro de frecuencia de una señal senoidal de 5 Khz.



A continuación se realizaron pruebas con señales triangulares, el espectro teórico esperado para este tipo de señales es un componente de frecuencia fundamental, y frecuencias armónicas múltiplos de la frecuencia original y cuya amplitud va disminuyendo. El resultado obtenido al analizar una señal triangular se muestra en las figuras 45.

Figura 45 Espectro de frecuencia de una señal triangular de 1 Khz



6.6 Programa ejecutable

Una vez que el programa ha sido depurado, y que el funcionamiento es el esperado, se ejecuta una nueva compilación que da como resultado un programa ejecutable, para ser utilizado en diferentes sistemas.

Para obtener el archivo ejecutable, se utiliza el comando *Make*, desde el entorno de programación turbo C++, este comando, combina todos los archivos del proyecto y produce el archivo FFT.EXE, que es el programa que puede ser ejecutado en cualquier computadora con sistema operativo Microsoft Windows.

El programa completo de interfase, se encuentra en el Apéndice II, de este trabajo de graduación.

CONCLUSIONES

1. El chip DSP TMS320C31, resulta apropiado para la implementación de un analizador de espectro para frecuencias de audio, ya que permite implementar una FFT de 256 puntos en 5 ms, maneja aritmética con datos de punto flotante lo que facilita la programación y dispone de 8 registros de propósito general, que permiten almacenar resultados temporales, lo que contribuye a que el programa sea más eficiente y se logre el análisis de frecuencias en tiempo real.

2. Es posible exceder las especificaciones del circuito de interfase analógica, de su frecuencia de muestreo nominal de 20 KHz, a una frecuencia de muestreo de 44 KHz, lo cual permite lograr el objetivo trazado de desarrollar un analizador de espectro que cubra el rango completo de frecuencias de audio.

3. La implementación del programa de control para la PC, fue posible, gracias a un conjunto de funciones en lenguaje C, incluidas como parte de las herramientas de desarrollo de software, del sistema DSK, estas funciones, permiten ejecutar operaciones de lectura y escritura entre la PC y el DSP. Sin estas funciones, la complejidad de la programación, hubiera hecho irrealizable el proyecto.

4. El analizador de espectro implementado en este trabajo de graduación, permite analizar el contenido de frecuencias de señales cuya frecuencia este entre 0 y 20 Khz. El elemento central de este analizador, es el dispositivo DSP, que ejecuta el cálculo de una FFT de 256 puntos para determinar los componentes de frecuencia de la señal bajo análisis. El trabajo se complementa con un programa diseñado para la PC, que permite controlar la operación del analizador y visualizar gráficamente, la amplitud de los componentes de frecuencia de la señal que se este analizando.

RECOMENDACIONES

1. El analizador de espectro implementado, tiene una funcionalidad básica, permitiendo visualizar el espectro de frecuencias de señales de audio y controlar la operación del mismo. Sin embargo, se pueden hacer algunas mejoras, como las siguientes. Aumentar el tamaño de la FFT, esto ayudaría a mejorar la resolución del analizador. Adicionalmente, sería conveniente disponer de algunas funciones más en el programa para la PC, como la posibilidad de almacenar los resultados de las señales, para una visualización posterior.

2. El sistema DSK utilizado en el desarrollo de este trabajo de graduación, puede ser utilizado para muchas otras aplicaciones, entre las cuales se puede mencionar, generación de señales, implementación de un osciloscopio para señales de baja frecuencia, generación de tonos. Los conceptos teóricos planteados en esta tesis, así como la bibliografía incluida, pueden ser valiosas referencias para los interesados en el desarrollo de este tipo de aplicaciones.

3. El desarrollo de un sistema que utiliza el procesamiento digital de señales, abarca dos grandes áreas. La primera es el conocimiento de las técnicas matemáticas apropiadas y los algoritmos que permiten implementar estas técnicas, en un sistema digital. La segunda es el conocimiento de los dispositivos DSP, y la manera de programar en ellos, los algoritmos necesarios. Las personas interesadas en desarrollar este tipo de sistemas, deben conocer ambas áreas para obtener el éxito.

4. El analizador de espectro implementado, se puede utilizar como base para el desarrollo de un sistema de apoyo, para mejorar la pronunciación de personas que tienen discapacidad auditiva. Una persona con problemas auditivos, no puede hablar bien, porque no puede escuchar su voz. Con el analizador de espectro esta persona, puede ver su pronunciación gráficamente, y compararla con la misma palabra pronunciada correctamente, de esta forma puede practicar, para mejorar su pronunciación. Para la implementación de un sistema como éste, debe mejorarse el programa proporcionado y trabajar estrechamente con una persona especializada en tratamiento de personas con discapacidades auditivas. Sería un aporte interesante y útil de la comunidad de ingeniería, para la ayuda de personas con discapacidades auditivas.

BIBLIOGRAFÍA

1. ADSP-21065L Ez-kit lite evaluation system manual. U.S.A: Analog Devices, 2000.

2. BRADLEY Jon, TMS320 Hardware applications. U.S.A: Texas Instruments, 1997.

3. DSP Selection Guide. U.S.A: Analog Devices, 2001.

4. PAPAMICHALIS, Panos. An implementation of FFT, DCT, and other transforms on the TMS320C30. U.S.A. Texas Instruments, 1997.

5. PERRY, Greg. C con Ejemplos. 1a. ed, Buenos Aires: Prentice Hall, 2000.

6. REEKIE, John. Realtime DSP:The TMS320C30 course. Australia: s.e. 1994.

7. SMITH, Steven W. The Scientist and Engineer's Guide to Digital Signal Processing. 2a Ed. U.S.A: California Technical Publishing, 1999.

8. TLC32040C, TLC32040I, TLC32041C, TLC32041I, Analog interface circuits. U.S.A: Texas Instruments, 1995

9. TMS320 DSP Product overview. U.S.A: Texas Instruments. 1998.

10 TMS320C3x DSP starter kit user's guide. U.S.A: Texas Instruments, 1996.

11. TMS320C3X General-purpose applications user's guide, U.S.A: Texas Instruments 1998.

12. TMS320C3x User's Guide. U.S.A: Texas Instruments, 1997.

APÉNDICE I

PROGRAMA PARA EL CÁLCULO DE LA FFT EN EL SISTEMA DSK

```
;FFT.ASM
;Programa que implementa una FFT de 256 puntos, en el DSP TMS320C31
;
;El programa comienza asignando valores a constantes que serán
;utilizadas en el desarrollo del programa, con el propósito de hacer
el ;programa más entendible, y poder modificar sus parámetros con
mayor ;fácilidad.
;
;La rutina inicial (INICIAL), inicializa los Timers y el puerto serial
;del TMS320C31, y además, inicializa el AIC (Analog Interface Circuit)
;que es el circuito de adquisición de datos y de conversión Analoga a
;Digital
;
;A continuación se ejecuta el programa principal, que básicamente
;funciona de la manera siguiente:
;
;se obtienen N muestras del ADC (N=256), y se almacenan en un buffer
;luego se ejecuta una FFT Radix-2 de 256 puntos, una vez almacenado el
;resultado estos datos se formatean adecuadamente (al hacer la
;convolución con una función de ventaneo) y el resultado se almacena
en ;memoria.
;Luego el DSP queda en estado de espera, y es el host quien decide si
;el programa continua leyendo más datos o término.
;
;
;En la ejecución del algoritmos para la FFT, se utiliza la técnica
;de decimado en frecuencia, a continuación, se muestra una mariposa
;típica, al utilizar esta técnica
;
;=====================================================
;
; a+jb  -------------------- TPR + jTPI
;             \  /
;              \/
;              /\
;             /  \ TWR+jTWI
; c+jd  -------------------- BTR + jBTI
;                -1
; Donde:
;     TPR = (a+c)
;     TPI = (b+d)
;     BTR = [(a-c)TWR - (b-d)TWI]
;     BTI = [(a-c)TWI + (b-d)TWR]
;     TWR = cos(x) (Factor "Twiddle" Parte Real)
;     TWI = sen(x) (Factor "Twiddle" Parte imaginaria)
;
;=====================================================
;
;-----------------------------------------------------
RAM0   .set   0x809800
RAM1     .set   0x809C00
;-----------------------------------------------------
TA     .set 10    ;
TB     .set 14    ;
RA     .set 10    ;
RB     .set   14  ;
       .include "C3XMMRS.ASM" ; Define los valores de los registros
                              ;mapeados por memoria
N        .set   256
N2       .set   N/2
```

```
PI       .set   3.1415926
PI2      .set   2*PI
PI2N     .set   2.0*PI/N
PIN      .set   PI/N
;=================================================
; Creación de Tabla con factores "Twiddle"
; y asignación de espacio para datos de E/S
;=================================================
         .start  "TWIDDLES",0X809800  ;
         .sect   "TWIDDLES"           ;
TWR                                   ; 0x809800
         .loop   N2                   ;
         .float  cos(br($-TWR,N)*PIN)  ;
         .endloop                     ;
TWI                                   ;
         .loop   N2                   ; 0x809880
         .float -1*sin(br($-TWI,N)*PIN);
         .endloop                     ;
;
RESULT   .set    TWI+N2                  ; 0x809900
DR       .set    RESULT+N                ; 0x809A00
DI       .set    DR+N                    ; 0x809B00
;*************************************************************
; Código de Inicialización

; Esta sección de código, solo se utiliza al inicio
; por lo que puede ser ensamblada en el área destinada
; para almacenamiento de datos, en este caso, se utiliza
; el espacio designado para almacenar el resultado (RESULT).
;*************************************************************
          .start   "INIT",RESULT    ; Colocar código en buffer de
datos
          .sect    "INIT"           ;
          .entry   INICIAL
INICIAL   ldp   T0_ctrl         ; Use kernel data page and stack
          ;----------------------
        ; Inicializar Timers
        ;----------------------
        ldi   0,R0            ; Parar TIM0 & TIM1
          sti   R0,@T0_ctrl     ;
          sti   R0,@T1_ctrl     ;
          sti   R0,@T0_count    ; Establecer cuenta a 0
          sti   R0,@T1_count    ;
          ldi   1,R0            ; Establecer periodo a 1
          sti   R0,@T0_prd      ;
          sti   R0,@T1_prd      ;
          ldi   0x2C1,R0        ; Reinicializar ambos timers
          sti   R0,@T0_ctrl     ;
          sti   R0,@T1_ctrl     ;
          ;-------------------------
        ; Inicializar puerto serial
        ;-------------------------
          ldi   @S0_xctrl_val,R0;
          sti   R0,@S0_xctrl        ; Cargar palabra de control de
                                    ;transmisión
          ldi   @S0_rctrl_val,R0;
          sti   R0,@S0_rctrl        ; Cargar palabra de control de
                                    ; recepción
          ldi   0,R0                ;
          sti   R0,@S0_xdata        ; valor de datos para DXR
```

```
          ldi   @S0_gctrl_val,R0   ;
          sti   R0,@S0_gctrl       ; Cargar palabra de control en
                                   ;regístro de control global
          ;---------------------
          call  AIC_INIT           ; Inicializar the AIC
          ldi   0x30,IE            ; Dar servicio a interrupciones
                                   ;RINT/XINT
          ldi   @S0_rdata,R0       ;
          b     inicio             ; Salto a programa principal
;*************************************************;
; Vectores para el manejo de interrupciones       ;
;*************************************************;
          .start   "SP0VECTS",0x809FC5
          .sect    "SP0VECTS"
          B     DAC            ; XINT0
          B     ADC            ; RINT0
;================================================
          .start  "FFTCODE",0x809C00          ;Define la sección de
                                              ;memoria RAM donde
          .sect   "FFTCODE"                   ;se ubicara el código.
;================================================
_STOP     .set   1
_START    .set   2
;
MSG_BOX   .word  _STOP             ; 0x809C00
TLVL      .word  1000              ; 0x809C01
A_REG     .word  (TA<<9)+(RA<<2)+0 ; 0x809C02
B_REG     .word  (TB<<9)+(RB<<2)+2 ; 0x809C03
C_REG     .word  00000011b         ; 0x809C04  +/- 1.5 V
EDGESEL   .word  1                 ; 0x809C05
SIZE      .word  N                 ; 0x809C06
MASK      .word  0xFFFFFFFF        ; 0x809C07  Mascara para los bits
de
                                   ; la mantisa
A_REGOLD  .word  0
B_REGOLD  .word  0
C_REGOLD  .word  0
;
S0_gctrl_val  .word  0x0E970300 ; Valor que habilita interrupciones de
                                ;entrada y salida
S0_xctrl_val  .word  0x00000111 ;
S0_rctrl_val  .word  0x00000111 ;
;
FFTSIZE   .word N                ; N=256 FFT de 256 puntos
TR_ADDR   .word TWR              ; TWR (128 valores de coseno)
TI_ADDR   .word TWI              ; TWI (128 valores de seno)
DR_ADDR   .word DR               ; 256 Valores parte real
DI_ADDR   .word DI               ; 256 Valores parte imaginaria
RS_ADDR   .word RESULT           ; Arreglo de 512 espacios, para
                                 ;almacenar resultado
TEMP      .word 0
;-------------------------------
inicio    ldi   0x30,IE
          ldi   @S0_rdata,R0     ; Limpiar SP under/overflow
          ldi   0,R0             ;
          sti   R0,@S0_xdata     ;
          ldi   @S0_rdata,R0     ;
          ldi   0,R0             ;
          sti   R0,@S0_xdata     ;
          sti   R0,@RAMP
```

```
        ldi   25,RC           ; Precargar algunos datos del ADC para
                              ; limpiar
          rptb  precarga      ; el AIC despues de cierto tiempo de
                              ; inactividad
precarga  call  GETADC          ;
        ;---------------------
        ldi   @DR_ADDR,AR0    ; AR0 = DR[0]
          ldi   @DI_ADDR,AR1  ; AR1 = DI[0]
          ldi   @FFTSIZE,RC   ; Obtener datos
          subi  1,RC          ; N+1 repeticiones
        rptb muestreo
        ;---------------------
        call  GETADC            ;
          float R0,R0           ;
          stf   R0,*AR0++       ; Almacenar en DR
          ldf   0,R0            ;
muestreo  stf   R0,*AR1++       ; Llenar arreglo DI con ceros
          ;---------------------
        ldi   0,R0              ; Escribir 0 en DXR cuando no
          sti   R0,@S0_xdata    ; se utilizará por algun tiempo
;=======================================================
; Calcular FFT
;------------------------------
FFT:      ldi   @FFTSIZE,IR0    ;
          ldi   @FFTSIZE,IR1    ;
          lsh   -1,IR0
LOOP:
          ldi   @FFTSIZE,RC     ; RC = 256
          ldi   @DR_ADDR,AR0    ; AR0 = DR[0]
          ldi   @DI_ADDR,AR1    ; AR1 = DR[128]
          ldi   @TR_ADDR,AR2    ; AR2 = TWR[0]
          ldi   @TI_ADDR,AR3    ; AR1 = TWI[0]
          lsh   -1,RC           ; RC = 128 butterfly loops
          subi  1,RC            ;
          lsh   -1,IR0          ; IRO e IR1 indexan el factor twiddle
          lsh   -1,IR1          ; que se accesa
          ldi   IR1,R0          ; en el inicio de una nueva etapa
          bz    FFT_END         ;
          ;---------------------
Blk_Top   rptb  B_Fly           ; Obtener los 6 datos de entrada para
                                ; calcular mariposa
          ldf   *+AR0(IR1)  ,R0 ; Carga c
    ||    ldf   *AR0        ,R1 ; Carga a
          ldf   *+AR1(IR1)  ,R2 ; carga d
    ||    ldf   *AR1        ,R3 ; carga b
          ldf   *AR2++(IR0) ,R4 ; Cargar TWR
    ||    ldf   *AR3++(IR0) ,R5 ; Cargar TWI

        and   @MASK,R0
        and   @MASK,R1
        and   @MASK,R2
        and   @MASK,R3
        and   @MASK,R4
        and   @MASK,R5

          addf3 R0,R1,R6         ; (a+c)
          addf3 *+AR1(IR1),R3,R7 ; (b+d)
    ||    stf   R6,*AR0          ;

          subf3 R0,R1,R6        ; R6=(a-c)
```

```
          stf   R7,*AR1         ;

          subf3 R2,R3,R7        ; R7=(b-d)  (R3 free)

          mpyf3 R6,R4,R1        ; R1 = (a-c)*TWR = REAL_1
          mpyf3 R7,R5,R3        ; R3 = (b-d)*TWI = REAL_2
          subf  R3,R1           ; R1 = BTR = [(a-c)TWR - (b-d)TWI]
          stf   R1,*+AR0(IR1)   ; Almacenar BTR
          nop   *++AR0
          mpyf3 R6,R5,R1        ; R1 = (a-c)*TWI = IMAG_1
          mpyf3 R7,R4,R3        ; R3 = (b-d)*TWR = IMAG_2
          addf  R3,R1           ; R1 = BTI = [(a-c)TWI + (b-d)TWR]
          stf   R1,*+AR1(IR1)   ; Almacenar BTI
          nop   *++AR1
          ;------------------------------
          ; Identificar fin de la mariposa
          ;------------------------------
ident     ldi   @TR_ADDR,R7     ;
          subi  AR2,R7          ;
          ldiz  IR1,R7          ;
          ldinz 0,R7            ;
          addi  R7,AR0          ;
B_Fly     addi  R7,AR1          ; Continuar el loop hasta terminar
                                ; todas las mariposas
          b     LOOP            ; Cuando RC=0, iniciar nueva etapa
;-------------------------------------------------------------------------
-

FFT_END   ldi @RS_ADDR,AR0    ; Puntero a arreglo donde se almacenara
                              ; resultado
        ldi @DR_ADDR,AR1      ; Puntero a arreglo donde se almacena
                              ; parte real
        ldi @DI_ADDR,AR4      ; Puntero a arreglo donde se almacena
                              ; parte imaginaria
        ldi @FFTSIZE,IR0      ; IR0=256
        lsh -1,IR0            ; IR0=128
        ldi IR0,R0            ;
        subi 1,R0             ;
        or  IR0,R0            ;
        addi R0,AR1           ;
        addi R0,AR4           ;
        sti  R0,@BRINDX_N     ;
        sti  IR0,@BRINDX_P    ;
        ;---------------------
        ldi  AR1,AR2          ; AR1 = REAL[N-1]
        ldi  AR4,AR5          ; AR4 = IMAG[N-1]
        ;======================================
        ldi  @SIZE,RC         ;
        lsh  -2,RC            ; Empacar 4 resultados por palabra de
                              ; memoria RC=64 Iteraciones
        rptb WINDOW           ;
        ;----------------------
        ldi  0,R7       ; El llamado a la rutina Log_Mag, retorna con
                        ; R^2 + I^2
        call Log_Mag          ; filtrado y empacado en los 8 MSBs
        lsh  -24,R0           ; Desplazar y empacar 4 resultados por
                              ; palabra
        or   R0,R7            ;
        call Log_Mag          ;
        lsh  -16,R0           ;
```

```
        or   R0,R7           ;
        call Log_Mag         ;
        lsh  -8 ,R0          ;
        or   R0,R7           ;
        call Log_Mag         ;
        lsh  0  ,R0          ;
        or   R0, R7          ;
WINDOW    sti  R7,*AR0++     ; Almacenar el resultado empacado
        ;---------------------
        ldi  0x4,IE          ;
        ldi  _START,R0  ;
NO_START  cmpi @MSG_BOX,R0   ;
        bnz  NO_START        ;
        ldi  _STOP,R0        ;
        sti  R0,@MSG_BOX     ;
        ;=================================
        ;Cargar nuevos parametros para el AIC
        ;=================================
        ldi  0,R2       ;
        ldi  @A_REG,R0  ;
        cmpi @A_REGOLD,R0    ;
        ldinz 1,R2           ;
        sti  R0,@A_REGOLD    ;
        ;---------------------
        ldi  @B_REG,R0  ;
        cmpi @B_REGOLD,R0    ;
        ldinz 1,R2           ;
        sti  R0,@B_REGOLD    ;
        ;---------------------
        ldi  @C_REG,R0  ;
        cmpi @C_REGOLD,R0    ;
        ldinz 1,R2           ;
        sti  R0,@C_REGOLD    ;
        ;---------------------
        cmpi 0,R2       ;
        bz   inicio          ;
        call AIC_INIT        ; Reiniciar con nuevos valores para AIC
        b    inicio          ; Volver a ejecutar el calculo
;=====================================================================
=
; Log_Mag: Esta función calcula R^2 + I^2 y almacena el resultado
;          en los 8 MSBs (los 24 LSBs son 0), ademas ejecuta la
función
;        de ventaneo y calcula el logaritmo del resultado
;=====================================================================
=
BRINDX_N  .word   0              ; BR index usado para moverse a
traves de la data
BRINDX_P  .word   0              ;
VU_scale  .float  1/(N*128.0)    ; Factor de escala para la FFT
;- - - - - - - - - - - - - - - - -
Log_Mag   push  R2               ; 26 pixels/10 dB
          push  R4               ; 0.75000 V = 0 dBm
          push  R5               ;
          pushf R2               ;
          pushf R4               ;
          pushf R5               ;
          ldf   -0.500,R5        ;
          ;- - - - - - - - - - - -
          mpyf3 R5,*AR1++(IR0)B,R0;
```

```
          addf     *AR1++(IR0)B,R0;
          mpyf3 R5,*AR1        ,R2;
          addf  R0,R2             ;
          mpyf  @VU_scale,R2      ;
          mpyf  R2,R2             ; REAL^2
          ;- - - - - - - - - - - - -
          mpyf3 R5,*AR4++(IR0)B,R0;
          addf     *AR4++(IR0)B,R0;
          mpyf3 R5,*AR4        ,R4;
          addf  R0,R4             ;
          mpyf  @VU_scale,R4      ;
          mpyf  R4,R4             ; IMAG^2
          ;- - - - - - - - - - - - -
          ldi   @BRINDX_N,IR0     ;
          nop   *AR1++(IR0)B      ;
          nop   *AR4++(IR0)B      ;
          ldi   @BRINDX_P,IR0     ; Indexado normal
          ;- - - - - - - - - - - - -
          addf  R4,R2             ; REAL^2 + IMAG^2
          ;- - - - - - - - - - - - -
          lsh   1,R2              ; logaritmo rapido usando
equivlancia
                                  ; de punto flotante
          pushf R2                ;
          pop   R0                ;
          ;- - - - - - - - - - - - -
          ash   -21,R0            ; Empacar resultado en los  8 MSBs
          cmpi  -128,R0           ; Preservando 3 bits de mantisa
          ldile -128,R0           ; y recortando el resultado
          cmpi   127,R0           ;
          ldige  127,R0           ;
          lsh    24,R0            ; Resultado en los 8 MSBs, 24 LSBs
                                  ; son 0
          ;- - - - - - - - - - - - -
          popf  R5                ;
          popf  R4                ;
          popf  R2                ;
          pop   R5                ;
          pop   R4                ;
          pop   R2                ;
          rets                    ;
;*************************************************************************
*
RAMP      .word 0
FLAGS     .word 0

GETADC    ldi   0x30,IE         ; Esperar por confirmación de
Interrupción del ADC
          IDLE                  ; para salvar potencia y espacio de
                                ; código
          ldi   @FLAGS,R0       ;
          tstb  0x20,R0         ;
          bz    $-3             ;
          andn  0x20,R0         ;
          sti   R0,@FLAGS       ;
          ldi   @S0_rdata,R0    ; Retornar valor ADC de signo
extendido
          lsh   16,R0           ;
          ash   -16,R0          ;
          rets
```

```
ADC       push  ST              ;
          push  R0              ;
          ldi   @S0_rdata,R0    ;
          ldi   @FLAGS,R0       ;
          or    0x20,R0         ;
          sti   R0,@FLAGS       ;
          pop   R0              ;
          pop   ST              ;
          reti                  ;

DAC       push  ST              ;
          push  R1              ;
          ldi   @RAMP,R1        ;
          subi  1024,R1         ;
          lsh   17,R1
          ash   -17,R1
          andn  3,R1            ;
          sti   R1,@RAMP        ;
          sti   R1,@S0_xdata    ; loopback ADC->DAC
          pop   R1
          pop   ST
          reti
;--------------------------------
prog_AIC  push  R1              ;
          push  IE              ;
          ldi   0x10,IE         ;
          andn  0x30,IF         ;
          ldi   @S0_xdata,R1    ; Usar DXR origina durante
comunicación
                                ; secundaria
          sti   R1,@S0_xdata    ;
          idle
          ldi   @S0_xdata,R1    ; Usar DXR origina durante
comunicación
                                ; secundaria
          or    3,R1            ; Solicitar comunicación secundari
XMIT
          sti   R1,@S0_xdata    ;
          idle                  ;
          sti   R0,@S0_xdata    ; Enviar el valor de registro
          idle                  ;
          andn  3,R1            ;
          sti   R1,@S0_xdata    ;
          pop   IE              ;
          pop   R1              ;
          rets                  ;
;=======================================================;
; Esta sección de código, es llamada por el código      ;
; de inicialización, y también por el programa principal  ;
; por lo tanto, se ensambla en la RAM regular de programa ;
;=======================================================;
AIC_INIT  push  R0              ;
          LDI   0x10,IE         ; Habilitar interrupción XINT
          andn  0x34,IF         ;
AIC_reset
          ldi   0,R0            ;
          sti   R0,@S0_xdata    ;
          RPTS  0x040           ;
          LDI   2,IOF           ; XF0=0 resetea AIC
```

```
rpts  0x40            ;
LDI   6,IOF           ; XF0=1 ejecutar AIC
ldi   @S0_rdata,R0
ldi   0,R0
sti   R0,@S0_xdata
;-----------------------------
ldi   @C_REG,R0       ; Establecer registro de control
call  prog_AIC        ;
ldi   0xfffc  ,R0     ; Programar el AIC a velocidad lenta
call  prog_AIC        ;
ldi   0xfffc|2,R0     ;
call  prog_AIC        ;
ldi   @B_REG,R0       ; Empujar el Fs hasta velocidad final
call  prog_AIC        ;
ldi   @A_REG,R0       ;
call  prog_AIC        ;
pop   R0              ;
ldi   0,R0            ; Poner un 0 in DXR
sti   R0,@S0_xdata    ;
ldi   @S0_rdata,R0    ; Clear receive underrun
rets                  ;
```

APÉNDICE II

PROGRAMA DE CONTROL PARA PC

```
//*********************************************************************
//FFT Versión 0.5
//Programa para despliegue y control del analizador de espectro
//
//En el programa, se carga y ejecuta la aplicación DSK
//y se lee de manera continua la memoria del DSP
//desplegando en forma gráfica los resultados.
//Este archivo se debe compilar junto con las siguientes aplicaciones
//
//    fft01.cpp         Este archivo
//    driver.cpp        Drivers de la impresora de bajo nivel
//    target.cpp        Comandos a nivel del DSK
//    object.cpp        Rutinas para cargar aplicaciones al DSK
//    dsk_coff.cpp      Utilerías para carga de archivos DSK y COFF
//    errormsg.cpp      Mensajes de retorno de la mayoría de funciones
//    symbols.cpp       Tabla de símbolos (necesarias para dsk_coff)
//    textwin.cpp       Funciones de ventana de texto a nivel de DOS
//    egavga.obj        drivers para gráficos EGA y VGA
//
//Para compilar el programa, se utilizo Turbo C++ versión 3.0 de --
//Borland.
//*********************************************************************
//
//-----------------------------------------------------
#include <dos.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#include <ctype.h>
#include <bios.h>
#include <math.h>
#include <string.h>
#include "DSK.H"
#include "DSK_COFF.H"
#include "C3XMMRS.H"
#include "keydef.h"
//-----------------------------------------------------

#define FFTSize  256         //El tamaño de la FFT a calcular es de
256 puntos
#define MSG_DSP    0x809C00L      //Palabra de control enviada al DSK
#define MEMDSP (0x809800L + FFTSize)     //Bloque de memoria a leer
//#define A_BOX    =  0x809802L;
//#define B_BOX    =  0x809803L;
//#define C_BOX    =  0x809804L;
ulong A_BOX = 0x809C02L;
ulong B_BOX = 0x809C03L;
ulong C_BOX = 0x809c04L;
#define THx 40e-9      //Tiempo de ejecución del DSP
#define MASK       =  0x809807L;
```

```
#define  graph_vwport()       setviewport(0, 0, 563, 340, 1)
#define  menu_vwport()        setviewport(334, 0, 639, 340, 1)
#define  A_REG  ((TA<<9)+(RA<<2)+0)//Divisor A, establece la velocidad
                                   //de SCF
#define  B_REG  ((TB<<9)+(RB<<2)+2)//Divisor B
int Samples = 256;
int Step = 2;
char DSK_APP[] = "FFT25610.DSK";
char DSK_EXE[] = "FFT05.EXE";

ulong T0_prdv = 0x00000001L;
ulong Zero    = 0x00000000L;
float Hz_por_div = 0.0;
float Fsr = 1000.0;
float Fsx = 1000.0;
int   TA = 10; //Divisores del DAC
int   TB = 14;
int   RA = 10; //Divisores del DAC
int   RB = 14;

typedef enum messages
{
  DETENER = 1,
  INICIAR = 2
} message;

int  C_REG = 0x03;     //Bits del registro de control del AIC

int buf_1[512];
char buf_0[512];
int menu(void);
void init_graphics(void);
void binsprintf(char *s,int val);

//-----------------------------------------------
//draw_vect() función para graficar el resultado
///-----------------------------------------------
void draw_vect()
{
  int x, y;
  int *old;
  char *ptr0;
  char *tmp0;
  tmp0 = buf_0;
  ptr0 = tmp0;
  setcolor(WHITE);
  old = buf_1;
  setwritemode(1);
  for (x=0;x<FFTSize;x+=2)
  {
    y = 128 - *ptr0++;
```

```
    if (y > *old) line(x,*old+1,x,y);
    if (y < *old) line(x,*old,x,y+1);
    *old++ = y;
  }
  setfillstyle(SOLID_FILL, BLACK);
  bar(0,261,256,268);
}
//-----------------------------------------------------
//Programa principal
//-----------------------------------------------------
void main(void)
{
  int reset = 0;
  int nparam = 1;
  ulong MSG=INICIAR;
  ulong aic;
//  ulong TEMP;
  MSGS err;
  float sum;
  int x;
  Scan_Command_line(DSK_EXE);
  //-----------------------------------------------------
  for(;;)
  { if (Init_Communication(10000) == NO_ERR) break;
    if (kbhit()) exit(0);
  }
  HALT_CPU; //Detener cualquier aplicación ejecutándose previamente en
            //el DSP
  if ((err = Load_File(DSK_APP, LOAD)) != NO_ERR)
  {  printf("%s %s\n", DSK_APP, Error_Strg(err));
     exit(0);
  }
  if ((err = Load_File(DSK_APP, SLOAD)) != NO_ERR)
  {  printf("%s %s\n", DSK_APP, Error_Strg(err));
  }
  RUN_CPU();      //Ejecutar aplicación FFT256.DSK, en el DSP
  init_graphics();
  //
  //
  for (x=0;x<256;x++)
  {
    buf_1[x] = 255;
    buf_0[x] = 0;
  }
  menu();
  draw_vect();

  //-----------------------------------------------------
  for (;;)
  {
//    draw_vect();
```

```
    HPI_STRB(0);
    reset=0;
    for (;;)
    {
      if (kbhit())
      {
       reset=menu();
       nparam=1;
      }
      if (HPI_ACK()) break;
      delay(1);
    }
    if (reset) break;

    //Si se presiona una tecla desde la pantalla del programa
    //Reprogramar el AIC, de acuerdo a los nuevos valores

    if (nparam)
    {
     // TEMP = Samples;
//    if (putmem(SAMPLES, 1,&TEMP)!=NO_ERR) break;
      if (putmem(T0_prd  ,1,&T0_prdv)!=NO_ERR) break;
      if (putmem(T0_count,1,   &Zero)!=NO_ERR) break;
      if (putmem(T1_prd  ,1,&T0_prdv)!=NO_ERR) break;
      if (putmem(T1_count,1,   &Zero)!=NO_ERR) break;
      if (TB>TA)
      {
      aic = A_REG;
      if (putmem(A_BOX,1,&aic)!=NO_ERR) break;
      aic = B_REG;
      if (putmem(B_BOX,1,&aic)!=NO_ERR) break;
      }
      else
      {
      aic = B_REG;
      if (putmem(A_BOX,1,&aic)!=NO_ERR) break;
      aic = A_REG;
      if (putmem(B_BOX,1,&aic)!=NO_ERR) break;
      }
      aic = C_REG;
      if (putmem(C_BOX,1,&aic)!=NO_ERR) break;
    }
    nparam = 0;

    if (getmem(MEMDSP, FFTSize/8,(ulong *)buf_0)!=NO_ERR) break;
    putmem(MSG_DSP,1,&MSG);
    draw_vect();
//    Fsx = 1/(2*TA*TB*(2*THx*T0_prdv));
//    if (C_REG & 0x20) Fsr = Fsx;
//    else Fsr = 1/(2*RA*RB*(2*THx*T0_prdv));
  }
```

```
  closegraph();
}
//-----------------------------------------------
//clip(), es usado para confinar a limites
//superior e inferior un numero
//-----------------------------------------------
long inline clip(long x, int min, int max)
{
  if(x<min) return min;
  if(x>max) return max;
  return x;
}

//-----------------------------------------------
//menu() La función menú, se utiliza para
//desplegar el menú de opciones en pantalla
//de acuerdo a la opción seleccionada
//ejecuta la acción correspondiente
//-----------------------------------------------
int menu(void)
{
 int key=0;
 int Yt=0;
 char buf[80];
 static int old_key=0;
 static int accel = 1;
 if(kbhit()) key = bioskey(0) & 0xFF00;
 if (old_key==key) accel = accel + 1;
 else accel = accel/4;
 accel = clip(accel,1,200);
 old_key = key;

 if(key)
 {
  switch(key)
  {
    case _R  : return 1;      //Resetear el dsk
    case _S  : closegraph();
               _setcursortype(_NORMALCURSOR);
               exit(0);
               break;
    case _F1 : TA++; break;
    case _F2 : TA--; break;
    case _F3 : TB++; break;
    case _F4 : TB--; break;
    case _F5 : RA++; break;
    case _F6 : RA--; break;
    case _F7 : RB++; break;
    case _F8 : RB--; break;

    case _1  : C_REG = C_REG ^ 0x80; break; //bit 7
```

```
    case _2  : C_REG = C_REG ^ 0x40; break; //bit 6
    case _3  : C_REG = C_REG ^ 0x20; break; //bit 5
    case _4  : C_REG = C_REG ^ 0x10; break; //bit 4
    case _5  : C_REG = C_REG ^ 0x08; break; //bit 3
    case _6  : C_REG = C_REG ^ 0x04; break; //bit 2

    case _Ins : T0_prdv -= accel; break;
    case _Del : T0_prdv += accel; break;

    default  : return 0;
  }
  TA = clip(TA, 3, 31);
  TB = clip(TB, 12, 63);
  RA = clip(RA, 3, 31);
  RB = clip(RB, 12, 63);
  T0_prdv = clip(T0_prdv, 1, 64);
 }
 setwritemode(0);
 menu_vwport();
 clearviewport();
 setcolor(11);
 Yt=1;
 #define COL1 1
 outtextxy(COL1,Yt,"Menu: Pulse la tecla correspondiente");
 outtextxy(COL1,Yt+=10,"a la acción que desea ejecutar");
 outtextxy(COL1,Yt+=10,"");
 outtextxy(COL1,Yt+=10 ,"S   Salir");
 outtextxy(COL1,Yt+=10,"R   Reiniciar");

// outtextxy(COL1,Yt+=10,buf);

 Yt+=10;
 sprintf(buf,"Ins TIM0+"); outtextxy(COL1,Yt+=10,buf);
 sprintf(buf,"Del TIM0- => Tprd=%02d",T0_prdv);
outtextxy(COL1,Yt+=10,buf);

 Yt+=10;
 sprintf(buf,"F1  TA++");  outtextxy(COL1,Yt+=10,buf);
 sprintf(buf,"F2  TA--  => TA=%02d",TA);  outtextxy(COL1,Yt+=10,buf);
 sprintf(buf,"F3  TB++");  outtextxy(COL1,Yt+=10,buf);
 sprintf(buf,"F4  TB--  => TB=%02d",TB);  outtextxy(COL1,Yt+=10,buf);
 sprintf(buf,"F5  RA++");  outtextxy(COL1,Yt+=10,buf);
 sprintf(buf,"F6  RA--  => RA=%02d",RA);  outtextxy(COL1,Yt+=10,buf);
 sprintf(buf,"F7  RB++");  outtextxy(COL1,Yt+=10,buf);
 sprintf(buf,"F8  RB--  => RB=%02d",RB);  outtextxy(COL1,Yt+=10,buf);

 Yt+=10;
 sprintf(buf,"AIC CTRL:
");binsprintf(buf+10,C_REG);outtextxy(COL1,Yt+=10,buf);
 sprintf(buf,"          ||||||");
outtextxy(COL1,Yt+=10,buf);
```

```
 sprintf(buf," usar num |||||+---6  Filtro PB");
outtextxy(COL1,Yt+=10,buf);
 sprintf(buf," para     ||||+----5  Loopback");
outtextxy(COL1,Yt+=10,buf);
 sprintf(buf," activar  |||+-----4  AIN/AUXIN");
outtextxy(COL1,Yt+=10,buf);
 sprintf(buf," opcion   ||+------3  Asynch/synch");
outtextxy(COL1,Yt+=10,buf);
 sprintf(buf,"          |+-------1,2 Ganancia");
outtextxy(COL1,Yt+=10,buf);

 Yt+=10;
 switch (C_REG&0xC0)
 {
   case 0x00:  outtextxy(COL1,Yt+=10,"+0dB    6.0 Vmax"); break;
   case 0x40:  outtextxy(COL1,Yt+=10,"+6dB    3.0 Vmax"); break;
   case 0x80:  outtextxy(COL1,Yt+=10,"+12dB   1.5 Vmax"); break;
   case 0xC0:  outtextxy(COL1,Yt+=10,"+0dB    6.0 Vmax"); break;
 }
 if (C_REG&0x20) outtextxy(COL1,Yt+=10,"Synch     Fadc==Fdac");
 else            outtextxy(COL1,Yt+=10,"Asynch    Fadc!=Fdac");
 if (C_REG&0x10) outtextxy(COL1,Yt+=10,"AUXIN     ENABLE");
 else            outtextxy(COL1,Yt+=10,"AIN       ENABLE");
 if (C_REG&0x08) outtextxy(COL1,Yt+=10,"loopback  ENABLE");
 else            outtextxy(COL1,Yt+=10,"loopback  DISABLE");
 if (C_REG&0x04) outtextxy(COL1,Yt+=10,"Filtro PB ENABLE");
 else            outtextxy(COL1,Yt+=10,"Filtro PB DISABLE");

 Yt+=10;

 graph_vwport();
 Fsx = 1/(2*TA*TB*(2*THx*T0_prdv));
 if (C_REG & 0x20) Fsr = Fsx;
 else Fsr = 1/(2*RA*RB*(2*THx*T0_prdv));
// Hz_por_div = (520.0/512.0)*Fsr/(2.0*10);
 Hz_por_div = Fsr/(2.0*10);

 setfillstyle(SOLID_FILL, BLACK);
 bar(0,269,260,310);

 sprintf(buf, "Hz/div=%7.2f",Hz_por_div); outtextxy(10,270,buf);
 sprintf(buf, " Fdac=%7.2f",Fsx);         outtextxy(10,285,buf);
 if (C_REG&0x20)
   sprintf(buf," Fadc=Fdac");
 else
   sprintf(buf," Fadc=%7.2f",Fsr);
 outtextxy(10,295,buf);

 return 0;
}
//---------------------------------------------
```

```
//init_graphics()
//Rutina para la inicialización del modo grafico
//-----------------------------------------------
void init_graphics(void)
{
  int gdriver = EGA;
  int gmode = EGAHI;
  int errorcode;
  int Y;
  int X;
  errorcode = registerbgidriver(EGAVGA_driver);
  if (errorcode < 0)
  {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    exit(1);
  }
  initgraph(&gdriver, &gmode, "");
  errorcode = graphresult();
  if (errorcode != grOk)
  { printf("Graphics error: %s\n", grapherrormsg(errorcode));
    exit(1);
  }
  clearviewport();
  setcolor(BLUE);
  for (Y=0;Y<=260;Y+=26) line(0,Y,260,Y);
  for (X=0;X<=260;X+=26) line(X,0,X,260);

  Y = 2;
  X = 2;
  outtextxy(X,Y,"+20");
  outtextxy(X,Y+=26,"+10");
  outtextxy(X,Y+=26," +0");
  outtextxy(X,Y+=26,"-10");
  outtextxy(X,Y+=26,"-20");
  outtextxy(X,Y+=26,"-30");
  outtextxy(X,Y+=26,"-40");
  outtextxy(X,Y+=26,"-50");
  outtextxy(X,Y+=26,"-60");
  outtextxy(X,Y+=26,"-70");

  setwritemode(1);
  setcolor(15);
}

//-----------------------------------------------
//binsprintf()
//-----------------------------------------------
void binsprintf(char *s,int val)
{
  char *p;
  unsigned int t;
```

```
  p = s;
  t = 0x80;
  for (;t>0;t=t>>1)
  {
    if(val & t) *p++ = '1';
    else *p++ = '0';
  }
  *p=0;
}
```