PROTOTIPO DE UNIDAD DE PROCESAMIENTO BASADA EN DSP PARA UN MONITOR DE LA CALIDAD DE LA ENERGÍA ELÉCTRICA

OSCAR LEONARDO QUINTERO YANES

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECANICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2004.

PROTOTIPO DE UNIDAD DE PROCESAMIENTO BASADA EN DSP PARA UN MONITOR DE LA CALIDAD DE LA ENERGÍA ELÉCTRICA

OSCAR LEONARDO QUINTERO YANES

Este proyecto es presentado como requisito para optar al título de Ingeniero Electrónico

Director
CÉSAR ANTONIO DUARTE GUALDRÓN
Magíster en Potencia Eléctrica (MPE)

Codirector
JESÚS DAVID ANGULO
Ingeniero Electricista.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECANICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2004.

DEDICATORIA

Este gran paso dado en mi etapa de formación personal y académica se lo dedico a mis padres, Gilberto Quintero Becerra y Marlene Yanes de Quintero; quienes con mucho esfuerzo y sacrificio me dejan la única herencia valiosa en la vida, el estudio y la honradez.

Este trabajo también esta dedicado a mis hermanos, mis familiares y mis amigos; a quienes siempre llevo en mi mente y quiero mucho.

AGRADECIMIENTOS

Mucha gente hizo posible este triunfo en mi vida, y si empiezo a nombrarlos a cada uno por separado podría caer en el error de olvidar a alguien. Por eso, les doy muchísimas gracias a toda mi familia, amigos y compañeros que estuvieron siempre a mi lado apoyándome, dándome fuerzas cuando lo necesitaba y a pesar de los problemas siempre creyeron en mí.

Un agradecimiento especial, en particular, a:

- Dios, por dejarme vivir este momento.
- César Antonio Duarte Gualdrón; por su guía, consejos, paciencia y conocimiento para desarrollar este proyecto.
- Jesús David Angulo Acero y Valdomiro Vega; quienes me apoyaron en la culminación de este trabajo.
- Erika Rocío Quintero Yanes; quien me animo cuando más lo necesitaba y siempre creyó en mí.
- Julio Mario Durán Ramírez; por su compañía durante todo este tiempo y por haberme enseñado el verdadero significado de la amistad y la nobleza.
- Gustavo Sánchez, Wilson Castillo, Jorge Arturo Corso, Elker Patiño, Luis Lozano, Carlos Echeverría, Julián Sánchez, César Tovar, Juan Carlos López, Sandra Milena Balcarcel, Renata Bravo, Paola Sanguíno, Vikter Ortiz, Nataly Jaimes, Andrea Ramírez, Carolina González, Gerardo Sierra, Yasmith Agudelo y Jimena Galvis; por todos los momentos alegres que compartimos y su valiosa amistad.

RESUMEN

TITULO:

PROTOTIPO DE UNIDAD DE PROCESAMIENTO BASADA EN DSP PARA UN MONITOR DE LA CALIDAD DE LA ENERGÍA ELÉCTRICA*

AUTOR:

OSCAR LEONARDO QUINTERO YANES**

PALABRAS CLAVE: **Calidad de la energía eléctrica, monitorización, tiempo real, armónicos, transformada discreta de Fourier (*DFT*), transformada rápida de Fourier (*FFT*), valor *rms*, DSP, TMS320LF2407A, *Hanning*, *Hamming*, *Blackman*.**

DESCRIPCIÓN:

En este documento se explica cómo se implementaron los algoritmos en tiempo real para la monitorización de armónicos de una señal sintetizada o analógica. Esta monitorización se realiza a través del módulo de evaluación TMS320LF2407 EVM de *Texas Instruments*, y consta de tres etapas: digitalización de la señal, procesamiento y visualización de los resultados.

Inicialmente se describe el *hardware* utilizado para la implementación del prototipo. Por lo tanto, se describe en forma general el *DSP* TMS320LF2407A, su módulo de evaluación, la memoria y los periféricos del *DSP* utilizados en la implementación.

En el Capítulo 2 se describe el software de programación del *DSP*, el *Code Composer Studio* y las herramientas del lenguaje *assembly*.

El Capítulo 3 describe y explica lo referente a la emulación en tiempo real. Esta es una herramienta de depuración del *Code Composer Studio*, la cual permite acceder a las memorias internas y externas del *DSP* mientras se ejecuta el programa, entre otras cosas.

En el Capítulo 4 se explica brevemente lo referente a la monitorización de la calidad de la energía eléctrica, enfocado hacia la monitorización de eventos en estado estacionario, específicamente de armónicos. También se presenta lo referente a los equipos de monitorización de calidad, su clasificación y requerimientos.

En el Capítulo 5 se analiza la transformada discreta de Fourier (*DFT*), su funcionamiento y método usado para calcularla. El método utilizado fue la transformada rápida de Fourier (*FFT*).

El Capítulo 6 explica cómo se realizó la implementación. Por lo tanto, se comienza por una descripción del prototipo implementado y se da el diagrama de flujo del algoritmo. Luego se especifican los requerimientos ya propuestos. Por último se describe la interfaz usada para la visualización y se presentan los resultados obtenidos.

---

* Trabajo de grado

** Facultad de Ingenierías Fisicomecánicas, Ingeniería Electrónica, César Antonio Duarte Gualdrón.

SUMMARY

TITLE:

PROTOTYPE OF A DSP-BASED CENTRAL PROCESSING UNIT FOR POWER QUALITY MONITORING*

AUTHOR:

OSCAR LEONARDO QUINTERO YANES**

KEYWORDS: **Power quality, monitoring, real time, harmonics, discrete Fourier transform (*DFT*), fast Fourier transform (*FFT*), root mean square value, digital signal processor (*DSP*), TMS320LF2407A, Hanning, Hamming, Blackman.**

DESCRIPTION:

This document explains how algorithms were implemented in real time for monitoring synthetic or analogical signals whit harmonics. This monitoring is carried out by the evaluation module TMS320LF2407A of Texas Instruments, and it consists of three stages: digitization; processing and visualization of the results.

Initially the hardware used for the prototype is described. Therefore, the DSP TMS320LF2407A, their evaluation module, memory and peripherals of the DSP used in the implementation are described in general form.

In Chapter 2 the DSP software, the Code Composer Studio, and the tools of the language assembly are described.

In Chapter 3 the emulation in real time is described and explained. This is a debugging tool of Code Composer Studio, which allows; for example, access the internal and external DSP memories while the program is executed.

In Chapter 4 power quality monitoring process is shortly explained. This explanation is focused toward the monitoring of stationary state events, harmonics specifically. Classification and requirements of monitoring equipments used for power quality are also explained.

In Chapter 5 discrete Fourier transform (DFT), its operation principle and the method used for calculate it, are analyzed. The used method was the fast Fourier transform (FFT).

Chapter 6 explains how the implementation was carried out. Therefore, it begins with a description of the implemented prototype and the flow diagram of the algorithm is given. Then, the proposed requirements are specified. Lastly, the interface used for visualization and the results are described and presented.

---

* Degree work

** Physics and Mechanics Engineering College, Electronics Engineering, César Antonio Duarte Gualdrón

TABLA DE CONTENIDO

INTRODUCCIÓN 1

1 HARDWARE DEL PROTOTIPO 7

1.1 PROCESADOR DE SEÑALES DIGITALES TMS320LF2407A 9

1.1.1 MANEJADOR DE EVENTOS (EV) 11

1.1.2 CONVERSOR ANÁLOGO-DIGITAL (A/D) 16

1.1.3 INTERFAZ DE COMUNICACIÓN SERIAL (SCI) 20

1.1.4 TEMPORIZADOR WATCHDOG (WD) 23

1.2 MÓDULO DE EVALUACIÓN EVM TMS320LF2407 23

1.2.1 MEMORIA DE PROGRAMA 24

1.2.2 MEMORIA DE DATOS 26

2 HERRAMIENTAS DE PROGRAMACIÓN 28

2.1 HERRAMIENTAS DEL LENGUAJE ASSEMBLY 30

2.1.1 EL ENSAMBLADOR 32

2.1.2 EL ENLAZADOR 34

2.1.3 EL COMPILADOR DE C 39

2.2 CODE COMPOSER STUDIO 43

2.2.1 EDITOR 43

2.2.2 ENTORNO DEL PROYECTO 44

2.2.3 VENTANA DE OBSERVACIÓN 45

2.2.4 PUNTOS DE PARADA (BREAKPOINTS) 46

2.2.5 PUNTOS DE PRUEBA (PROBE POINTS) 47

2.2.6 MONITORIZACIÓN DE LA EJECUCIÓN DEL CÓDIGO EN EL DSP 49

2.2.7 LENGUAJE DE EXTENSIÓN GENERAL (GEL) 50

2.2.8 CONTROL AVANZADO DE PROGRAMACIÓN 51

2.2.9 MANEJADOR DE DEPURACIÓN PARALELA 52

2.2.10 MAPA DE MEMORIA 52

2.2.11 EJECUCIÓN POR PASOS 55

2.2.12 VENTANA DIS-ASSEMBLY 55

2.2.13 VENTANAS DE VISUALIZACIÓN DE VARIABLES 56

3 EMULACIÓN EN TIEMPO REAL (*REAL-TIME*) 60

3.1 EMULADOR XDS510PP PLUS 60

3.2 MODOS DE CONTROL DE LA EJECUCIÓN 62
3.2.1 MODO STOPMODE 62
3.2.2 MODO REAL-TIME 63
3.2.3 CONFIGURACIÓN DEL MONITOR REAL-TIME 65

4 MONITORIZACIÓN DE LA CALIDAD DE LA ENERGÍA ELÉCTRICA 66

4.1 MONITORIZACION DE EVENTOS EN ESTADO ESTACIONARIO 67
4.2 EQUIPOS DE MONITORIZACIÓN 69
4.2.1 EQUIPOS UTILIZADOS PARA EL ANALISÍS DE TENSIONES Y CORRIENTES NO SINUSOIDALES 70
4.2.2 CLASIFICACIÓN GENERAL DE LOS EQUIPOS DE MEDICIÓN DE ARMÓNICOS 71
4.2.3 REQUISITOS DE LOS EQUIPOS DE MEDIDA 73

5 LA TRANSFORMADA DISCRETA DE FOURIER (*DFT*) 78

5.1 ¿CÓMO FUNCIONA LA DFT? 78
5.2 LA TRANSFORMADA RÁPIDA DE FOURIER (*FFT*) 81
5.2.1 ¿CÓMO FUNCIONA LA FFT? 81
5.2.2 ALGORITMO FFT IMPLEMENTADO 83
5.2.3 CONSIDERACIONES PARA LA IMPLEMENTACIÓN DEL ALGORITMO .. 85
5.2.4 ERRORES EN LA IMPLEMENTACIÓN 94

6 IMPLEMENTACIÓN DEL PROTOTIPO DE UNIDAD DE PROCESAMIENTO 97

6.1 DESCRIPCIÓN DEL PROTOTIPO IMPLEMENTADO 97
6.2 REQUERIMIENTOS DEL PROTOTIPO 103
6.3 LIBRERÍAS FFT UTILIZADAS 104
6.3.1 MODIFICACIONES REALIZADAS A LAS LIBRERÍAS 104
6.4 INTERFAZ DE VISUALIZACIÓN 112
6.5 RESULTADOS DEL PROTOTIPO 114

7 CONLUSIONES Y TRABAJOS FUTUROS 119

7.1 CONCLUSIONES 119
7.2 TRABAJOS FUTUROS 122

REFERENCIAS BIBLIOGRÁFICAS 125

BIBLIOGRAFÍA 127

ANEXOS.....130

LISTA DE FIGURAS

Figura 1. Principales etapas de un equipo de monitorización 4
Figura 2. Diagrama de bloques del EVM TMS320LF2407 8
Figura 3. DSP LF2407A 9
Figura 4. Modo de funcionamiento del temporizador 2 15
Figura 5. Funcionamiento del conversor análogo-digital (*ADC*) 18
Figura 6. Pines del conector serie. 22
Figura 7. Memoria de programa 25
Figura 8. Memoria de datos 27
Figura 9. Flujo del desarrollo de un programa 30
Figura 10. Opciones de un proyecto. 32
Figura 11. Opciones del Ensamblador. 33
Figura 12. Archivo de comando de enlace 36
Figura 13. Opciones del enlazador 37
Figura 14. Opciones del compilador. 41
Figura 15. Ventana de visualización del proyecto. 45
Figura 16. Ventana de observación 46
Figura 17. Breakpoints 47
Figura 18. Entrada y salida de archivos. 48
Figura 19. Puntos de prueba. 49
Figura 20. Puntos de monitorización. 50
Figura 21. Mapa de memoria 53
Figura 22. Función *StartUp* del archivo *init.gel.* 54
Figura 23. Definiciones del mapa de memoria en el archivo *init.gel.* 54
Figura 24. Ventana *Dis-Assembly.* 56
Figura 25. Ventana de memoria. 57
Figura 26. Registros de la *CPU* y de estados. 57
Figura 27. Gráfica de tiempo y sus propiedades. 59
Figura 28. Sistema de evaluación. 61

Figura 29. Forma de onda: fundamental sinusoidal, componente armónica y distorsión [NTC 5000, 02]..... 67
Figura 30. Transformada discreta de Fourier compleja..... 79
Figura 31. Síntesis realizada por la FFT..... 82
Figura 32. Diagrama de flujo de la síntesis..... 83
Figura 33. Grafo del flujo del cálculo en mariposa. 83
Figura 34. Diezmado de una señal de 8 muestras. 84
Figura 35. Ordenamiento de inversión de bits. 84
Figura 36. FFT de 8 puntos. 85
Figura 37. Mariposa con entradas y salidas etiquetadas..... 89
Figura 38. Diagrama de flujo del algoritmo FFT implementado 94
Figura 39. Etapas del prototipo implementado. 97
Figura 40. Algoritmo del prototipo de monitor de calidad. 99
Figura 41. Diagrama de flujo de la interrupción 1..... 102
Figura 42. Diagrama de flujo de la interrupción 3..... 103
Figura 43. Algoritmo de la interfaz de visualización. 112
Figura 44. Interfaz de visualización. 113
Figura 45. Amplitud calculada por el monitor vs. error con respecto a los resultados de *Matlab*..... 116
Figura 46. Amplitud calculada por el monitor vs. error con respecto a los resultados de *Matlab* (escala logaritmica)..... 117
Figura 47. Diagrama de bloques de la expansión de interrupciones periféricas. 130
Figura 48. Plantilla para el archivo enlazador. 133
Figura 49. Proyecto implementado para el prototipo..... 136

LISTA DE TABLAS

Tabla 1. Conexiones para el cable de la interfaz serial. 22
Tabla 2. Requerimientos de exactitud para la medición de la amplitud de las componentes de frecuencia [IEC 61000-4-7, 91]..... 73
Tabla 3. Requisitos mínimos de atenuación (dB). 74
Tabla 4. Requisitos básicos para equipos que utilizan la FFT según [IEC 61000-4-7, 91]..... 76
Tabla 5. Resultados de la señal sintetizada con una sola componente de frecuencia. 114
Tabla 6. Resultados de la señal sintetizada compuesta de tres sinusoidales..... 115
Tabla 7. Resultados de la señal sintetizada compuesta de tres sinusoidales..... 116
Tabla 8. Resultados de la señal análoga..... 118
Tabla 9. Registros del LF2407A utilizados en la implementación..... 131
Tabla 10. Sincronía con la frecuencia de 60 Hz..... 137

LISTA DE ANEXOS

ANEXO A. DIAGRAMA DE BLOQUES DE LA EXPANSIÓN DE INTERRUPCIONES PERIFÉRICAS 130

ANEXO B. REGISTROS UTILIZADOS EN LA IMPLEMENTACIÓN ... 131

ANEXO C. DESARROLLO DE UN PROYECTO UTILIZANDO EL CODE COMPOSER STUDIO 133

ANEXO D. SINCRONÍA CON LA FRECUENCIA DE 60 HZ 137

INTRODUCCIÓN

A continuación se hará una descripción del punto principal de este proyecto y de la metodología utilizada para llevarlo a cabo y cumplir con los objetivos planteados. El punto central a resolver es implementar un prototipo de procesamiento para un monitor de calidad de energía eléctrica con los siguientes requisitos:

1. El prototipo debe permitir configurar la operación del monitor de calidad de acuerdo con las necesidades del análisis de la señal establecidas por el usuario. Por lo tanto, este prototipo en cualquier momento debe permitir cambiar la configuración, es decir, las propiedades del monitor sin reiniciar el procesador (DSP).
2. El análisis de la señal que se está capturando debe realizarse en tiempo real. Esto se debe a que los resultados que se obtienen en esta clase de análisis son más útiles ya que la información que está proporcionando es actual. Se pueden observar los cambios de la señal en el mismo momento que ocurren y no después que hayan pasado las transiciones.
3. El prototipo debe estar en capacidad de comunicar los cálculos obtenidos de la señal para poder visualizarlos y así observar el comportamiento de la señal. Esta característica es muy importante ya que es necesario comprobar que el monitor está funcionando correctamente, y esto se logra visualizando los datos obtenidos de los cálculos realizados sobre las muestras de la señal.

Considerando estos requerimientos se determinó la metodología para lograr implementar este prototipo y cumplir con los requerimientos mencionados para el monitor de calidad de energía eléctrica.

Como se ha visto ya se ha descrito una de las dos grandes inquietudes durante la elaboración del proyecto, la cual es: ¿Qué es lo que se quiere realizar?; ahora es necesario describir y explicar el ¿Cómo se va a lograr?. Para responder a esta inquietud aparece un grupo de interrogantes que proporcionan la orientación en la implementación del prototipo. Dentro de este grupo de inquietudes se tienen las siguientes:

1. ¿Cuál va a ser el hardware a utilizar para la elaboración del prototipo del procesador?.
2. ¿Qué herramientas de programación se van a utilizar para desarrollar este prototipo?.
3. ¿Cuál lenguaje de programación se usará para elaborar el algoritmo de captura, análisis de la señal, y de comunicación de los datos obtenidos?.
4. ¿Cuál es el *software* para hacer la plataforma de la interfaz de visualización de los datos obtenidos?.

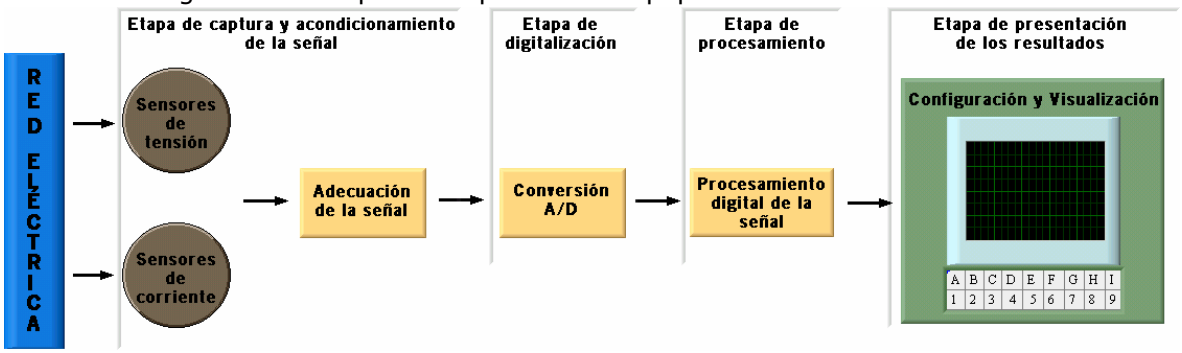
Estos cuatro interrogantes son los grandes pilares en los que se basa la elaboración del prototipo y serán resueltos de manera más precisa y detallada durante este libro en los siguientes capítulos. Es posible que muchos de los aspectos sobre el hardware utilizado y de las herramientas de programación no sean aclarados ya que no fueron utilizados en la elaboración del prototipo; no obstante, se indicarán referencias para encontrar toda la información necesaria para la elaboración de nuevos proyectos.

La implementación de un prototipo de unidad de procesamiento para un monitor de la calidad de energía eléctrica surge de la necesidad de caracterizar las perturbaciones electromagnéticas en una ubicación particular, y puede tener varios enfoques como: diagnóstico de incompatibilidades entre el sistema eléctrico y las cargas, evaluación del ambiente eléctrico en un punto particular o predicción del comportamiento de un equipo o dispositivo para determinar la necesidad de mejorar la calidad de la energía eléctrica en ese punto.

La monitorización de la calidad de la energía eléctrica requiere de equipos con características especiales de procesamiento de acuerdo con los objetivos de la monitorización. Actualmente se emplean desde equipos sencillos como los vóltmetros hasta equipos complejos y costosos como los analizadores de espectro.

Para realizar esta monitorización de la energía eléctrica se implemento un prototipo de unidad de procesamiento digital (sin incluir la etapa de captura y acondicionamiento de la señal, ver Figura 1), utilizando un procesador de señales digitales (*DSP*), ya que en este momento se presentan como una buena opción de procesadores para los sistemas electrónicos de monitorización y control en tiempo real. Aprovechando su especial arquitectura y su gran desempeño se pueden implementar una amplia variedad de algoritmos complejos de procesamiento, control y medición necesarios para monitorizar la calidad de servicio de energía eléctrica. De esta forma se podrán desarrollar equipos confiables y precisos contribuyendo a la solución de los problemas asociados a la calidad del servicio que existen en el sistema eléctrico de Colombia.

Figura 1. Principales etapas de un equipo de monitorización



Esta implementación del prototipo se realizó con el módulo de evaluación del procesador digital de señales TMS320LF2407A fabricado por *Texas Instruments*. Por medio de este módulo de evaluación, el cual permite acceder a todas las capacidades y pines del *DSP*, y del *Code Composer Studio*, que es el *software* elaborado por *Texas Instruments* para poder programar el *DSP*, se implementaron los algoritmos necesarios para la adquisición y procesamiento de la señal y transmisión de resultados. Por lo tanto se estudio la arquitectura, capacidades de procesamiento y programación del procesador de señales *(DSP)* de coma fija *TMS320LF2407A* y las limitaciones del modulo de evaluación para poder realizar esta implementación. Este algoritmo realiza: adquisición y digitalización de una cantidad de muestras de una señal analógica por medio del conversor análogo-digital (*A/D*) del *DSP*, almacena estas muestras, las procesa según la configuración para la monitorización y luego las transmite a un computador de visualización por medio de una interfaz serial. Para la visualización de los resultados y configuración del prototipo se desarrollo una interfaz gráfica en *LabView*.

Los usuarios directos del prototipo serán los miembros del grupo GISEL y la comunidad académica de la Universidad Industrial de Santander. De manera indirecta, lo serán también los empresarios interesados en el desarrollo de equipos basados en *DSPs,* las empresas distribuidoras de la

energía eléctrica y los entes reguladores, de normalización y de vigilancia del sector eléctrico colombiano.

La organización de este documento es descrita a continuación. En el primer capítulo se describe el *hardware* utilizado para la implementación del prototipo. Por lo tanto, se describe en forma general el *DSP* TMS320LF2407A y su módulo de evaluación, se describen la memoria y los periféricos del *DSP* utilizados en la implementación.

En el capitulo 2 se describe el software de programación del *DSP*, el *Code Composer Studio* y las herramientas del lenguaje *assembly*. La descripción de las herramientas del lenguaje *assembly* se centra en: el ensamblador, el enlazador, el compilador de C. También se van explicando las opciones que fueron utilizadas para cada una de estas en la implementación del prototipo. Para el *Code Composer Studio* la descripción se va realizando para cada una de las herramientas de depuración y evaluación que este posee.

El capítulo 3 describe y explica lo referente a la emulación[1] en tiempo real. Esta es una característica que posee el modulo de evaluación y que puede ser usada a través de su emulador. Esta es una herramienta de depuración del *Code Composer Studio*, la cual permite acceder a las memorias internas y externas del *DSP* mientras se ejecuta el programa entre otras cosas.

En el capítulo 4 se explica brevemente lo referente a la monitorización de la calidad de energía eléctrica, enfocado hacia la monitorización de

[1] Imitación que un dispositivo hace del funcionamiento de otro. Esta característica permite observar el funcionamiento del DSP y acceder a su memoria y registros mientras se ejecuta el programa.

eventos en estado estacionario en donde se encuentran los armónicos. También se presenta lo referente a los equipos de monitorización de calidad, su clasificación y los requerimientos. Esta explicación se basa en las normas y estándares internacionales.

En el capítulo 5 se analiza la transformada discreta de Fourier (*DFT*), su funcionamiento y método usado para calcularlo. El método usado fue la transformada rápida de Fourier. Además se explica el análisis realizado para desarrollar el algoritmo, las consideraciones usadas para su programación y los errores en su implementación.

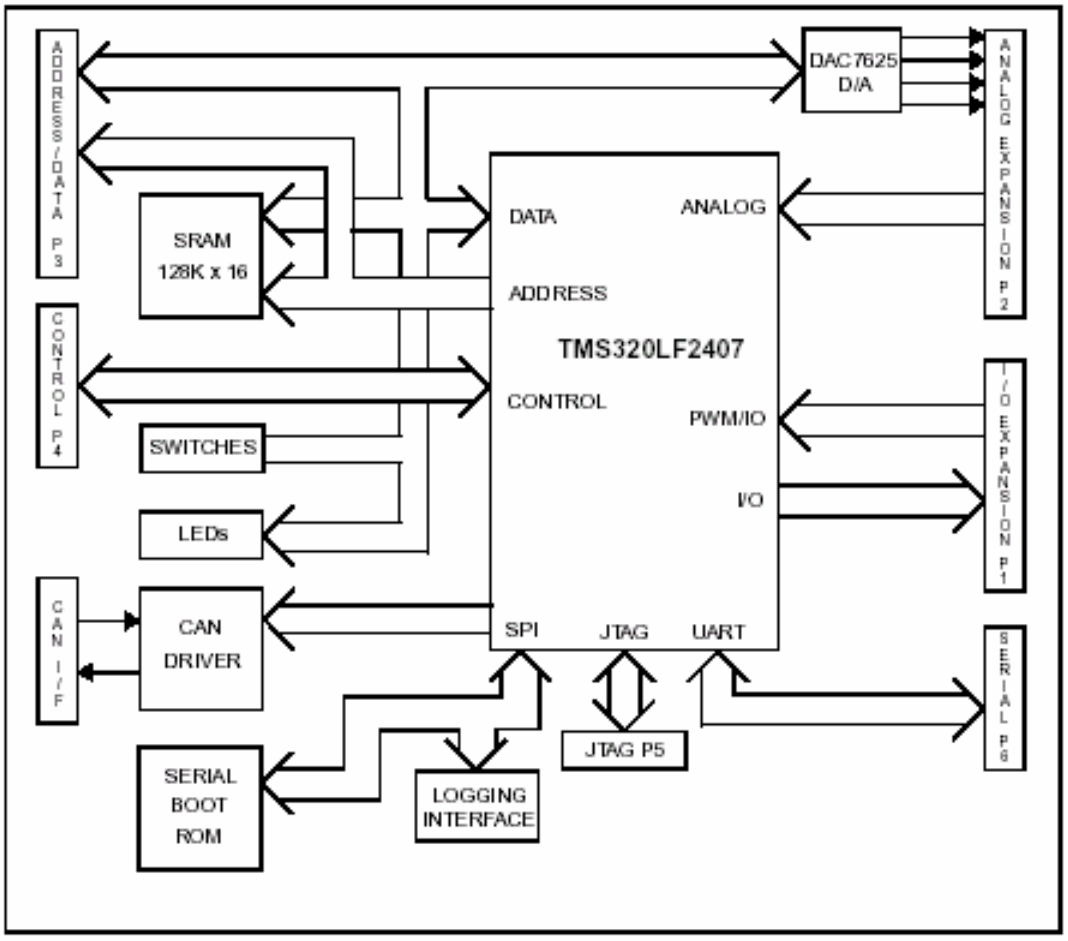
El Capítulo 6 explica cómo se realizó la implementación. Por lo tanto, se comienza por una descripción del prototipo implementado y se da el diagrama de flujo del algoritmo. Luego se van explicando una a una las modificaciones de la librería usada como plantilla para poder cumplir con los requerimientos ya propuestos. Y por último se describe la interfaz usada para la visualización y se presentan los resultados obtenidos.

1 HARDWARE DEL PROTOTIPO

En este primer capítulo se hará una descripción del hardware utilizado para la implementación del prototipo. En primera instancia la base tecnológica seleccionada para este prototipo de procesador de monitorización es el procesador digital de señales (*DSP*) TMS320LF2407A de *Texas Instruments* y su módulo de evaluación TMS320LF2407 EVM, el cual pertenece a la escuela de ingenierías eléctrica, electrónica y de telecomunicaciones. De aquí en adelante para hacer referencia a este *DSP* se utilizará la contracción LF2407A.

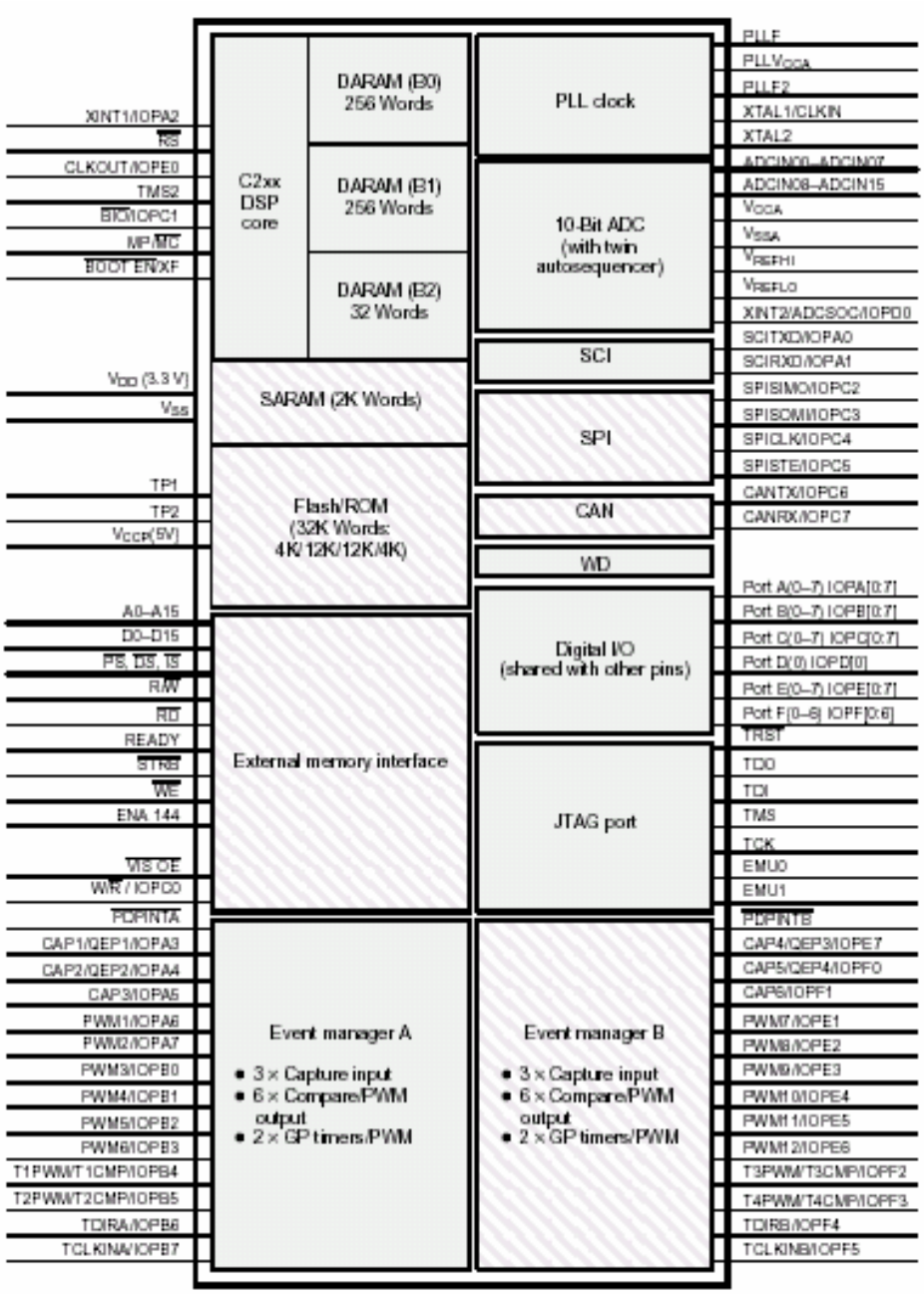
El LF2407A es un *DSP* basado en la *CPU* C2xLP de 16-bit, de coma fija y de bajo consumo y es complementado con un gran rango de periféricos dentro del chip, una memoria de programa flash y una memoria *RAM* de acceso dual. Los características principales de este *DSP* y su modulo de evaluación son: Posee dos manejadores de eventos (*EVA* y *EVB*), un controlador de área local (*CAN*), dos puertos de comunicación serial (*SPI* y *SCI*), un conversor análogo-digital (*A/D*) de 10 *bit*, un temporizador *watchdog*, pines de entrada y salida digitales bidireccionales de propósito general (*GPIO*), un reloj para el funcionamiento de la CPU hasta 40 MHz, un puerto *JTAG*, una interfaz para la memoria externa, un conversor digital-análogo de 4 canales, 4 conectores de expansión (para señales digitales de entrada y salida, señales análogas, señales de datos y dirección y señales de control), 16 *jumpers* para seleccionar las características de funcionamiento del LF2407A y *LEDS* de prueba. Estas características pueden verse en las Figuras 2 y 3.

Figura 2. Diagrama de bloques del EVM TMS320LF2407



Todas estas características antes mencionadas le permiten al LF2407A cumplir con una gran variedad de aplicaciones de control y de procesamiento de señales sin perder precisión ni rendimiento. Este diseño le permite al LF2407A ser la alternativa más completa para cumplir a cabalidad con los requerimientos en aplicaciones de control y medición en sistemas de energía eléctrica.

Figura 3. DSP LF2407A



1.1 PROCESADOR DE SEÑALES DIGITALES TMS320LF2407A

Este procesador pertenece a la familia de procesadores TMS320. Esta familia de procesadores consta de multiprocesadores de coma fija y de

coma flotante, y de controladores de coma fija. Los *DSP*s TMS320 tienen una arquitectura diseñada especialmente para el procesamiento se señales en tiempo real. La serie LF240xA de controladores *DSP* combina esta capacidad de procesamiento de señales en tiempo real con un controlador de periféricos para crear una solución ideal en aplicaciones de sistemas de control [T.I spru357b, 01].

Por la integración del elevado desempeño de un *DSP* y los periféricos en el chip de un microcontrolador dentro de una solución de un solo chip, la serie LF240xA es una alternativa económica a las unidades de microcontroladores tradicionales (*MCUs*) y diseños de multichip costosos [T.I spru357b,01]. Otras ventajas de esta serie son: la gran velocidad de la unidad de procesamiento central (*CPU*) (40 millones de instrucciones por segundo, *MIPS*) que permite programar algoritmos en tiempo real en lugar de aproximar resultados con tablas de búsqueda; la arquitectura basada en palabras de 16-bit con registros de 32-bit para almacenamiento de resultados intermedios, y además los dos desplazadores en hardware para escalar números, minimizando los errores de cuantificación y de truncamiento [T.I spru357b, 01].

Como ya se ha mencionado antes, una de las principales características de la serie LF240xA es su controlador de periféricos, esto es muy favorable para la implementación del prototipo ya que se cuenta con la posibilidad de utilizar tales periféricos acorde con los requerimientos del prototipo. De los periféricos que posee el *DSP* TMS320LF2407A se usaron los siguientes:

- El controlador de eventos (*EV*)
- El conversor análogo-digital (*ADC*)
- La interfase de comunicación serial (*SCI*)
- El temporizador *watchdog*

A continuación se describen los periféricos utilizados y los registros del DSP para controlar su funcionamiento.

1.1.1 MANEJADOR DE EVENTOS (EV)

El módulo manejador de eventos (*EV*) provee un amplio rango de funciones y características que son particularmente útiles en el control de movimiento y aplicaciones de control de motores. El 2407A tiene dos manejadores de eventos, EVA y EVB, los cuales tienen funcionalidad y registros exactamente idénticos [T.I spru357b, 01].

Cada módulo de *EV* contiene los siguientes bloques funcionales:

- Dos temporizadores de propósito general (*GP*)
- Tres unidades de comparación
- Circuitos de modulación de ancho de pulso (*PWM*)
- Tres unidades de captura para registrar flancos de subida o de bajada en una entrada.
- Circuito codificador de pulsos en cuadratura (*QEP*) utilizados para determinar posición, velocidad y dirección de giro de un motor.
- Lógica de interrupción para atender cada una de las interrupciones de los anteriores bloques funcionales.

De estos bloques se describirán los temporizadores ya que para el prototipo se utilizó el temporizador 2 (*Timer* 2) del manejador de eventos A (*EVA*), asimismo se describirá la lógica de interrupción para explicar como se utiliza este temporizador como señal de disparo para comenzar con la adquisición de la señal. Mayor información sobre el EV se puede encontrar en [T.I spru357b, 01].

Las interrupciones del *EV* están organizadas en tres grupos. Cada grupo es asignado a una interrupción de *CPU* (INT2,3 ó 4). Como cada grupo tiene fuentes de interrupciones múltiples, las demandas de interrupción

de *CPU* son procesadas por el controlador de expansión de interrupciones periféricas (*PIE*) [T.I spru357b, 01].

En este momento puede surgir la pregunta ¿Qué son las interrupciones de *CPU*?. Lo que ocurre es que la *CPU* del LF2407A soporta una interrupción no enmascarable (*NMI*) y seis demandas de interrupciones con prioridad enmascarables (INT1-INT6). Estas seis demandas tienen dos registros: el registro de enmascaramiento de interrupciones (*IMR*) que se utiliza para activar o inhabilitar la interrupción deseada y el registro bandera de interrupción (*IFR*) que es usado para identificar las interrupciones pendientes. Como los dispositivos LF2407A tienen muchos periféricos, donde cada periférico es capaz de generar una o más interrupciones en respuesta a muchos eventos en el nivel de periféricos, y la CPU del 240xA no tiene la suficiente capacidad para manejar todas las demandas de interrupciones periféricas a nivel de chip, un controlador de interrupciones centralizado (*PIE*) es requerido para arbitrar las demandas de interrupciones de varias fuentes tales como periféricos y otros pines externos. En el anexo A se puede ver en la Figura 47 el diagrama de bloques de cómo funciona este controlador. Para tener una información más especifica de cómo funcionan las interrupciones en el DSP 2407A, de su reconocimiento, de sus registros de control y direcciones en la memoria es necesario consultar el capítulo 2 de la referencia [T.I spru357b, 01].

Cuando el manejador de eventos produce una demanda de interrupción se deben tener en cuenta dos aspectos. El primero es que se debe activar, es decir, desenmascarar la interrupción que se va a utilizar y para lograr se deben utilizar los registros de enmascaramiento de interrupciones EVxIMRA, EVxIMRB y EVxIMRC (x = A o B) estableciendo un uno (1) para habilitar la interrupción o un cero (0) para inhabilitarla/enmascararla. El segundo aspecto a tener en cuenta es que

si una condición de interrupción periférica ocurre, el respectivo bit de bandera de los registros EVxIFRA, EVxIFRB y EVxIFRC (x = A o B) es puesto en 1. Una vez establecidas estas banderas es necesario borrarlas por medio de *software*. Es obligatorio limpiar estas banderas o las futuras interrupciones no serán reconocidas. Los registros de control del LF2407A utilizados para la implementación del prototipo y sus direcciones de memoria se encuentran la Tabla 9 en el anexo B.

Existen 16 banderas de interrupciones en los registros EVAIFRA, EVAIFRB, EVBIFRA y EVBIFRB para los temporizadores *GP, c*ada uno de los cuatro temporizadores de propósito general pueden generar cuatro interrupciones debido a los siguientes eventos:

- Desbordamiento por encima TxOFINT(x =1, 2, 3 ó4): Es cuando el contador alcanza FFFFh.
- Desbordamiento por debajo TxUFINT(x =1, 2, 3 ó 4): Es cuando el contador alcanza 0000h.
- Coincidencia de comparación TxCINT(x =1, 2, 3 ó 4): Es cuando el contenido del registro del contador coincide con el del registro de comparación.
- Coincidencia de periodo TxPINT(x =1, 2, 3 ó 4): Es cuando el contenido del registro del contador coincide con el del registro de periodo de conteo.

En la implementación del prototipo se utilizó la interrupción por desbordamiento por debajo (*underflow*) del temporizador 2 (T2UFINT). Esta interrupción y la operación que ella conlleva serán explicadas a continuación al explicar el modo de operación del temporizador.

Existen dos temporizadores *GP* en cada manejador de eventos. Para el prototipo se utilizó el temporizador 2 (*Timer* 2) del manejador de eventos A (*EVA*) como base de tiempo para la generación del periodo de muestreo

de la señal a monitorizar y analizar, por lo tanto, en esta sección se tratara solamente este temporizador ya que el funcionamiento de los otros temporizadores es idéntico.

En primer lugar es necesario conocer los bloques funcionales del temporizador, específicamente los utilizados en la implementación del prototipo. Los cuales son:

- Un registro contador ascendente/descendente de 16-bit con acceso de lectura y escritura TxCNT (x = 1, 2, 3 ó 4). Este registro almacena el valor actual del contador y se va incrementando o disminuyendo dependiendo de la dirección de conteo.
- Un registro de periodo con acceso de lectura y escritura de 16-bit TxPR (x = 1, 2, 3 ó 4).
- Un registro individual de control del temporizador con acceso de lectura y escritura de 16-bit TxCON (x = 1, 2, 3 ó 4).

Para empezar, se estudiara el registro de control del temporizador ya que éste es el encargado de controlar su modo de operación. Los bits en el registro T2CON determinan lo siguiente:

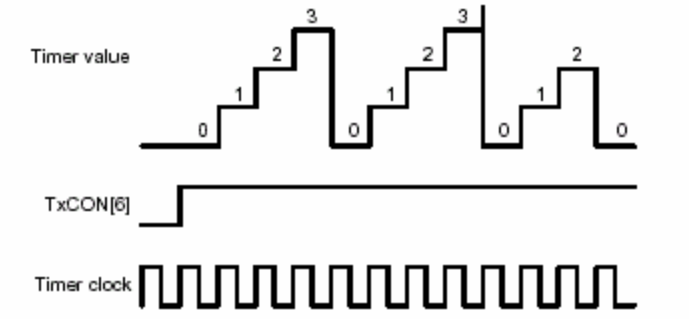
1. Cuál de los 4 modos de conteo se le establece al temporizador
2. Seleccionar el uso de un reloj interno o externo para el temporizador.
3. Cuál de las 8 entradas de los factores de pre-escalamiento de reloj es utilizada.
4. En cuál condición el registro de comparación del temporizador es cargado de nuevo.
5. Cuándo un temporizador está habilitado o inhabilitado.
6. Cuándo la operación de comparación del comparador está habilitada o inhabilitada.
7. Cuál registro de periodo es usado para el temporizador 2, el propio o el del temporizador 1.

Ahora se describirán los 4 modos de operación del temporizador ya que de estos depende la interrupción a utilizar del manejador de eventos. Los 4 modos de operación que se tienen para el temporizador GP son:

- Modo *STOP/HOLD.*
- Modo de conteo creciente continúo.
- Modo de conteo creciente/decreciente direccional.
- Modo de conteo creciente/decreciente continúo.

La opción utilizada para la elaboración del prototipo fue la de modo de conteo creciente continuo como se muestra en a Figura 4. En este modo el temporizador GP cuenta de manera ascendente de acuerdo al reloj de entrada escalado hasta que el valor del registro del contador del temporizador coincida con el valor del registro de periodo del mismo. En el flanco de subida del reloj de entrada después de la coincidencia, el temporizador GP va a cero y comienza a contar ascendente de nuevo.

Figura 4. Modo de funcionamiento del temporizador 2.



Ya que se conoce el modo de operación escogido para el funcionamiento del temporizador GP se puede entender la razón por la cual la interrupción del manejador de eventos A (EVA) que se seleccionó fue la interrupción de desbordamiento por debajo (*underflow*), ya que cuando el temporizador alcance el valor que se establece en el registro de periodo el se reinicia y va a cero (0000h) lo cual produce la demanda de la interrupción y ésta debe ser atendida. Como se ha dicho, en la

implementación del prototipo se utilizó la característica de esta interrupción para generar el periodo de muestreo para la captura de las muestras de la señal con el conversor análogo-digital.

En este se ha presentado lo que se necesita conocer sobre el funcionamiento del temporizador GP y sobre las interrupciones a manejar con este módulo. Las demás opciones sobre el registro T2CON que fueron seleccionadas son: Para el proyecto se determinó que la operación del temporizador no fuese afectada con la suspensión de la emulación, la fuente de reloj para el temporizador es el reloj de la CPU y no tiene ningún factor de pre-escalamiento, es decir, es el mismo que el de la CPU (40 Mhz), por obvias razones el temporizador está habilitado, la función de comparación está inhabilitada y el registro de periodo del temporizador 2 es el de él y no el del temporizador 1.

1.1.2 CONVERSOR ANÁLOGO-DIGITAL (A/D)

Este es otro aspecto muy importante para la implementación del prototipo ya que todos los cálculos a desarrollar y los resultados a visualizar en el monitor de calidad se basan en los datos obtenidos por este módulo o periférico del *DSP*.

En el algoritmo desarrollado para el prototipo no es muy claro el manejo de este periférico ya que solo aparecen unas cuantas sentencias de iniciación en el archivo principal o fuente (archivo .c) y algunas otras de declaración en los archivos de encabezado (archivos adc.h, cdemo.h y F2407ADC.h) utilizados para este módulo. Esto se debe a que todo el código que lleva a cabo la puesta en marcha del conversor está en la librería stb.lib que hace parte del algoritmo que se desarrollo para el prototipo. A continuación se da una descripción de cómo esta librería maneja y configura el conversor.

Para comenzar se describirán las características que hacen parte del conversor A/D:

- Conversor *Sample and Hold* (*S/H*) de 10 bit.
- Tiempo de conversión de 375 ns (S/H + conversión).
- 16 entradas análogas multiplexadas, (ADCIN0-ADCIN15).
- Capacidad de auto-secuencia de hasta 16 autoconversiones en una secuencia simple. Cada sesión de conversión puede ser programada para seleccionar cualquiera de los 16 canales de entrada.
- Dos secuenciadores independientes de 8 estados (SEQ1 y SEQ2) que pueden ser operados individualmente en modo de secuenciador dual o en modo de cascada en un secuenciador de 16 estados (SEQ).
- Cuatro registros de control de secuencia (CHSELSEQ*n* *n* = 1, 2, 3 y 4) que determinan la secuencia de canales análogos que son tomados para conversión en un modo secuencial dado.
- 16 registros de resultados, direccionados individualmente, para almacenar los valores de las conversiones (RESULT0-RESULT15).

- Múltiples fuentes de disparo para la secuencia de comienzo de conversión (*SOC*).

Teniendo en cuenta estas características se explicará el funcionamiento del conversor. EL diagrama de flujo del funcionamiento del *ADC* se encuentra en la Figura 5. El conversor A/D tiene dos secuenciadores de 8 estados (o auto-conversiones) independientes (SEQ1 y SEQ2) que pueden también estar en cascada para formar un secuenciador de 16 estados. Para toda conversión puede ser seleccionado cualquiera de los 16 canales de entrada disponibles y el valor digital se almacena en el registro de resultados respectivo a cada canal (RESULT*n* *n* = 0 hasta 15).

Para este proyecto, la adquisición de las muestras de la señal a monitorizar se realiza por medio de unas funciones ya implementadas en las librerías utilizadas como plantillas base. Estas librerías configuran el ADC en modo de cascada y por lo tanto lo resta por saber es ¿cómo se determinan los canales de entrada para el conversor?. Para lograr esto, el DSP 2407A tiene los registros CHSELSEQ*n* (n = 1, 2, 3 y 4) en donde los campos de bit CONV*n* (*n* = 0 hasta 15) de estos registros definen tales entradas. CONV*n* es un campo de 4 bit que especifica uno de los 16 canales para la conversión. Como se puede lograr un máximo de 16 conversiones en una secuencia cuando se usa el secuenciador en cascada, los 16 campos de 4 bits (CONV00-CONV15) son disponibles y esparcidos a través de los 4 registros de 16 bit (CHSELSEQ1-CHSELSEQ4). Los bits CONV*n* pueden tener cualquier valor de 00 a 15, dependiendo del canal que se desee muestrear, en el orden que se requiera y además el mismo canal puede ser seleccionado varias veces.

Figura 5. Funcionamiento del conversor análogo-digital (*ADC*)

Para lograr todo lo anterior, al igual que el temporizador, el conversor A/D tiene sus propios registros de control de operación. Los registros encargados del control del conversor y los usados para el prototipo son el ADCTRL1, ADCTRL2, MAXCONV, RESULT0 y el CHSELSEQ1. Como se mencionó anteriormente esta adquisición y conversión es realizada por una función definida en una de las librerías base usadas en el proyecto, y por lo tanto, esta librería es la encargada de la configuración del funcionamiento del ADC. Estos registros fueron usados de la siguiente forma:

1. El registro ADCTRL1 se utilizó para configurar que; cuando la emulación se detenga el conversor complete la conversión que se esté llevando a cabo y luego si quede inactivo; que se aplique un factor de 2 de pre-escalado al reloj del conversor para aumentar el tiempo de adquisición y que el reloj del conversor sea el mismo de la *CPU* del *DSP* y que el conversor trabaje en cascada y en modo *start-stop* el cual es sincronizado para varios disparos de inicio de conversión (*SOC*) separados en tiempo.
2. Con el registro ADCTRL2 se configura el disparo de inicio de conversión (*SOC*) de la señal análoga presente en los canales de entrada seleccionados; se inhabilitan las interrupciones producidas por el conversor debidas a la finalización de la conversión (*EOC*) y se da la orden de reiniciar el secuenciador (SEQ1) para que SEQ CNTR*n* sea recargado con el valor original de MAXCONV.
3. Con el registro MAXCONV se controla el número de conversiones en una secuencia. Este registro puede tener valores entre 0 y 7, y es cargado en los bits de estado de contador de secuencia (SEQ CNTR0-3) en el registro de estados de auto-secuencia (AUTO_SEQ_SR) al comienzo de una sesión de conversión. Los bits de SEQ CNTR*n* disminuyen su valor hasta que éste alcance el valor de cero. El número de conversiones completas durante una secuencia es igual a MAX CONV*n* más uno. Para el prototipo se

utilizaron 4 auto-conversiones y por lo tanto el registro MAXCONV se cargó con el valor de 3.

4. Con el registro CHSELSEQ1 se seleccionaron los canales a ser utilizados como entradas para el conversor. Como en la implementación del prototipo solo se utilizó un canal, se dispuso el mismo canal para las cuatro auto-conversiones utilizadas.
5. Por último, está el registro RESULT0 en donde se almacena el resultado de la conversión realizada para el canal seleccionado en el campo de bit CONV00 en el secuenciador 1 (SEQ1). Como ya se mencionó, el único canal utilizado fue el primero (el cero) y por lo tanto solo se tiene en cuenta este registro de resultados aunque se hayan llevado a cabo 4 auto-conversiones.

1.1.3 INTERFAZ DE COMUNICACIÓN SERIAL (SCI)

En esta sección se describe la arquitectura, las funciones y la programación del módulo de interfase de comunicación serial (*SCI*). Las características principales de este módulo son:

1. El módulo *SCI* programable soporta comunicación digital serial asíncrona entre la *CPU* del *DSP* y otros periféricos asíncronos que usen el formato estándar sin retorno a cero (*NRZ, non-return-to-zero*).
2. Posee dos pines. El pin SCIRXD que recibe los datos de entrada y el pin SCITXD que transmite los datos de salida.
3. Tanto el receptor como el transmisor del módulo *SCI* poseen un *buffer* propio cada uno, poseen bits de habilitación e interrupción y pueden funcionar independientes o simultáneamente en el modo *full-duplex*.
4. Posee un registro de 16 bit para programar la velocidad de transmisión alrededor de 65 000 velocidades diferentes.

5. Posee una longitud de palabra de datos programable de uno a ocho bits, posee bits de parada programables (uno o dos), posee un reloj serial interno y banderas de detección de error.
6. Tanto el transmisor como el receptor pueden ser operados por interrupciones o por sondeo de las banderas de estado de cada uno.

Para la implementación del prototipo se empleo el formato de comunicación asíncrono de dos líneas o *full-duplex*. Los principales elementos que se usan en la operación *full-duplex* son:

- Un transmisor (TX) y sus registros más importantes:
    1. El registro del *buffer* de los datos del transmisor SCITXBUF. Contiene los datos cargados por la *CPU* a ser transmitidos.
    2. El registro de desplazamiento del transmisor TXSHF. Acepta los datos del registro SCITXBUF y desplaza los datos hacia el pin SCITXD un bit a la vez.
- Un receptor (RX) y sus registros más importantes:
    1. El registro de desplazamiento del receptor RXSHF. Desplaza los datos desde el pin SCIRXD un bit a la vez.
    2. El registro del *buffer* de los datos del receptor SCIRXBUF. Contiene los datos a ser leídos por la *CPU*. Los datos provenientes de un procesador remoto son cargados en el registro RXSHF y luego en los registros SCIRXBUF y SCIRXEMU.
- Un generador de baudios programable.
- Registros de control y estado del módulo, los cuales están mapeados en la memoria de datos.

Además de seleccionar que el módulo SCI trabajara en modo *full-duplex*, se configuraron las siguientes características para la transmisión de datos: un bit de inicio, ocho bits de datos, sin paridad, un bit de parada y

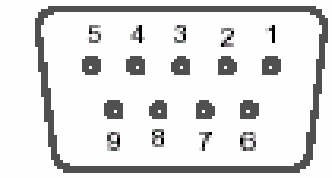
se habilitó la interrupción de alta prioridad debida al receptor del módulo *SCI* del *DSP*. Para esta configuración se utilizaron los registros SCICCR, SCICTL1, SCICTL2, SCIPRI, SCIHBAUD y SCILBAUD del módulo *SCI*. Con los registros SCIHBAUD y SCILBAUD se establece la velocidad a la que se transmiten y reciben los datos de la comunicación serial. La velocidad seleccionada para el prototipo fue de 115200 baudios por segundo.

Este modo de comunicación *full-duplex* se hizo a través de un receptor-transmisor asíncrono universal (*UART*) que usa formato RS-232-C, el cual está incluido en el módulo de evaluación usado para el prototipo. Las conexiones de los pines para el conector serie y la configuración del cable usado para la interfaz serial se muestran a continuación en la Tabla 1 y en la Figura 6.

Tabla 1. Conexiones para el cable de la interfaz serial.

Pin #	***PC***	*EVM*
2	Rx, entrada	Tx, salida
3	Tx, salida	Rx, entrada
4	DTR, salida	*Reset*/CTS, entrada
5	Tierra, GND	Tierra, GND
8	CTS, input	RTS, salida

Figura 6. Pines del conector serie.



1.1.4 TEMPORIZADOR WATCHDOG (WD)

La función principal del temporizador *watchdog* es monitorizar las operaciones de software y de hardware e implementar funciones de reinicio (*reset*) del sistema. Este temporizador es un contador de 8-bit, incremental y con reset, que protege contra fallas de software del sistema y ruptura de la CPU suministrando un reset del sistema cuando el registro WDKEY no es atendido antes de un desbordamiento del contador del *WD* (WDCNTR) o cuando una secuencia errónea es escrita al WDKEY.

El temporizador *WD* opera independientemente de la *CPU* y está habilitado siempre ya que no necesita ninguna inicialización de la *CPU* para funcionar. Esto significó que, para evitar en *reset* prematuro, en la implementación del prototipo la configuración del WD se debe hacer prontamente en la secuencia de inicio. Por esta razón una de las primeras instrucciones que se ejecutan en el programa cargado en el DSP es inhabilitar el temporizador WD, usando el bit 6 del registro de control WDCR del WD) y reiniciar el contador. Estas instrucciones mencionadas anteriormente se llevan a cabo con las funciones *wdog.disable* y *wdog.reset*, de la función de inicio *RstIni*t, y cuyo código fuente se encuentra en la librería *stb.lib*.

1.2 MÓDULO DE EVALUACIÓN EVM TMS320LF2407

En este punto ya se ha descrito la mayor parte de la información requerida sobre el *hardware* utilizado (*DSP* LF2407A). En esta sección se presenta una característica fundamental para la implementación del prototipo, la cual consiste en que el módulo de evaluación (*EVM*) es una tarjeta *stand-alone* que permite examinar varias características del *DSP* LF2407A y además es una excelente plataforma para el desarrollo y ejecución de *software* para esta familia de procesadores.

Entre las principales características de este módulo, se tiene que el módulo opera a 40 millones de instrucciones por segundo (MIPS), posee 128k palabras de memoria sin ningún estado de espera, tiene 32k palabras de memoria flash *ROM* en el chip, 4 conectores de expansión (para datos/direccionamiento, entradas/salidas digitales, canales análogos y control), un conector *JTAG IEEE* 1149.1 para emulación opcional y una entrada de alimentación de 5 voltios.

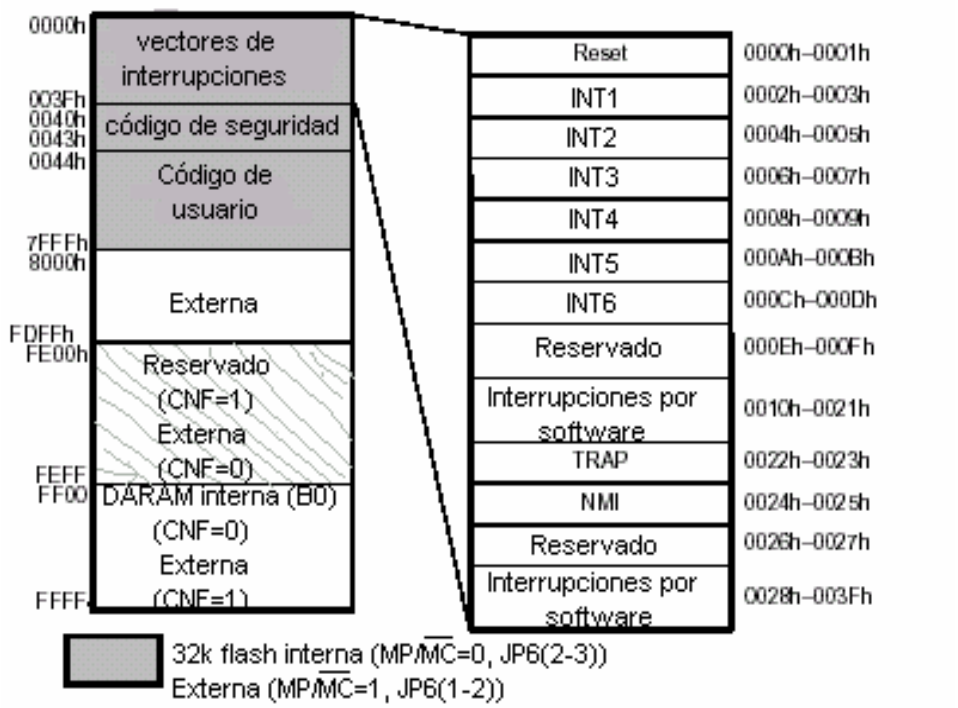
Finalmente la memoria del EVM consta de la memoria interna de el DSP y de una memoria externa, la cual se encuentra incluida en el módulo de evaluación y puede ser accesada sin estados de espera programando apropiadamente el registro de generación de estados de espera (WSGR).

1.2.1 MEMORIA DE PROGRAMA

Esta memoria es usada para almacenar el código de usuario o código fuente, operandos inmediatos o tablas de información. Para esta memoria se tiene un máximo de 64k palabras de 16-bit. Dentro de esta capacidad están incluidas las memorias *RAM* internas de acceso dual y de acceso simple del *DSP* LF2407A, la memoria flash *EEPROM* interna del *DSP* LF1407A, y la memoria externa del módulo *EVM*.

Hay dos configuraciones para la memoria de programa. La selección de esta configuración se realiza por medio del *jumper* JP6. Si JP6 está en la posición 2-3 el DSP está en modo de microcomputador y la memoria flash interna está habilitada. Si JP6 está en posición 1-2 el DSP está en modo de microprocesador, por lo tanto la memoria flash está inhabilitada y todo el rango de direcciones de programa está disponible para la memoria externa.

Figura 7. Memoria de programa



Observando la Figura 7, se colige que entre las direcciones 0000h-003Fh se cargan todas las interrupciones que el DSP LF2407A es capaz de manejar. Este espacio de memoria es reservado especialmente para esto y en él se encuentran las direcciones a las que el puntero del programa debe saltar para atender la demanda de la interrupción que este pendiente, es decir, donde se encuentra la rutina de servicio de interrupción (ISR).

Para realizar todas las pruebas e implementación del prototipo se configuro la memoria de programa en modo de microprocesador, es decir, usando la memoria externa y se seleccionaron 7 estados de espera para la lectura o escritura de la memoria de programa y de datos con el registro de estados de espera WSGR.

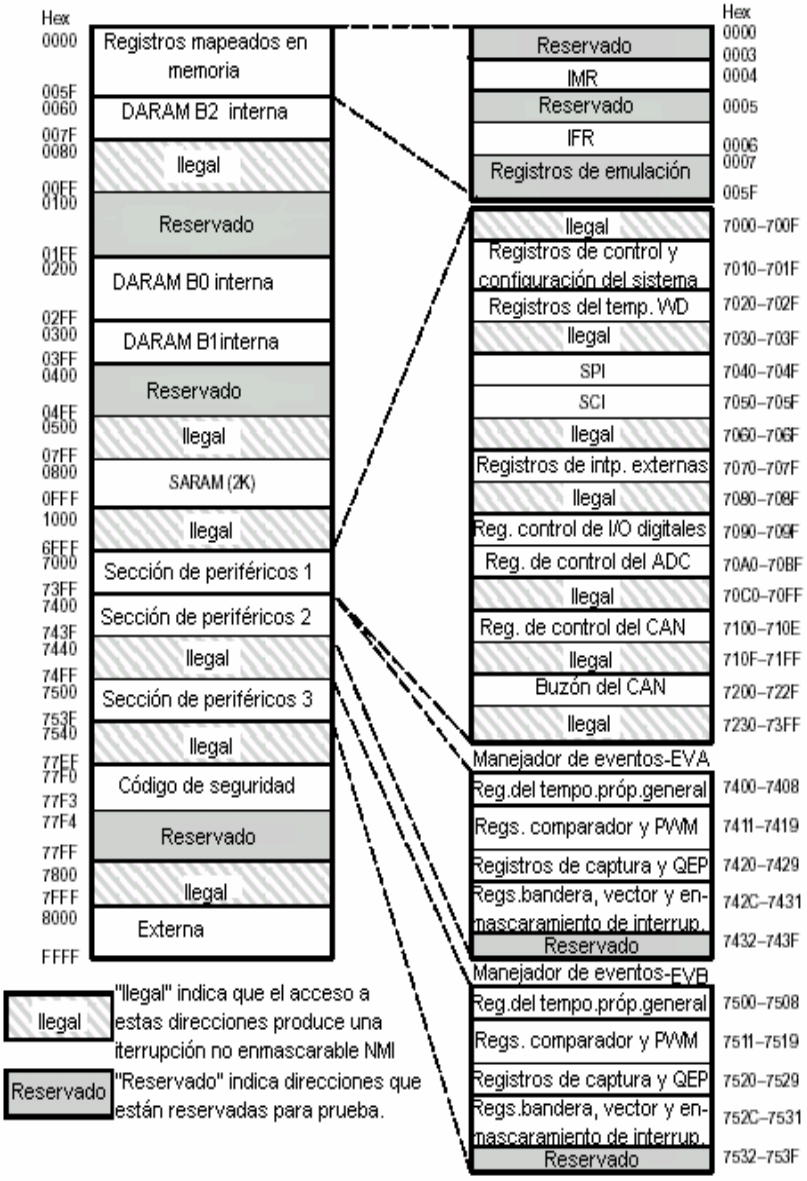
1.2.2 MEMORIA DE DATOS

La memoria de datos direcciona hasta 64k palabras de 16-bit. Esta memoria consiste de 32k palabras de memoria interna en el rango 0000h-7FFFh. La memoria interna de datos incluye los registros de control del DSP, la memoria DARAM y los registros de los periféricos mapeados en memoria. Las restantes 32k palabras de memoria (8000h-FFFFh) forman parte de la memoria de datos externa.

En la Figura 8 se muestra el diagrama de bloques de esta memoria, y en ella se puede ver que a través de la memoria de datos se controla y configura la operación del DSP LF2407A y de su módulo de evaluación. Esta configuración se realiza por medio del acceso a los registros mapeados en esta memoria, en los cuales se escriben los valores que se necesiten según las descripciones de los registros y las propiedades que ellos controlan.

Si se requiere de una información más detallada sobre todos los aspectos del módulo de evaluación se puede consultar la referencia [T.I evm2407b, 00], o si desea profundizar en cuanto a la arquitectura, direccionamiento, *CPU* y descripción de los registros del *DSP* LF2407A puede dirigirse a las referencias bibliográficas [T.I spru160c, 99] y [T.I spru357b, 01].

Figura 8. Memoria de datos



2 HERRAMIENTAS DE PROGRAMACIÓN

Una vez estudiado el hardware para la implementación del prototipo es necesario conocer cómo utilizarlo para poder lograr que funcione como un monitor de calidad de energía eléctrica. Para esto se utiliza el programa *Code Composer Studio C2xx*. Este software es la plataforma necesaria para desarrollar, comprobar y evaluar el algoritmo implementado en el DSP. En primer lugar se representarán las herramientas del lenguaje de *assembly*, las cuales se pueden utilizar fácilmente gracias a la plataforma *Code Composer Studio*.

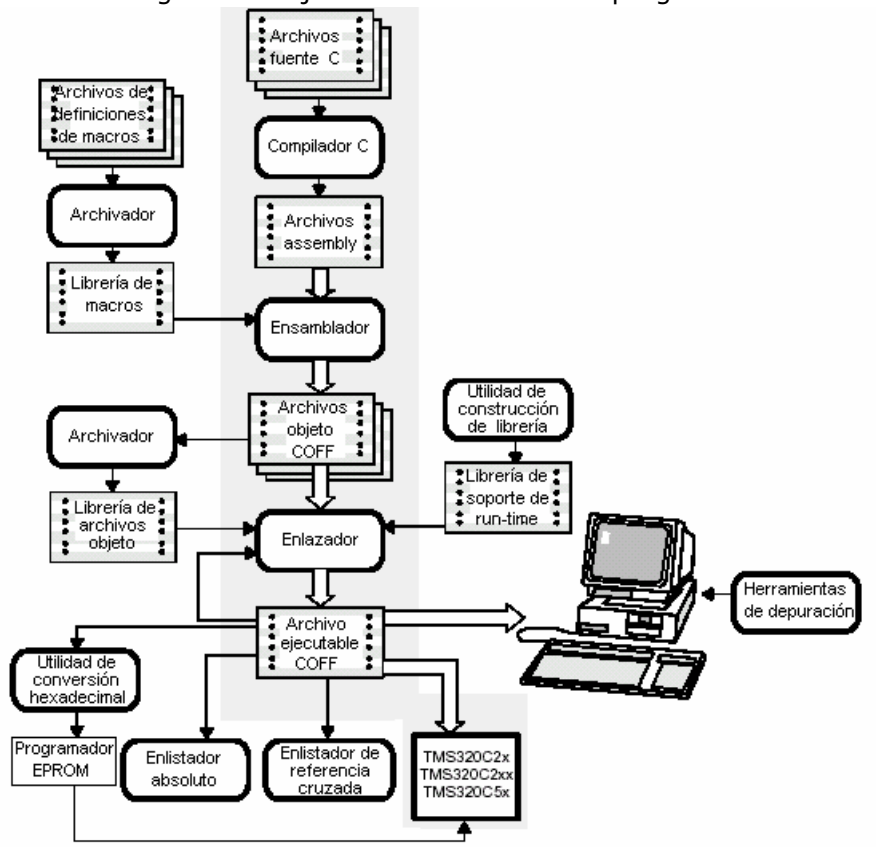
Las herramientas del lenguaje *assembly* son:

- El compilador de C: Acepta archivos en lenguaje C y produce archivos en lenguaje *assembly* para las series TMS320C2x, TMS320C2xx o TMS320C5x.
- El ensamblador: Traduce archivos en lenguaje assembly a archivos objeto en lenguaje de maquina. El lenguaje de maquina está basado en el formato de archivo de objeto común (*COFF*).
- El enlazador: Combina los archivos objeto en un módulo objeto ejecutable. Este crea el módulo ejecutable, desarrolla relocalizaciones y resuelve referencias externas. Una referencia externa se trata la utilización de una variable que se encuentra definida en otro archivo. El enlazador acepta archivos objeto COFF y librerías objeto como entradas.
- El archivador: Permite coleccionar un grupo de archivos en un solo archivo llamado librería. El archivador permite modificar una librería por medio de: la eliminación, reemplazo o adición de miembros.

- La utilidad de construcción de librería: Permite construir una librería de soporte de ejecución (*run-time*) personalizada.
- La utilidad de conversión hexadecimal: Convierte un archivo objeto *COFF* en un archivo objeto de formato *TI-Tagged*, *ASCII-hex*, *Intel*, *Motorola-S* o *Tektronix*. El archivo resultante puede ser cargado a un programador *EEPROM.*
- El enlistador absoluto: Es una herramienta de depuración. Acepta archivos objetos enlazados como entrada y crea archivos .abs como salida. Una vez ensamblados estos archivos producen listas que contienen las direcciones absolutas de las variables y funciones.
- El enlistador de referencia cruzada: Utiliza archivos objeto para producir un listado de referencia cruzada donde aparecen las variables, sus definiciones y su referencia para ser utilizada.

En la Figura 9 se muestra el diagrama de flujo del desarrollo de un programa. La columna vertebral de este desarrollo es la que se encuentra sombreada con gris y en las ramificaciones se muestra el uso de las herramientas de depuración y de construcción de librerías.

Figura 9. Flujo del desarrollo de un programa



2.1 HERRAMIENTAS DEL LENGUAJE ASSEMBLY

El DSP LF2407A soporta las siguientes herramientas del lenguaje *assembly*:

- Ensamblador.
- Compilador de C.
- Archivador.
- Enlazador.
- Enlistador absoluto.
- Enlistador de referencia cruzada.
- Utilidad de conversión hexadecimal.

Estas herramientas son las encargadas de generar los archivos objeto en formato común (*COFF*) los cuales facilitan la programación modular. Estos

archivos objeto contienen bloques separados de código y de datos (llamados secciones) que son los que uno carga en los espacios de memoria del DSP LF2407A.

A continuación se describe la función de las herramientas que se usaron en la implementación del prototipo:

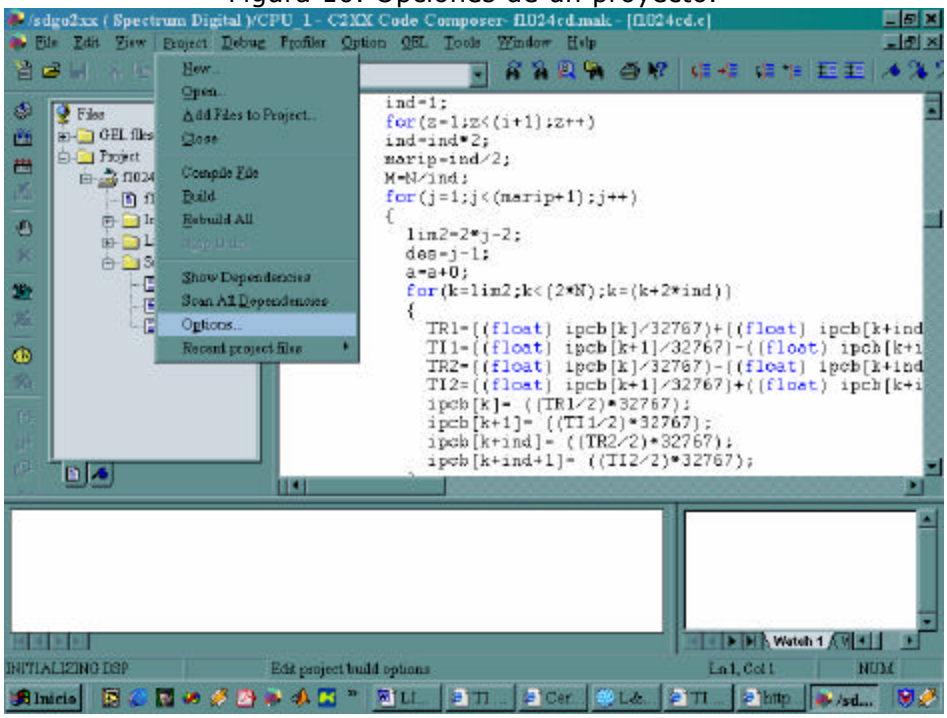
1. El Compilador de C: Esta herramienta se usa para "traducir" código fuente en lenguaje C a código fuente en el lenguaje *assembly* del DSP LF2407A.
2. El Ensamblador: Es el encargado de "traducir" archivos fuente en lenguaje assembly en archivos objeto en formato *COFF* de lenguaje de máquina. Los archivos fuente pueden contener instrucciones, directivas de ensamblador y directivas macro. Con las directivas del Ensamblador se pueden controlar varios aspectos del proceso de ensamblaje, entre las cuales están el formato del listado de fuente, la alineación de los datos y el contenido de las secciones de memoria.
3. El Enlazador: La función de esta herramienta consiste en combinar archivos objeto en un módulo objeto ejecutable de formato *COFF*. El enlazador realiza relocalizaciones y resuelve referencias externas. Las directivas del Enlazador permiten combinar secciones de archivos objeto, atar secciones o símbolos a direcciones o dentro de rangos de memorias y define o redefine símbolos globales.

A continuación se explicarán los aspectos más importantes (características, directivas, parámetros, definiciones, etc.) usados de cada una de las herramientas en esta implementación.

2.1.1 EL ENSAMBLADOR

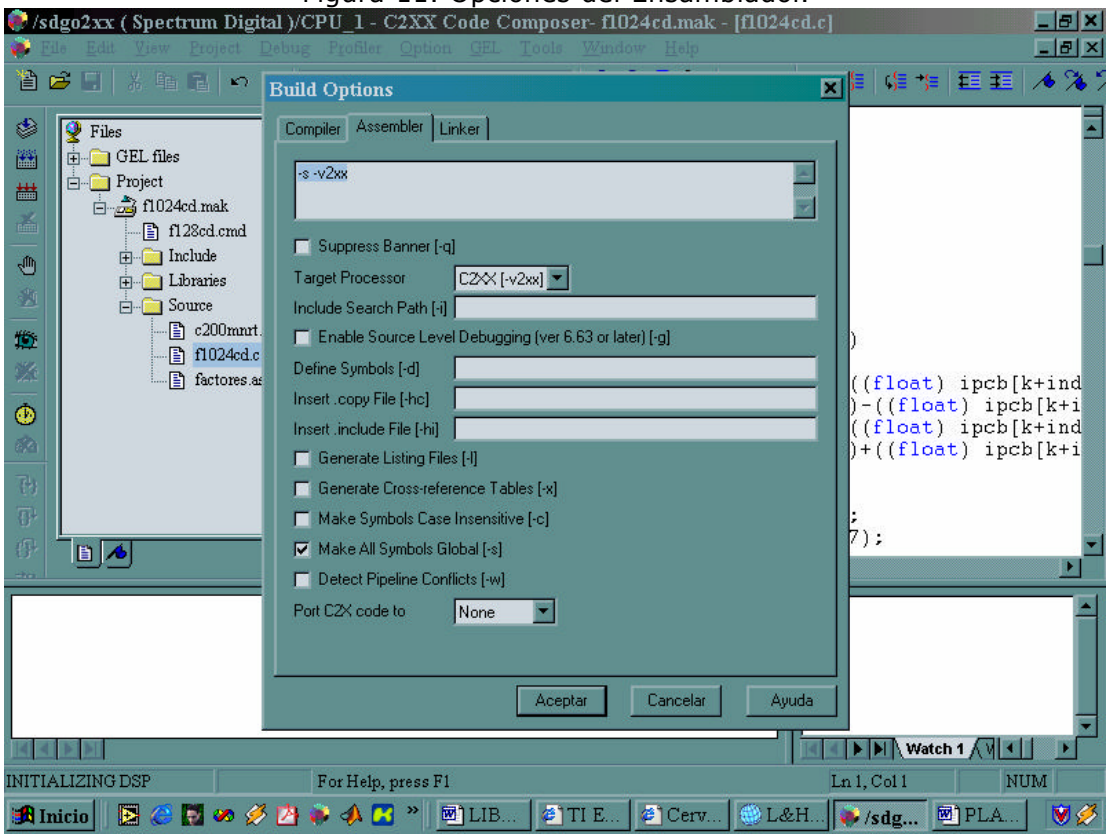
Esta herramienta permite definir los bloques denominados secciones (*sections*). Las secciones son las unidades que conforman un archivo objeto. Por lo tanto, estas secciones son bloques de código o datos que fundamentalmente ocuparán espacio en el mapa de memoria. Estas secciones están separadas y distinguidas, y además, el ensamblador y el enlazador permiten crear, nombrar y enlazar más secciones de las que son usadas por defecto [T.I spru018d, 95]. Las secciones pueden ser de dos tipos: inicializadas y sin inicializar. Las secciones sin inicializar son aquellas para las que el ensamblador reserva solamente espacio de memoria, no tienen ningún contenido en el archivo objeto, se localizan comúnmente en memoria RAM y son usadas para almacenar y crear variables. Las secciones inicializadas contienen código ejecutable o datos inicializados. Las 3 secciones que se tienen por defecto son: .text (se almacena el código ejecutable), .data (se almacenan datos iniciados) y .bss (variables sin iniciar).

Figura 10. Opciones de un proyecto.



Cuando se está construyendo un proyecto en el *Code Composer,* hay una sección a la cual se puede ir para seleccionar las diferentes opciones que se pueden llevar a cabo con el ensamblador, el enlazador y el compilador de C. Esta sección se encuentra en el submenú *Options* del menú *Project* de la ventana principal del *Code Composer*. Esto se puede observar en la Figura 10.

Figura 11. Opciones del Ensamblador.



En la Figura 11 se puede ver la ventana de selección de las opciones disponibles para el Ensamblador. Para terminar esta sección, se dará una breve explicación de la utilidad de cada una de las diferentes opciones usadas para la implementación del prototipo. Si se desea conocer la funcionalidad de las demás opciones, el formato de las sentencias en lenguaje *assembly*, las directivas que el ensamblador usa para manejar secciones, para manejar símbolos o variables, para manejar constantes,

para manejar *strings*, cómo se colocan y evalúan expresiones, los operadores que se usan y las instrucciones o comandos para programar en el *DSP* pueden consultarse los capítulos 2, 3, 4 y 5 de la referencia [T.I spru018d, 95]. Las opciones usadas fueron:

- Opción –s: Esta opción coloca todos los símbolos o variables definidas en la tabla de símbolos del archivo objeto. El ensamblador usualmente coloca solo variables globales dentro de la tabla de símbolos. Con –s los símbolos definidos como etiquetas o constantes creadas en el momento de ensamblaje también son colocados en la tabla. Una constante creada en el momento de ensamblaje se refiere a asignarle un valor constante a unos símbolos o etiquetas por medio de la directiva de ensamblador *.sect*. De este modo estos símbolos pueden ser usados más adelante en expresiones.
- Opción –v: Esta opción especifica una versión. La versión le indica al ensamblador para cuál de los siguientes dispositivos debe producir el código. Por defecto esta –v25:
  - –v10 selecciona el TMS320C1x
  - –v16 selecciona el TMS320C16
  - –v20 selecciona el TMS320C20
  - –v25 selecciona el TMS320C2x
  - –v2xx selecciona el TMS320C2xx
  - –v50 selecciona el TMS320C5x

2.1.2 EL ENLAZADOR

El enlazador tiene dos funciones principales. La primera consiste en utilizar las secciones de los archivos objeto *COFF* para formar bloques; el enlazador combina las secciones de entrada (cuando se está enlazando más de un archivo) para crear secciones de salida en un módulo ejecutable *COFF* de salida. Las secciones de entrada son las secciones que se definen por medio del ensamblador. Estas secciones de entrada se

encuentran en el programa que se desarrolle o en los archivos objeto que se le den como archivos de entrada. Las secciones de salida son las secciones que se definen con la directiva SECTIONS de el enlazador y son en donde se localizan las secciones de entrada. Estas secciones de salida se encuentran en el archivo objeto ejecutable que el desarrolla. La segunda función consiste en escoger las direcciones de memoria para las secciones de salida [T.I spru018d, 95].

En síntesis, el enlazador permite localizar secciones dentro de la memoria configurada del sistema, relocalizar símbolos y secciones para asignarles una dirección final y resuelve referencias externas indefinidas entre archivos de entrada. Por lo tanto, el enlazador controla la configuración de la memoria, la definición de secciones de salida y el direccionamiento de las secciones de salida en la memoria.

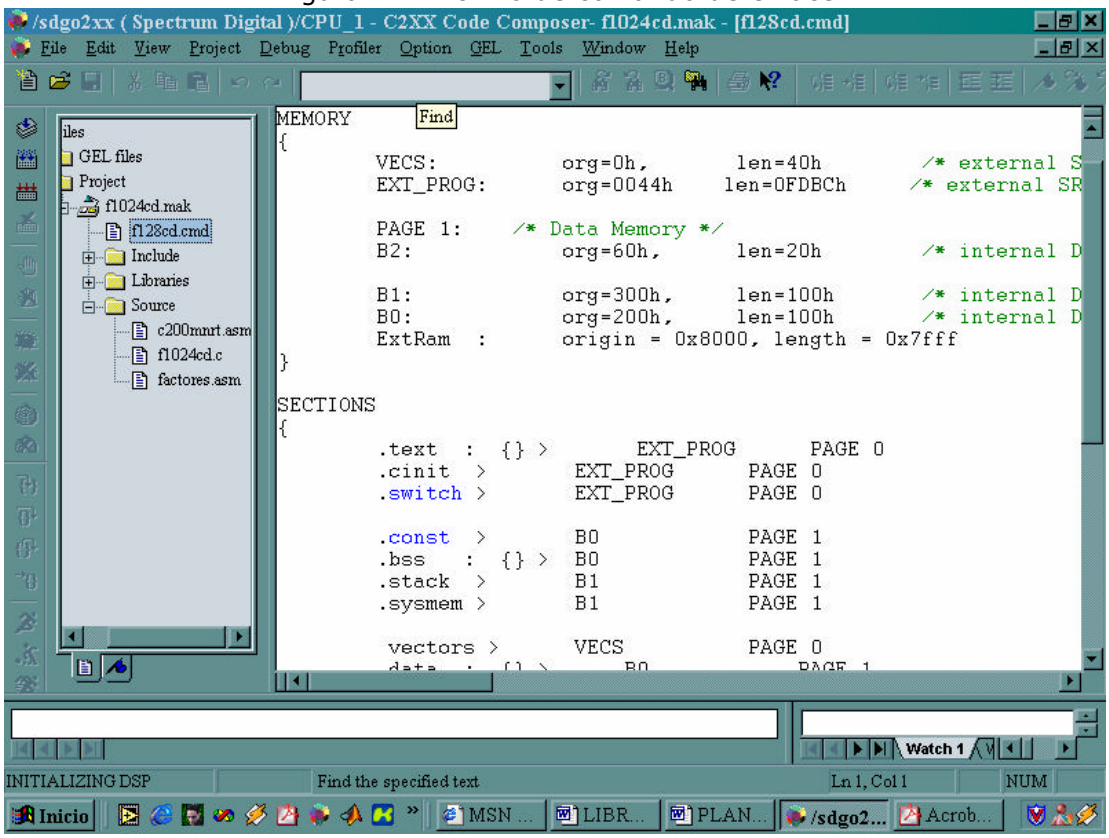
Para la configuración de memoria y la definición de secciones el enlazador soporta dos directivas:

- La directiva MEMORY: Esta directiva permite definir el mapa de memoria de un sistema designado. Con esta directiva se pueden nombrar porciones de memoria y especificar su dirección inicial y su longitud. El enlazador mantiene este modelo de memoria para localizar las secciones de salida y lo usa para determinar cual localización de memoria puede ser usada para código objeto.
- La directiva SECTIONS: Con esta directiva se le dice al enlazador como combinar las secciones de entrada y donde colocar las secciones de salida en la memoria.

El método para llamar al enlazador es de la siguiente manera: Es necesario colocar un archivo de comando enlazador, de tipo *enlazador.*cmd, en el cual se colocan las directivas MEMORY y SECTIONS para configurar la memoria del sistema de acuerdo con las necesidades

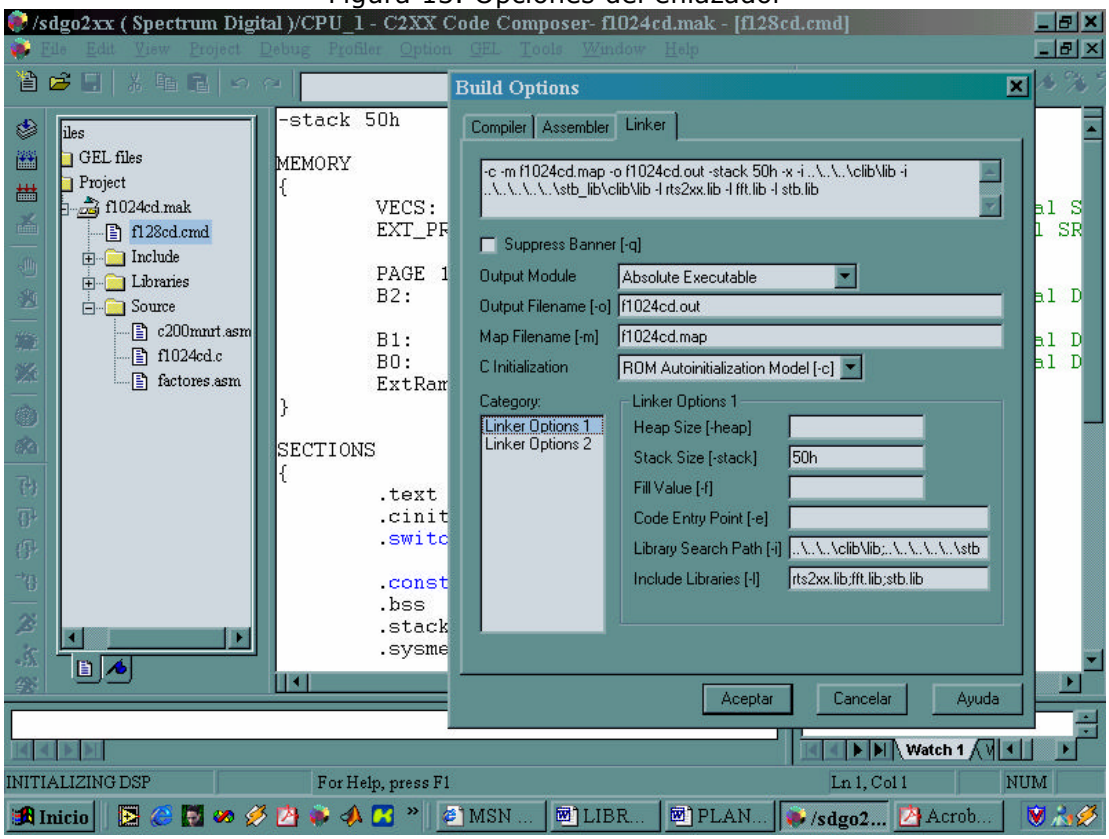
del usuario como se puede ver en la Figura 12. Si se desea entender la sintaxis de estas directivas es necesario consultar el capítulo 8 de la referencia [T.I spru018d, 95].

Figura 12. Archivo de comando de enlace



Al igual que el ensamblador, el enlazador posee varias opciones que pueden ser utilizadas en la construcción de un proyecto. A continuación se presentan las opciones utilizadas para el enlazador en la implementación del prototipo y se dejará al interés del lector para que profundice en estos aspectos en capítulo 8 de la referencia bibliográfica [T.I spru018d, 95], como ya he mencionado antes, para futuras aplicaciones.

Figura 13. Opciones del enlazador



En la Figura 13 se encuentra la ventana de selección de las posibles opciones disponibles para el enlazador. Las opciones usadas en la implementación del prototipo fueron:

- Opción -a: Sirve para seleccionar el modelo del archivo objeto de salida. Para este proyecto se uso un modelo ejecutable sin relocalización, es decir, absoluto. Una relocalización es cambiar la dirección de las definiciones de los símbolos. Esta relocalización se realiza debido a que, en el momento de enlazar los archivos objeto, a cada sección de memoria se le asignan nuevas direcciones diferentes a las que tenían, las cuales correspondían a las creadas cuando se estaba desarrollando el programa. Un archivo ejecutable contiene:

1. Símbolos especiales definidos por el enlazador.
2. Un encabezado opcional con información descriptiva, por ejemplo el punto de entrada del programa.
3. No tiene referencias sin resolver. Una referencia sin resolver aparece cuando una variable o función se llama pero no se encuentra definida los archivos fuente de entrada.

- Opción –o: Esta opción le asigna el nombre al modulo o archivo de salida que crea el enlazador si no hay ningún error.
- Opción –m: Esta opción crea un archivo mapa, en el cual se encuentra una lista de enlaces. El archivo mapa contiene:
    1. La configuración de la memoria.
    2. La localización de las secciones de entrada y de salida.
    3. Las direcciones de los símbolos externos después de que han sido re-localizados.
- Opción –c: Esta opción se utilizó para seleccionar el sistema de autoinicialización de variables en el momento de ejecución del programa. Esta autoinicialización es realizada por la función *c_int00*, donde *c_int00* es la rutina de inicio encargada de configurar un entorno para la ejecución de código C y se encuentra en el archivo *boot.asm* perteneciente a la librería de soporte de *run-time*. Cuando se programa en lenguaje C, el enlazador crea unas tablas de inicio para las variables globales. Estas tablas se encuentran en la sección *.cinit*. Esta autoinicialización realiza lo siguiente: carga la sección *.cinit* junto con las otras secciones en la memoria; el enlazador define el símbolo *cinit* el cual apunta al comienzo de las tablas de inicialización que se encuentran en la memoria y cuando el programa se ejecuta, la función *c_int00* copia los datos de estas tablas correspondientes a las variables que se encuentren en la sección *.bss*. Para entender a profundidad los modelos de auto-inicialización puede consultar el capítulo 4 (“Enlazando código C”) de la referencia [T.I spru024e, 99].

- Opción –i: Esta opción se usa para especificar la dirección o camino de búsqueda para las librerías incluidas en el programa.
- Opción –l: En esta opción se especifican las librerías que se van a usar en el programa.
- Opción –stack: El compilador de C del LF2407A emplea una sección sin inicializar *.stack* para localizar espacio para la pila de *software.* Con esta opción se establece el tamaño de esta sección.
- Opción –x: Normalmente el enlazador para crear el archivo objeto ejecutable lee los archivos de entrada, incluyendo las librerías, solo una vez. Cuando un archivo es leído, cualquier miembro que resuelve el llamado de una variable indefinida son incluidos en el enlace. Si un archivo de entrada más tarde llama una variable que fue definida en una lectura previa de una librería, la variable queda indefinida y no se puede crear el módulo de salida ejecutable. Por lo tanto esta opción obliga al enlazador a leer repetidamente todas las librerías hasta que todas las referencias o llamados estén resueltos.

2.1.3 EL COMPILADOR DE C

El compilador de C acepta código ANSI C, con extensión a C, y produce código fuente de lenguaje *assembly*. El compilador de C esta compuesto de: un programa esqueleto (*shell*), un optimizador y una utilidad de listado interno. Las funciones de cada uno de estos componentes son:

- El programa *shell* permite compilar automáticamente, ensamblar y enlazar módulos fuente.
- El optimizador modifica el código para mejorar la eficiencia de los programas en C.
- La utilidad del listado interno coloca en una lista las sentencias en código C con su correspondiente salida en lenguaje *assembly*.

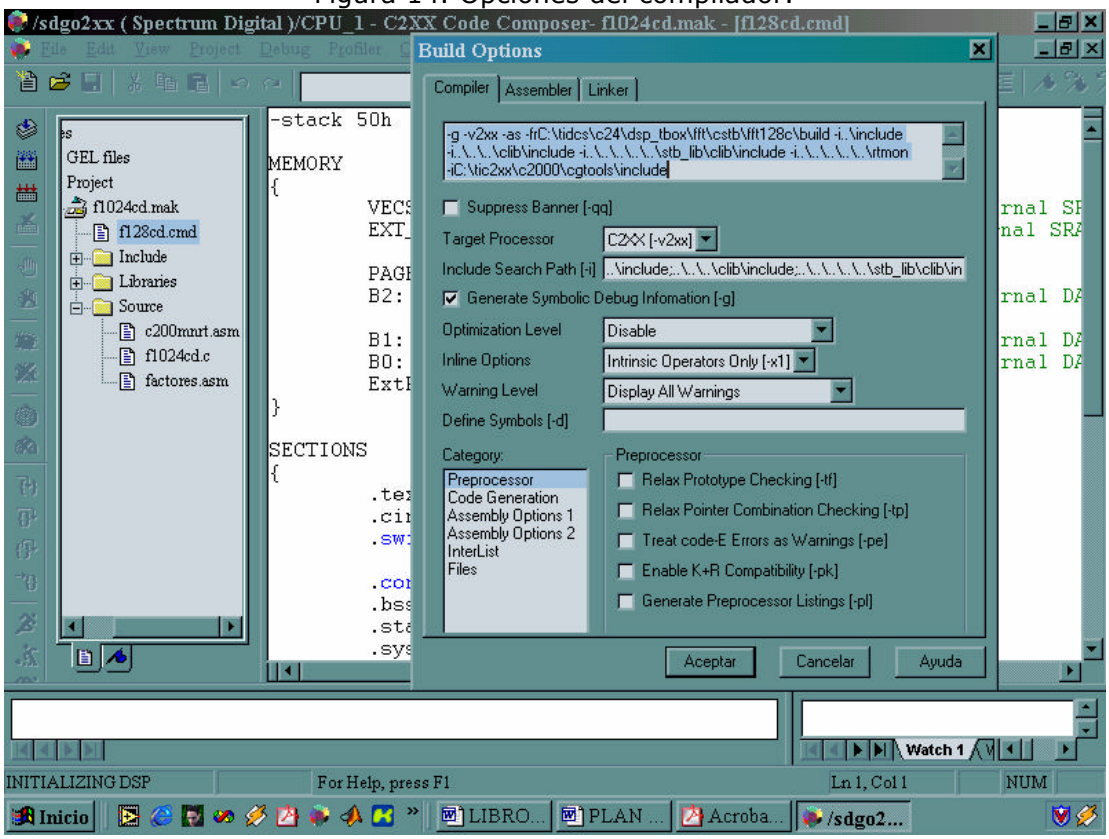
Entre las características principales del compilador se encuentran:

- El compilador acepta completamente el lenguaje estándar ANSI C y extensión a C.
- Las herramientas del compilador incluyen, para cada dispositivo, una librería de soporte para la ejecución del programa en lenguaje C (librería de soporte de *run-time*). En este caso el DSP LF2407A cuenta con la librería rts2xx.lib. Esta contiene todas las funciones de soporte de la librería estándar ANSI C; como son: el manejo de *string*, localización dinámica de la memoria, conversión de datos, medición de tiempo, trigonometría y funciones exponenciales e hiperbólicas. Esta librería también contiene la rutina de inicio para la configuración del entorno para la ejecución de un programa en código C.
- Como los archivos de objeto común (*COFF*) permiten definir el mapa de la memoria del sistema es posible enlazar código C y datos en áreas de memoria específicas.
- El compilador tiene convenciones de llamadas las cuales permiten escribir funciones en lenguaje C y en lenguaje *assembly* que se pueden invocar mutuamente.
- El compilador usa un paso de optimización que emplea varias técnicas avanzadas para una eficiente generación, compactando el código C.
- Para aplicaciones incluidas en modo *standalone* el compilador permite enlazar todo el código y los datos de inicialización dentro de la memoria *ROM*, permitiendo al código C ejecutarse desde reset. Esto se debe a que en la interrupción debida a un reset (interrupción 0, *int0*) se encuentra el llamado a la función *c_int00*. Está función, además de lo que se ha mencionado antes sobre el inicio del entorno para programas en lenguaje C y el inicio de variables, llama la función *main()* para que se comience a ejecutar el programa.

- Crea un listado de relación del código C original con el lenguaje *assembly* de salida, en el cual se puede inspeccionar el código *assembly* generado por cada sentencia de C.

En la figura 14 se encuentra la ventana de selección de las opciones del compilador en el entorno del *Code Composer*.

Figura 14. Opciones del compilador.



Como el algoritmo principal para el prototipo fue desarrollado en lenguaje C es necesario controlar el compilador a través de sus opciones. Las opciones usadas en la implementación del prototipo fueron:

- Opción –v: Esta opción especifica el tipo de procesador
    1. v25 para los procesadores TMS320LF/C2x
    2. v2xx para los procesadores TMS320LF/C2xx

3. v50 para los procesadores TMS320LF/C5x

- Opción -i: Esta opción agrega directorios a la lista de directorios en los que el compilador busca los archivos que están incluidos, estos archivos se incluyen con la sentencia *#include*.
- Opción –x: Sirve para controlar la expansión de una función en la línea donde es llamada. Esto significa que cuando se llame una función que haya sido declarada con el símbolo _INLINE del pre-procesador, el código fuente C de la función es insertado en el punto de llamado. Para el prototipo se selecciono la opción –x1 que solamente expande los operadores intrínsecos (abs, fabs y labs) ya que no se uso el símbolo _INLINE. Estos operadores son funciones intrínsecas del C y se encuentran en la librería estándar ANSI C. La función *abs()* calcula el valor absoluto de un entero y está declarada en el archivo de encabezado *stdlib.h*, *labs()* calcula el valor absoluto de un entero largo y también está declarada en el archivo de encabezado *stdlib.h* y *fabs()* calcula el valor absoluto de un número de coma flotante y está declarada en el archivo de encabezado *math.h*
- Opción –pw: Esta opción determina el nivel de mensaje de advertencia y errores a mostrar. Para el prototipo fueron habilitadas todas las advertencias y errores generadas durante la compilación del programa (opción –pw2).
- Opción –as: Invoca el ensamblador con la opción –s del ensamblador para colocar las etiquetas (*labels*) en la tabla de símbolos.
- Opción –fr: Esta opción especifica el directorio para los archivos objeto.

Al igual que las otras herramientas del lenguaje *assembly*, si se requiere una información más profunda sobre el compilador de C y de sus opciones

puede consultarse la referencia [T.I spru024e, 99] la cual trata toda sobre el compilador.

2.2 CODE COMPOSER STUDIO

El modulo de evaluación TMS320LF2407 EVM de *Texas Instruments* posee un software asociado llamado *Code Composer Studio*, el cual es un entorno completo de desarrollo, de comprobación y de evaluación.

El primer paso para utilizar el *Code Composer Studio* es su configuración a través de el *Setup* CC C200. Con este se establece la interfaz que permite que el *Code Composer* se comunique con el modulo de evaluación del *DSP*. El *Setup* define la configuración del sistema estableciendo los siguientes aspectos:

- La tarjeta o modulo de evaluación.
- El emulador o el simulador.
- El tipo de procesador (*DSP*).
- Los parámetros de comunicación.

Gracias al entorno del *Code Composer*, entorno de ventanas, el desarrollo de programas para el *DSP* es más sencillo y visual. Esta característica lo hace una herramienta versátil y fácil se usar. A continuación se estudiaran las herramientas y características principales del *Code Composer* utilizadas para llevar a cabo la implementación del prototipo.

2.2.1 EDITOR

Esta capacidad le permite al usuario crear nuevos archivos, abrir, guardar e imprimir archivos, abrir múltiples ventanas, dividir ventanas, duplicar las ventanas de los archivos, resaltar sentencias del algoritmo con diferentes colores, buscar y remplazar texto, buscar una palabra seleccionada, cortar, copiar, pegar y borrar texto, deshacer, rehacer, ir a

una sentencia, editar columnas, cambiar el color y tamaño del texto entre otras.

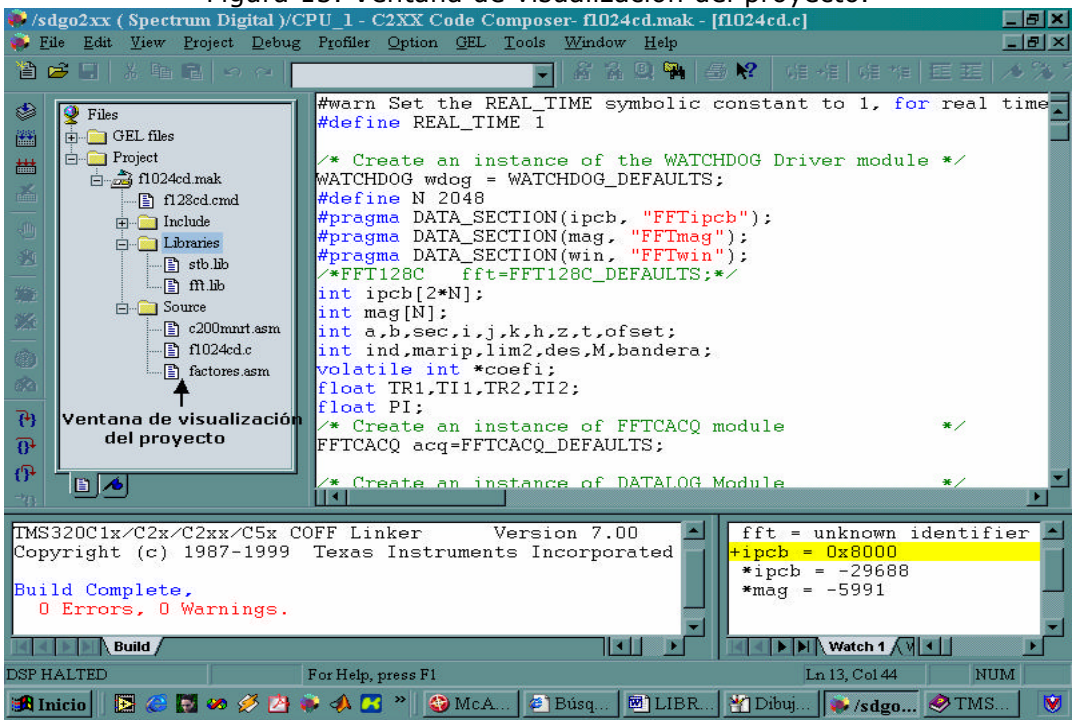
2.2.2 ENTORNO DEL PROYECTO

El *Code Composer* posee una herramienta para gestionar programas para los proyectos. Este manejador de proyectos realiza lo siguiente:

- Mantiene los archivos fuente y librerías objeto necesarias para construir un programa o librería para el DSP. El manejador de proyectos identifica los archivos por sus extensiones. Considera las opciones *.* y *.c para los archivos en lenguaje C (el manejador los compila y enlaza), *.a y *.s para los archivos en lenguaje *assembly* (el manejador los ensambla y los enlaza), *.o para archivos objeto y *.lib para las librerías (las cuales enlaza) y *.cmd para el archivo de comando de enlazador (el manejador enlaza con este archivo), el cual es uno solo para cada proyecto. Los archivos de encabezado (*.h) son agregados automáticamente por medio del escaneo de las dependencias de los archivos fuente. Asimismo el manejador también permite agregar y remover archivos (de tipo *.c, *.a, *.cmd y *.lib) del proyecto.
- Permite crear, abrir y cerrar proyectos. También permite visualizar el contenido del proyecto en la ventana de visualización del proyecto que sale a la izquierda de la pantalla. En esta ventana los archivos que están presentes en el proyecto aparecen en forma de lista agrupados en diferentes carpetas, como se puede ver en la Figura 15. En la carpeta *include* están los archivos incluidos y encabezados (*.h), en la carpeta *libraries* están las librerías objeto (*.lib) y en la carpeta *source* se encuentran los archivos fuente (*.c, *.a y *.s). El archivo de comando de enlace aparece directamente en la carpeta del proyecto.
- Permite seleccionar las opciones del compilador, ensamblador y el enlazador utilizadas para construir el programa o la librería. Para

la construcción de los programas el manejador de proyectos tiene varias opciones: compilar el archivo sin enlazar (*compile file*), compilar y enlazar solo los archivos que han sido modificados desde su última construcción (*incremental build*), recompilar y reenlazar (*rebuild all*) y detener la construcción (*stop build*).

Figura 15. Ventana de visualización del proyecto.

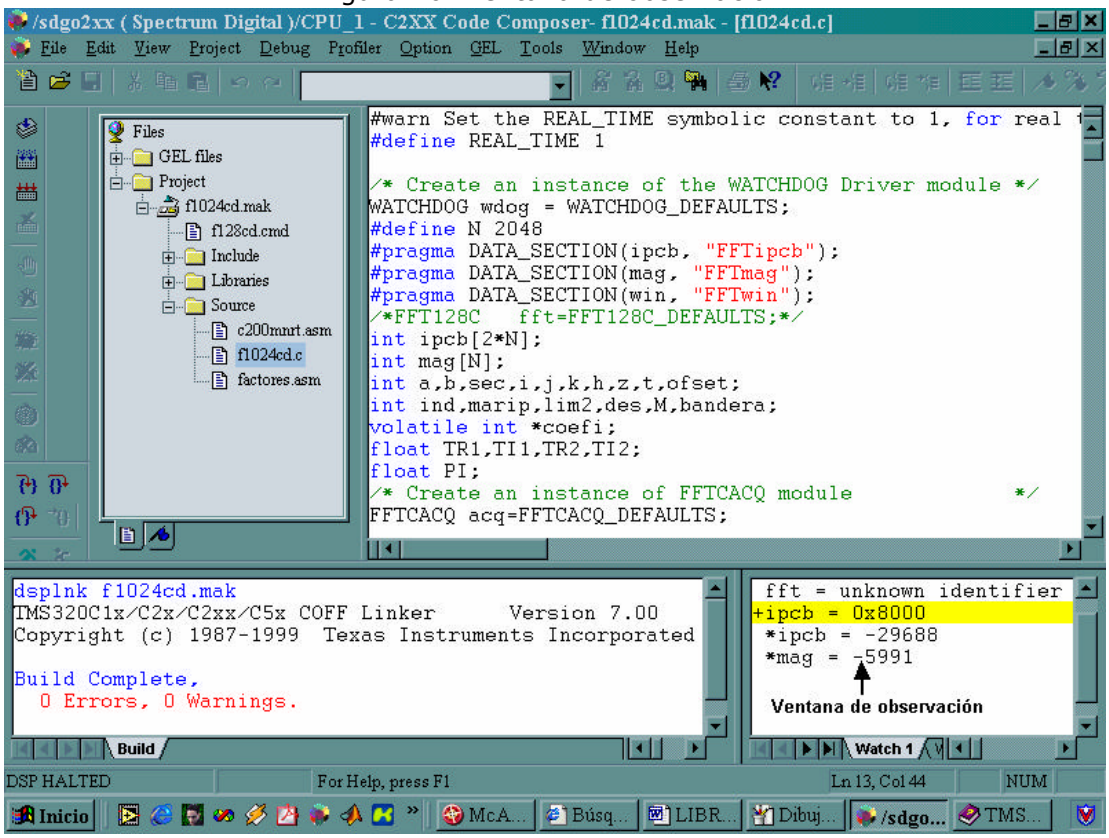


2.2.3 VENTANA DE OBSERVACIÓN

La ventana de observación permite examinar y editar variables y expresiones de C. Esta ventana de observación se despliega con el submenú *View->Watch Window* o con el icono de acceso directo en la barra de herramientas del depurador (*Debug Toolbar*). En la ventana de observación se pueden expandir o colapsar expresiones complejas (vectores), esto se logra dando *click* izquierdo con el *mouse* en el signo "+" que aparece al lado de la variable. También se pueden evaluar términos o expresiones y mostrar resultados en diferentes formatos. Por defecto al ingresar una variable se muestra su dirección de localización en

el mapa de memoria, si se desea saber que valor tiene debe ingresarse la variable precedida por un asterisco "*". Esto se puede observar en la Figura 16.

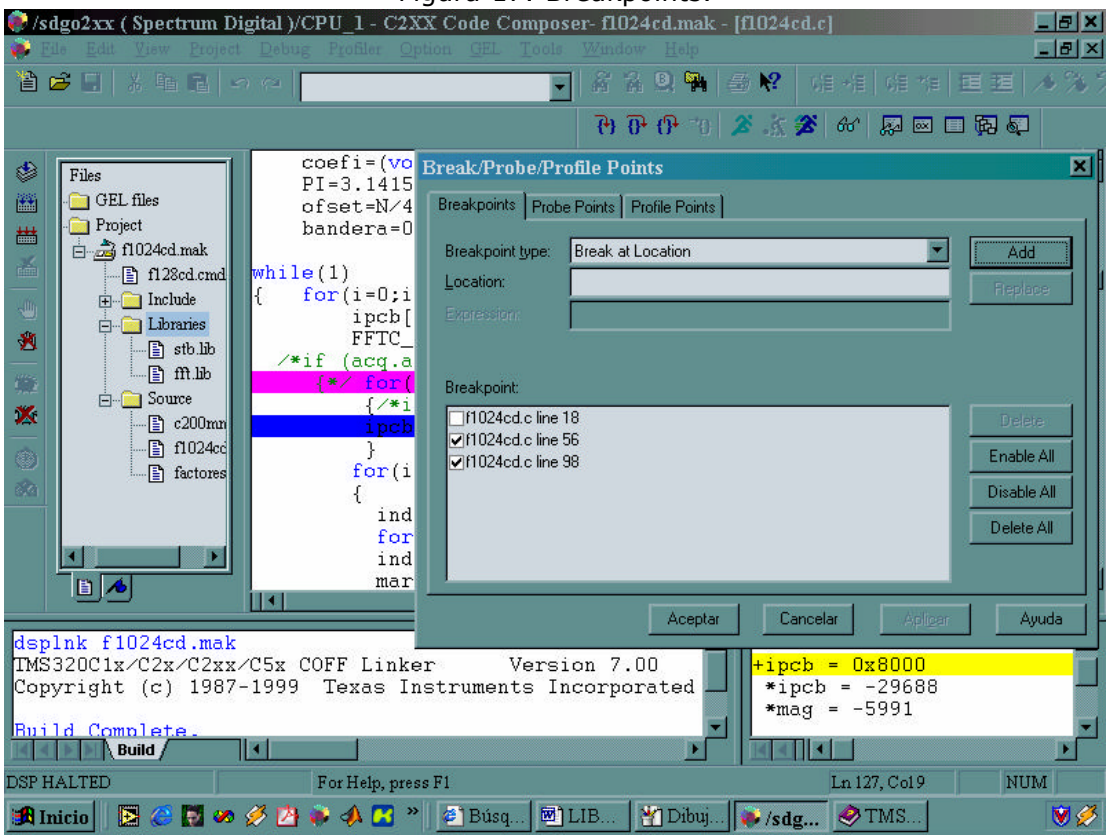
Figura 16. Ventana de observación.



2.2.4 PUNTOS DE PARADA (BREAKPOINTS)

Los *breakpoints* detienen la ejecución del programa. Cuando el programa es detenido, se puede examinar el estado del programa, examinar o modificar variables, examinar el llamado de la pila, etc. Estos *breakpoints* se pueden establecer a través del acceso directo, con la barra de herramientas del manejador de proyectos (*Project Toolbar*) o con el submenú *Debug->Breakpoints* del *Code Composer* como se muestra en la Figura 17. La localización de los *breakpoints* aparece en color violeta en los archivos fuente en los cuales se encuentran.

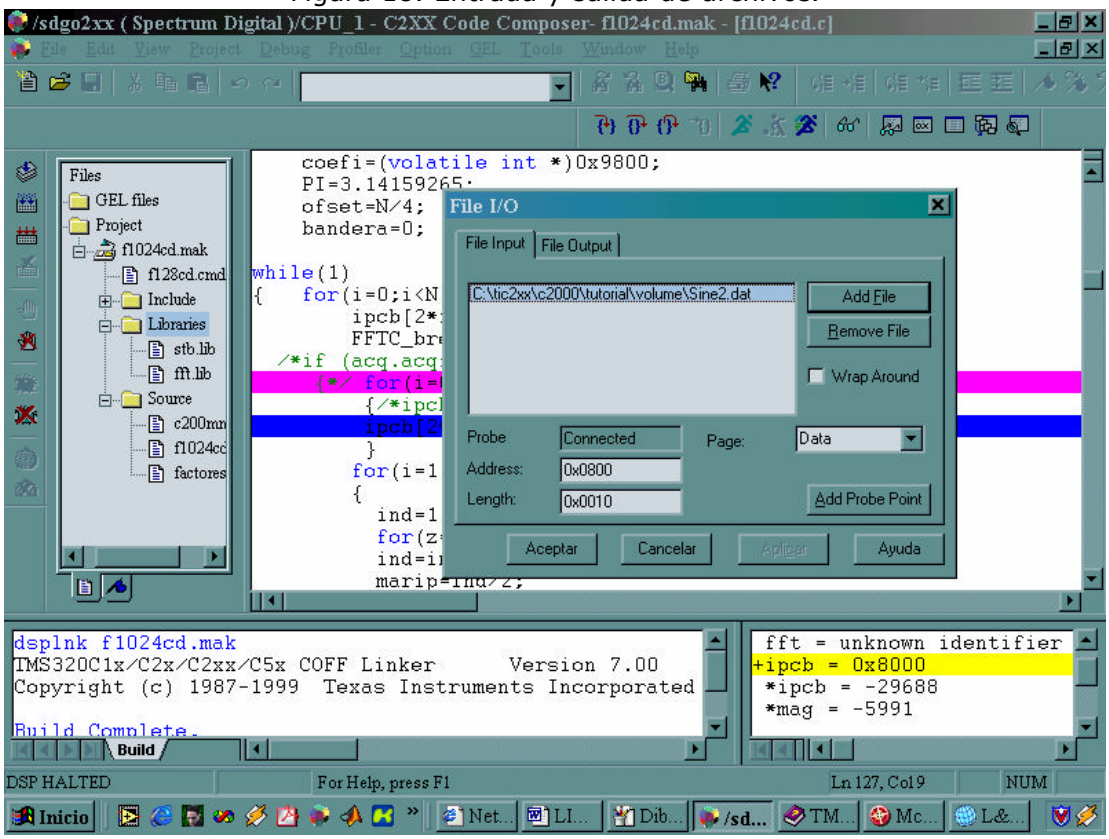
Figura 17. Breakpoints.



2.2.5 PUNTOS DE PRUEBA (PROBE POINTS)

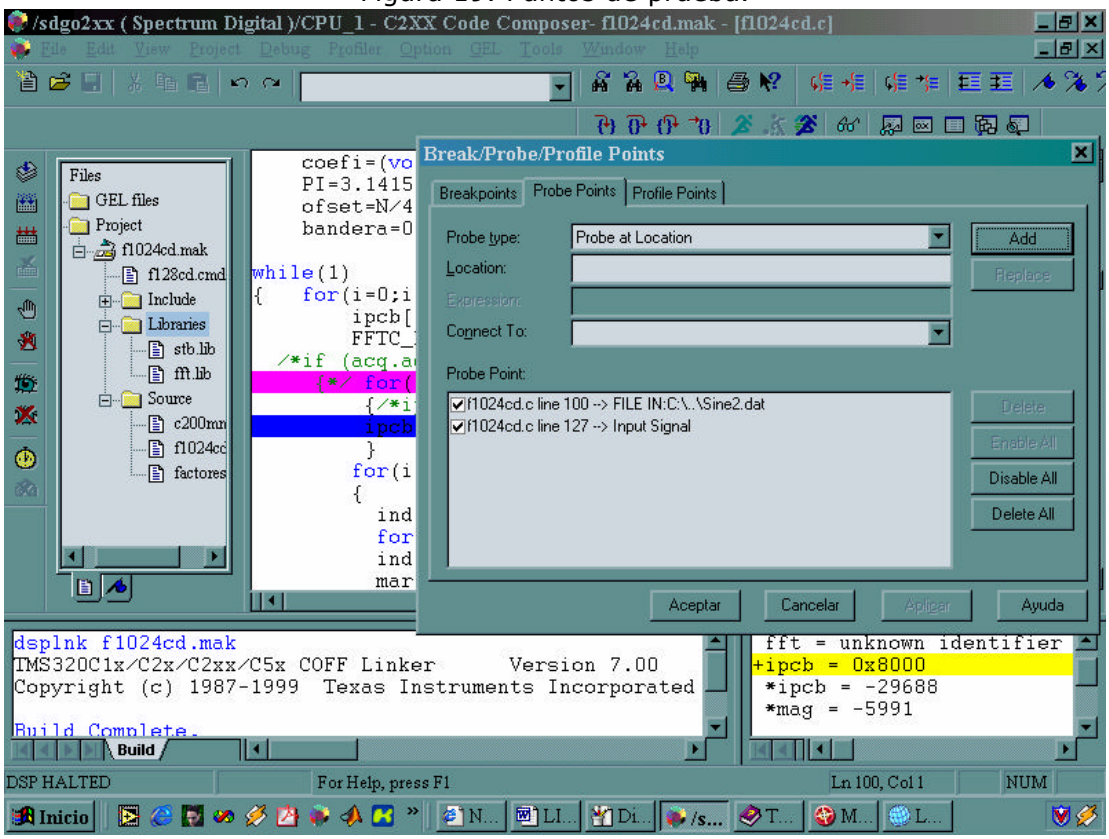
Estos puntos de prueba permiten llevar a cabo actualizaciones de una ventana particular o leer y escribir muestras de un archivo hacia un punto específico del algoritmo.

Figura 18. Entrada y salida de archivos.



Aprovechando las capacidades de entrada y salida de archivos (*File I/O*, figura 18) del *Code Composer*, se pueden conectar flujos de datos a un punto particular en el código del *DSP* y cuando el punto se alcance en el algoritmo, los datos son llevados de un área de memoria específica a el archivo o del archivo a la memoria. Al igual que los *breakpoints*, los puntos de prueba se colocan a través de su acceso directo o con el submenú *Debug->Probe Points.* Estos puntos de prueba aparecen de color azul en el código fuente como se muestra en la Figura 19.

Figura 19. Puntos de prueba.

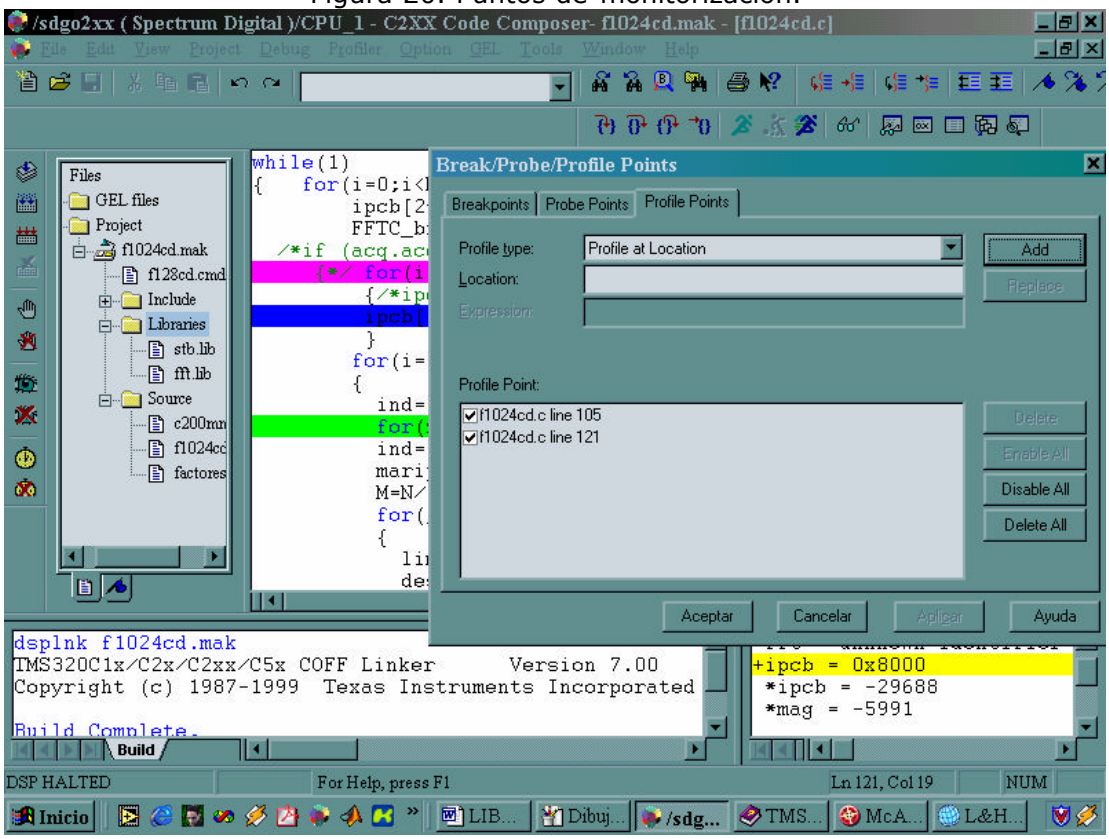


2.2.6 MONITORIZACIÓN DE LA EJECUCIÓN DEL CÓDIGO EN EL DSP

El *Code Composer* permite recopilar estadísticas de la ejecución del código. Esto es útil ya que ofrece retroalimentación inmediata del desempeño de la aplicación que se esté realizando, permitiendo optimizar el código. Se puede determinar, por ejemplo, cuánto tiempo de *CPU* del *DSP* emplean los diferentes algoritmos. También se pueden monitorizar otros eventos del procesador, entre los cuales se cuentan el número de saltos, el número de llamados de subrutinas o el número de interrupciones realizadas. Los puntos de monitorización se establecen con su acceso directo en la barra de herramientas del manejador de proyectos (Project Toolbar) o por medio del submenú *Profiler->Profile points* del

Code Composer. Estos puntos de monitorización aparecen en verde en el código fuente como se muestra en la Figura 20.

Figura 20. Puntos de monitorización.



2.2.7 LENGUAJE DE EXTENSIÓN GENERAL (GEL)

Este es un lenguaje interpretativo similar al lenguaje C, el cual permite crear funciones para extender la utilidad del *Code Composer*. Es posible crear funciones *GEL* usando la gramática *GE*L y luego cargarlas en el *Code Composer*. Con el *GEL*, se puede acceder a las localizaciones de memoria y agregar opciones al menú *GEL* del *Code Composer*. El *GEL* es particularmente útil para pruebas automatizadas y para personalizar del espacio de trabajo en el *Code Composer*.

2.2.8 CONTROL AVANZADO DE PROGRAMACIÓN

Estas características del *Code Composer* se llevan a cabo con el submenú *Debug* o con los iconos de acceso directo de la barra de herramientas del depurador (*Debug Toolbar*). Las acciones que se desarrollan con este submenú son:

- La ejecución de un programa (*Run*).
- Detener la ejecución del programa (*Halt*).
- Animar el programa (*Animate*). Por animación se entiende la ejecución del programa hasta que encuentre un *breakpoint* o se detenga su ejecución. También se actualizan las ventanas que no estén conectadas a puntos de prueba y se reanuda la ejecución. El *Code Composer* permite establecer la velocidad de animación.
- Le ejecución libre (*Run free*). Este comando inhabilita todos los *brekpoints*, los puntos de prueba y los puntos de monitorización antes de ejecutar el programa, empezando desde la ubicación actual del contador de programa (*PC*). Cualquier operación mientras se ejecuta libremente el programa reestablece los *breakpoints.* La ejecución se detiene con el comando *Halt*. También se puede realizar un *reset* vía *hardware.*
- Reiniciar el procesador de la tarjeta. Esto se logra con tres comandos. El primero es el *Reset DSP*, el cual inicializa todos los registros a sus estados de encendido y detiene la ejecución del programa. El segundo comando es el *Restart*, con este se restaura el *PC* al punto de entrada del programa cargado en el *DSP* pero no se reanuda la ejecución del programa. El tercero y último es el *Go Main*, este comando establece un *breakpoint* temporal en la función *main* del programa y comienza la ejecución. El *breakpoint* es removido hasta que otro *breakpoint* es encontrado o hasta que se detiene la ejecución.

2.2.9 MANEJADOR DE DEPURACIÓN PARALELA

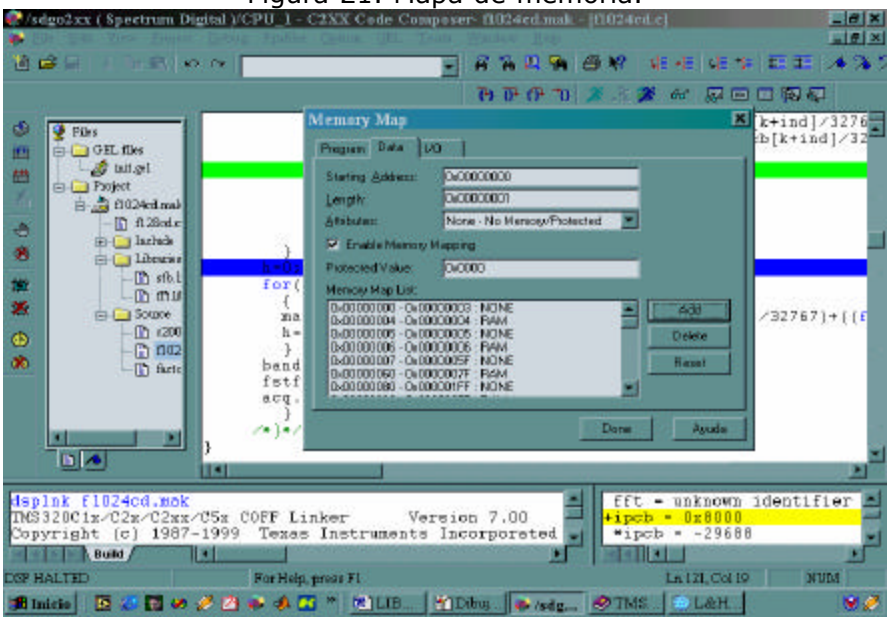
El manejador de depuración paralela es usado para transmitir comandos a múltiples procesadores y para seleccionarlos a cada uno de ellos individualmente. Esta opción es posible solo con el emulador y no con el simulador.

2.2.10 MAPA DE MEMORIA

Esta herramienta le indica al depurador del *Code Composer* cuáles áreas de memoria el puede acceder o no. Típicamente, el mapa de memoria coincide con la definición de memoria hecha en el archivo de comando de enlace. Cuando se habilita el mapa de memoria, el depurador del *Code Composer* verifica cada uno de los accesos de memoria con el mapa de memoria suministrado. Cuando se trata de acceder un área protegida o indefinida, el depurador muestra el valor que tiene por defecto y no trata de acceder a ella. Las verificaciones que hace el *Code Composer* son por *software* y no por *hardware*, por lo tanto no es posible evitar que el programa intente acceder memoria que no existe. Para definir un rango del mapa de memoria válido para el sistema se tienen dos opciones:

- Ingresar los comandos interactivamente mientras se está usando el depurador. Esto se logra usando el submenú *Option->Memory Map* del *Code Composer*, con el cual aparece una ventana de configuración, como se ve en la Figura 21, donde se selecciona cuál memoria se desea definir (memoria de programa, datos o *I/O*) y se establecen las direcciones, longitud y atributos específicos.

Figura 21. Mapa de memoria.



- Utilizando las funciones *GEL* ya construidas. El depurador provee un conjunto completo de comandos para el mapeo de la memoria los cuales pueden ser invocados por medio del lenguaje de extensión general y el submenú *GEL*. Esto explica el propósito de la carpeta GEL que aparece en la parte superior de la ventana de visualización del proyecto. Una vez inicia el *Code Composer*, éste carga el archivo *init.gel* en el cual está definido el mapa de memoria asumiendo que se está trabajando en modo de microprocesador. Esta definición se hace a través de la función *StartUp()*, la cual es llamada cada vez que se inicia el *Code Composer*. Lo anteriormente explicado se puede observar en las Figuras 22 y 23.

Figura 22. Función *StartUp* del archivo *init.gel.*

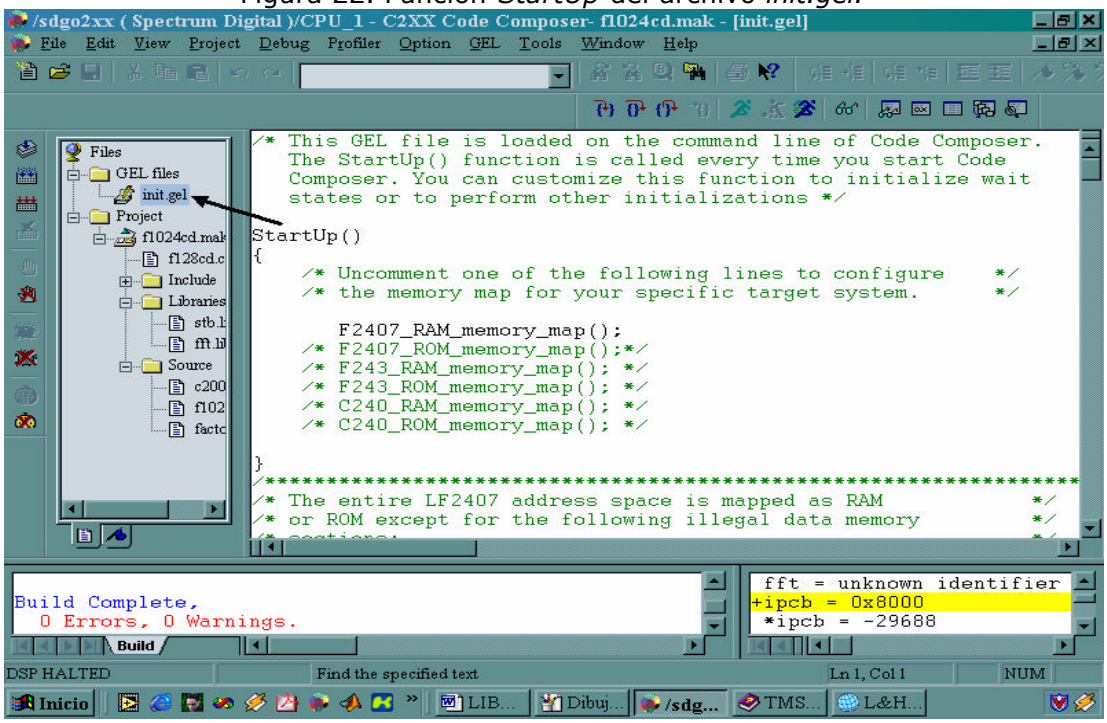
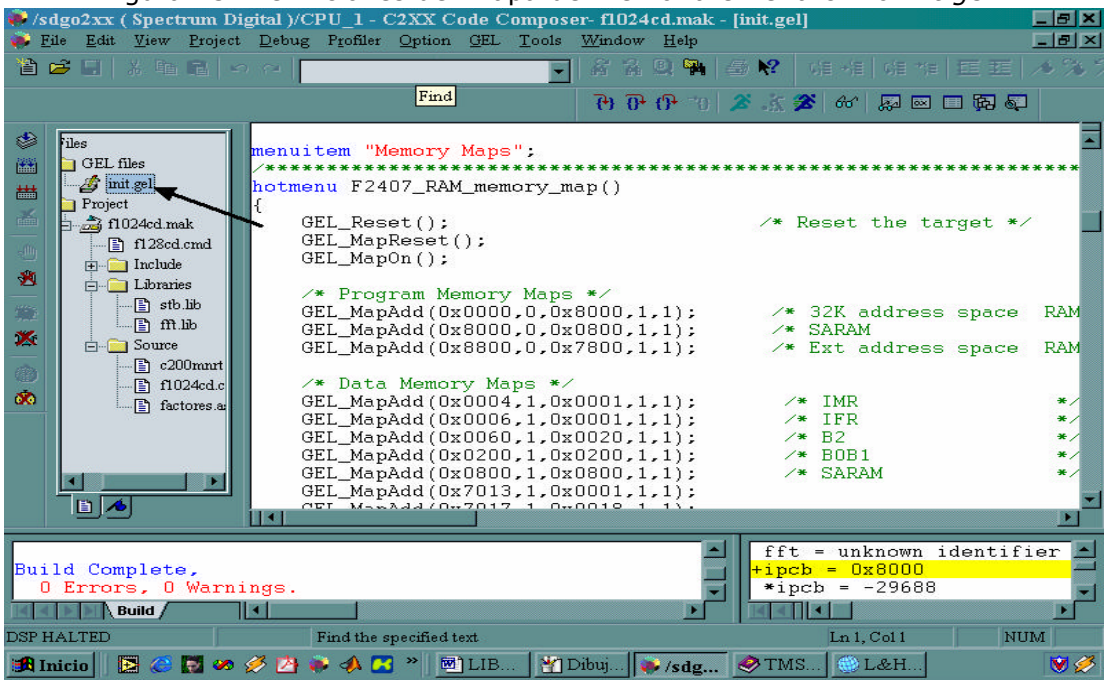


Figura 23. Definiciones del mapa de memoria en el archivo *init.gel.*



2.2.11 EJECUCIÓN POR PASOS

Esta es una herramienta importante para el desarrollo de los programas en el *DSP*, y fue una de las más utilizadas para la depuración de errores, para la evaluación y para la comprobación del algoritmo desarrollado para la implementación del prototipo. La utilización de las opciones de esta herramienta se logra a través del submenú *Debug* del *Code Composer* o por medio de los iconos de acceso directo de la barra de herramientas del depurador (*Debug Toolbar*). Para esta herramienta de seguimiento de la ejecución del programa se tienen las siguientes opciones:

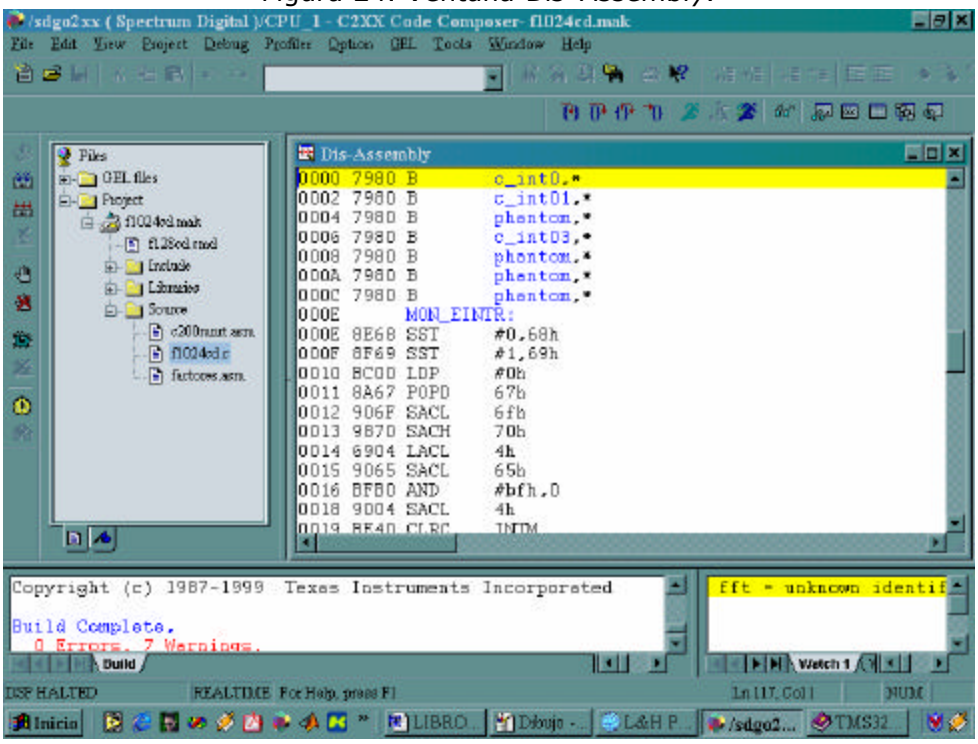
- *Step Into*: Con este comando se realiza un solo paso a través del código. Tanto en código de lenguaje C como en código de lenguaje *assembly* se ejecuta solamente una instrucción.
- *Step Over*: Este comando se emplea para realizar una función o instrucción dentro de un ciclo repetitivo, ejecutando cada sentencia de la función o de la instrucción individualmente. Si se encuentra un llamado de una función, la función se ejecuta hasta completarse.
- *Step Out*: Si la ejecución se encuentra dentro de una subrutina, este comando puede ser seleccionado para completar la ejecución de la subrutina. La ejecución se detiene después de que la función actual regresa a la función que hizo el llamado.
- *Run to Cursor*: Esta opción se usa para ejecutar el programa cargado en el *DSP* hasta encontrar la posición del cursor en la ventana *Dis-Assembly.*

2.2.12 VENTANA DIS-ASSEMBLY

Cuando se carga un programa en el *DSP*, el depurador del *Code Composer* abre automáticamente una ventana de salida *Dis-Assembly*. Esta ventana muestra las instrucciones desmontadas y la información simbólica necesaria para la depuración. El desmontaje es el proceso inverso del ensamblaje y permite que los contenidos de la memoria sean

visualizados como código de lenguaje *assembly*. La información simbólica consiste de símbolos y *strings* o caracteres alfanuméricos que representan direcciones o valores en el *DSP*.

Figura 24. Ventana *Dis-Assembly*.



La información que muestra la ventana *Dis-Assembly*, como se observa en la Figura 24, esta organizada de esta manera: En la primera columna se encuentra la dirección en la cual la instrucción está localizada, en la segunda columna se encuentra el *opcode* (código de máquina que representa la instrucción) y en la tercera columna se encuentra la instrucción desmontada, es decir, en lenguaje *assembly*.

2.2.13 VENTANAS DE VISUALIZACIÓN DE VARIABLES

Para terminar existen cuatro herramientas de visualización. La primera de ellas es la ventana de visualización de memoria. Esta se despliega con el submenú *View->Memory* o con el ícono de acceso directo que se encuentra en la barra de herramientas del depurador (*Debug toolbar*) y

aparece en pantalla como se muestra en la Figura 25. Con esta ventana se puede seleccionar la página de memoria a visualizar, la dirección de inicio, su formato y su valor Q. El valor del formato Q es lo mismo que la notación de coma fija del *DSP*. Este formato es usado para representar fracciones.

Figura 25. Ventana de memoria.

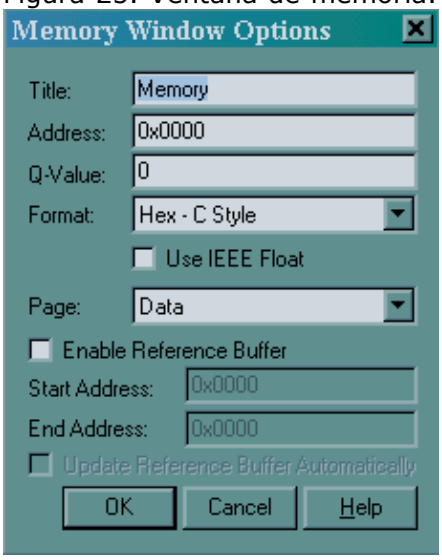
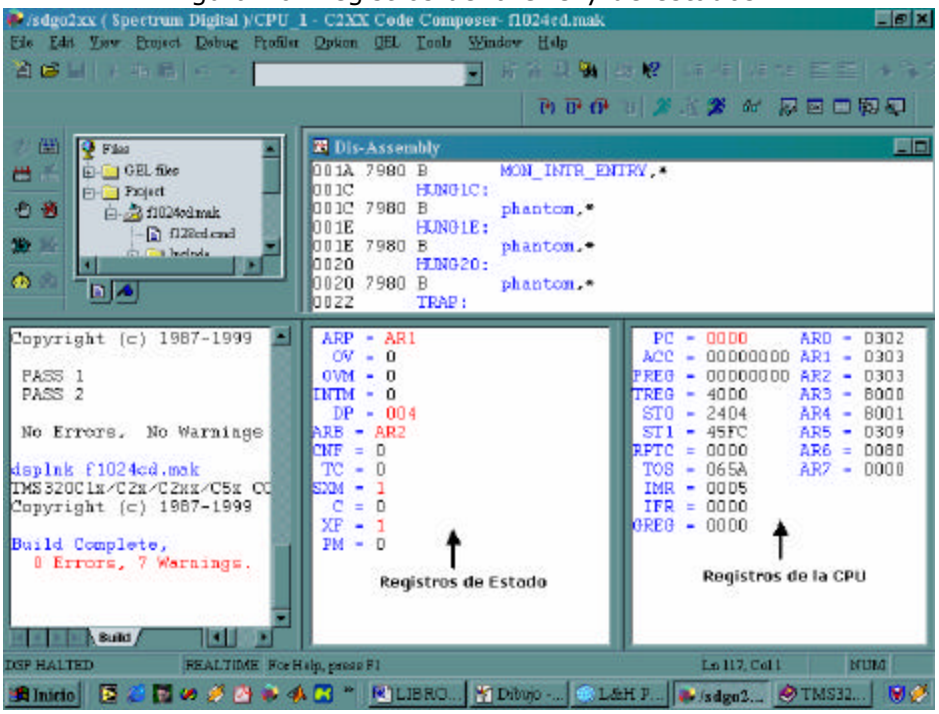


Figura 26. Registros de la *CPU* y de estados.



La segunda y la tercera son ventanas de visualización de los registros del DSP. Estas ventanas, las cuales se muestran en la Figura 26, son la ventana de visualización de los registros de la *CPU* del *DSP* y la ventana de visualización de los registros de estado del *DSP*. Para desplegar estas ventanas se utiliza el submenú *View->CPU Registres* o los íconos de acceso directo en la barra de herramientas del depurador (*Debug Toolsbar*).

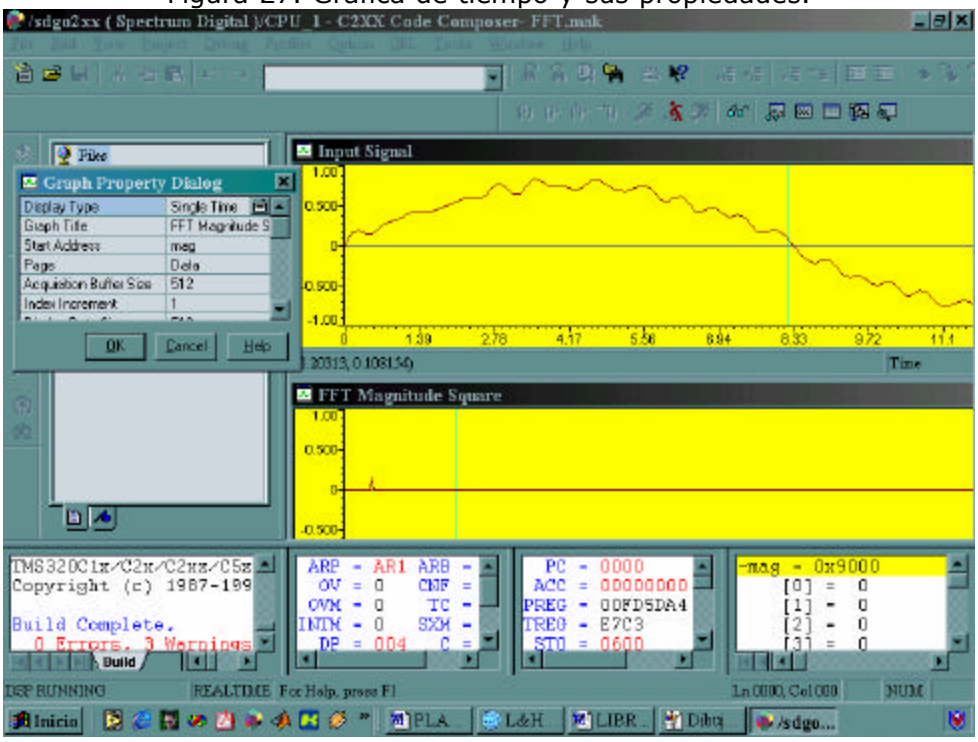
La cuarta y última herramienta es la ventana de visualización gráfica. El *Code Composer* incorpora una interfaz avanzada de análisis de señales, permitiendo la monitorización completa de los datos de las señales. Estas características son útiles en el desarrollo de aplicaciones para comunicación, comunicaciones inalámbricas, procesamiento de imágenes y aplicaciones generales del DSP.

Esta ventana de visualización gráfica tiene cuatro clases de gráficas disponibles: Gráficas tiempo/frecuencia (*Time/Frequency*), diagrama de constelación (*Constellation Diagram*), diagrama de ojo (*Eye Diagram*) e imagen (*Image*). Para la implementación del prototipo se utilizo la gráfica de tiempo/frecuencia, específicamente la gráfica de tiempo. Esta gráfica en función del tiempo funciona de la siguiente manera: existen dos *buffers* asociados a la ventana de visualización. Estos *buffers* son el *buffer* de adquisición y el *buffer* de visualización. El *buffer* de adquisición reside en el DSP, y contiene los datos de interés. Cuando una gráfica es actualizada, el *buffer* de adquisición es leído del DSP y se actualiza el *buffer* de visualización. El *buffer* de visualización reside en la memoria del computador y la gráfica es generada a partir de los datos en el *buffer* de visualización.

Al seleccionar la gráfica tiempo/frecuencia aparece una ventana en la cual se configuran las propiedades de la gráfica. Estas propiedades son fáciles

de entender pero si se requiere información explícita es necesario consultar la referencia [T.I spru328b, 00] o la ayuda del *Code Composer*. La ventana de configuración y la grafica de tiempo se pueden observar en la Figura 27.

Figura 27. Gráfica de tiempo y sus propiedades.



El único interrogante que falta responder en este momento, después de haber explicado el funcionamiento de cada una de las herramientas de programación, es cómo desarrollar un programa para una aplicación. Esta guía se presenta en el anexo C de este libro. En el se describen: todos las pasos a realizar para el desarrollo de una aplicación; los archivos que debe contener el programa; configuración de las opciones para el enlazador, compilador y ensamblador y cómo ejecutarlo.

3 EMULACIÓN EN TIEMPO REAL (*REAL-TIME*)

Aunque esta es una característica del *software* de programación (*Code Composer Studio*), se ha dedicado un capítulo debido a su importancia para la realización de este proyecto. La emulación en tiempo real se refiere a la característica de depuración del *Code Composer* que permite el control de la ejecución del código mientras el procesador continúa atendiendo otras interrupciones.

La importancia de esta herramienta en el desarrollo del proyecto radica en que gracias a ella se pudo ir comprobando y evaluando el funcionamiento del programa implementado para el monitor. Debido a que una de las características principales de la emulación en tiempo real es que le permite al usuario o programador acceder o monitorizar los registros de control y de estado del DSP e ir observando su memoria, se pudo constatar que los valores que se estaban obteniendo de las variables de control y de los parámetros calculados, que eran resultado de las rutinas implementadas para el algoritmo de monitorización de la calidad de energía eléctrica, fueran los esperados. En otras palabras fue una excelente herramienta para la depuración de errores, comprobación y evaluación de la implementación del prototipo.

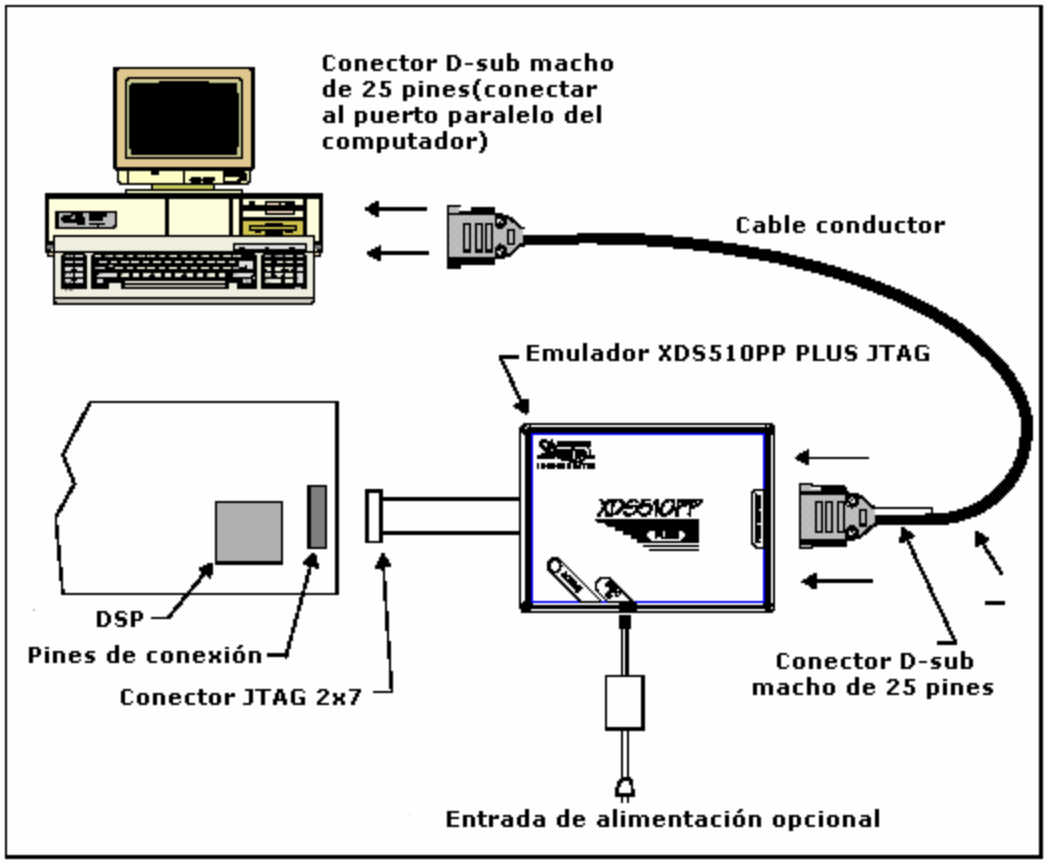
3.1 EMULADOR XDS510PP PLUS

La emulación en tiempo real es posible gracias a el emulador XDS510PP PLUS por puerto paralelo/JTAG. Las características principales de este emulador son: Soporta *DSP's* y microcontroladores de *Texas Instruments* con interfase JTAG (IEEE 1149.1); posee controladores de emulación avanzada suministrando un alto desempeño; soporta el depurador de

lenguaje de alto nivel C de *Texas Instruments.* es compatible con el *Code Composer Studio* de *Texas Instruments.*

En la Figura 28 se muestra el diagrama de conexiones del emulador, la tarjeta de desarrollo y el computador donde se encuentra instalado el *Code Composer Studio*.

Figura 28. Sistema de evaluación.



El emulador permite: El control de la ejecución del código principal (*backgound*) mientras se está atendiendo una interrupción; detectar un *reset* del sistema; acceder la memoria externa e interna mientras la *CPU* está ejecutando el programa y acceder a la *CPU* y registros periféricos mientras la *CPU* está ejecutando el programa.

Los siguientes términos y sus definiciones, son necesarios para entender la emulación en tiempo real:

- El código *background*: Es el código principal que se ejecuta, el cual puede ser detenido durante la depuración.
- El código *foreground*: Es el código de las rutinas de atención de interrupciones (*ISR*), las cuales son ejecutadas cada vez que el código *background* es detenido.
- El estado *Debug-halt*: Es el estado en el cual el dispositivo no ejecuta el código *background*.
- Un evento de depuración: Es una acción. Como ejemplo tenemos la decodificación de una instrucción de *breakpoint* de *software*, la ocurrencia de un análisis de un punto de observación o la demanda del computador para ser atendido, lo que puede terminar en un comportamiento de depuración especial como que el procesador se detenga.
- Un evento de parada: Es cuando un evento de depuración causa que el dispositivo esté en el estado *Debug-halt*.

3.2 MODOS DE CONTROL DE LA EJECUCIÓN

El emulador soporta dos modos de control de la ejecución de depuración: El modo *Stopmode* y el modo *Real-time*.

3.2.1 MODO STOPMODE

El modo *Stopmode* causa eventos de parada los cuales suspenden la ejecución del programa y el puntero de código se detiene en la próxima instrucción. Cuando la ejecución es suspendida, todas las interrupciones, incluyendo la no-enmascarable (*NMI)* y el *reset* (*RS*) son ignoradas hasta que el emulador recibe una directiva para ejecutar de nuevo el código.

En modo *Stopmode* los siguientes estados de ejecución pueden ser utilizados:

- Estado *Debug-halt*. Este estado ocurre por un evento de parada o por una demanda del computador. En este estado la *CPU* está detenida, y por lo tanto, las interrupciones no son atendidas, incluyendo la interrupción no-enmascarable (*NMI*) y el *reset* (*RS*), pero se puede acceder a la *CPU*. Cuando múltiples casos de la misma interrupción ocurren sin que la primera demanda haya sido atendida, las siguientes son ignoradas.
- El estado *Single-Step.* En este estado el emulador ejecuta la instrucción que está seleccionada por el contador de programa (*PC*) y luego regresa al estado *Debug-Halt*. El emulador está solamente en el estado *Single-Step* mientras la instrucción es ejecutada. En este estado no se puede observar la CPU pero se puede atender la demanda de una interrupción.
- El estado de ejecución. En este estado se ejecutan las instrucciones hasta que un comando del depurador o un evento de depuración retorne a la *CPU* al estado *Debug-Halt*. En este estado no se puede observar la *CPU* pero las interrupciones son atendidas.

3.2.2 MODO REAL-TIME

El modo *Real-Time* permite depurar código que interactúa con interrupciones, las cuales deben estar habilitadas. La depuración en *Real-Time* permite que las rutinas de atención de las interrupciones (*ISR*) o código *foreground*, sean llevadas a cabo mientras que la ejecución del código *background* se encuentra detenida. La suspensión de la ejecución del programa puede realizarse en cualquier lugar, lo que permite que se detenga dentro de una interrupción mientras se están atendiendo otras.

En el modo *Real-Time* los siguientes estados de ejecución pueden ser utilizados:

- Estado *Debug-Halt*. Este estado es accesible a través de un evento de parada o por una demanda del computador. En este estado las interrupciones, incluyendo las interrupciones *NMI* y *RS*, todavía son atendidas y se puede acceder a la *CPU*. Ningún otro código es ejecutado y es posible que múltiples interrupciones ocurran y sean atendidas mientras se está en este estado.
- El estado *Single-Step.* En este estado el emulador ejecuta la instrucción que está seleccionada por el contador de programa (*PC*) y luego regresa al estado *Debug-Halt*. El emulador está solamente en el estado *Single-Step* mientras la instrucción es ejecutada. En este estado la *CPU* no puede ser observada y las interrupciones pueden ser atendidas.
- El estado de ejecución. En este estado se ejecutan las instrucciones hasta que un comando del depurador o un evento de depuración retorne a la *CPU* al estado *Debug-Halt*. En este estado se puede observar la *CPU* y todas las interrupciones son atendidas. Cuando una interrupción ocurre simultáneamente con un evento de depuración, el evento de depuración tiene prioridad. Sin embargo, si la ejecución de la interrupción comenzó antes que el evento de depuración ocurriera, el evento de depuración no se puede ejecutar hasta que la rutina de interrupción termine.

El modo *Real-Time* está basado en las vías de análisis (construidas en el *DSP* para emulación o prueba) del *JTAG* y en un pequeño programa *monitor* que es cargado en la memoria o incluido en el código de aplicación. El depurador utiliza estas vías de análisis para enviarle mensajes al programa *monitor*. El *DSP* llama el programa *monitor* y responde a cualquier demanda sin importar que éste no tenga la más alta prioridad de las tareas a ejecutar. Estos mensajes le permiten modificar la memoria o los registros, o ejecutar el código en el *DSP* paso a paso ,lo cual produce que no sean atendidas las interrupciones. El *DSP* interpreta

el *monitor* como un manejador de interrupciones controlado por los mensajes enviados a través de las vías de análisis. El programa *monitor* que se ejecuta en el DSP se llama c200mnrt.out y para ser cargado en el DSP debe ser incluido como un componente en el código de aplicación.

3.2.3 CONFIGURACIÓN DEL MONITOR REAL-TIME

Para hacer uso del monitor se debe hacer lo siguiente:

1. Estudiar las notas detalladas en la sección *Setting Real-Time Configuration Options* de la ayuda del *Code Composer Studio*, la cual se encuentra en el submenú *Help-> Tutorial-> Code Composer Tutorial* y seleccionando la pestaña *Contents* seleccione el subdirectorio *Real-Time Emulation-> Configuring Real-Time Monitor Settings-> Setting Real-Time Configuration Options*. Si los valores que están por defecto no son apropiados para su aplicación, estos se deben asignar a las opciones de ensamblaje condicional en el archivo c200mnrt.i.
2. Leer las notas detalladas en la sección *Setting Real-Time Configuration Options* y colocar los macros: mon_eintr_vecs y mon_etrap_vecs en las localizaciones apropiadas. Estas localizaciones están en el final de la tabla de vectores de interrupción de la memoria de programa.
3. Llamar las rutinas MON_SE_CONFG y MON_RT_CNFG desde las localizaciones apropiadas en el código de reset del programa.
4. Editar el archivo de comando de enlace para coincidir con el archivo *monitor* en lenguaje *assembly*. Un archivo de ejemplo, el c200mnrt.cmd, está localizado en el directorio C:\tic2xx\c2000\MONITOR el cual aparece al instalar el *Code Composer Studio.*

4 MONITORIZACIÓN DE LA CALIDAD DE LA ENERGÍA ELÉCTRICA

La monitorización de la calidad de la energía eléctrica requiere de la adquisición y el procesamiento de las señales de tensión y corriente en diferentes puntos del sistema. Estas señales, que frecuentemente transitan por un ambiente electromagnético adverso deben ser adecuadas para que puedan ser procesadas, debido a sus niveles altos de tensión y de corriente.

Hay muchas razones importantes para la monitorización de la calidad de la energía eléctrica. La razón principal sobre todas las demás es económica, particularmente si las cargas críticas del proceso están siendo fuertemente afectadas por los fenómenos electromagnéticos. Entre los efectos causados por los fenómenos en los equipos y procesos se cuentan mal funcionamiento, daño e interrupción de los procesos.

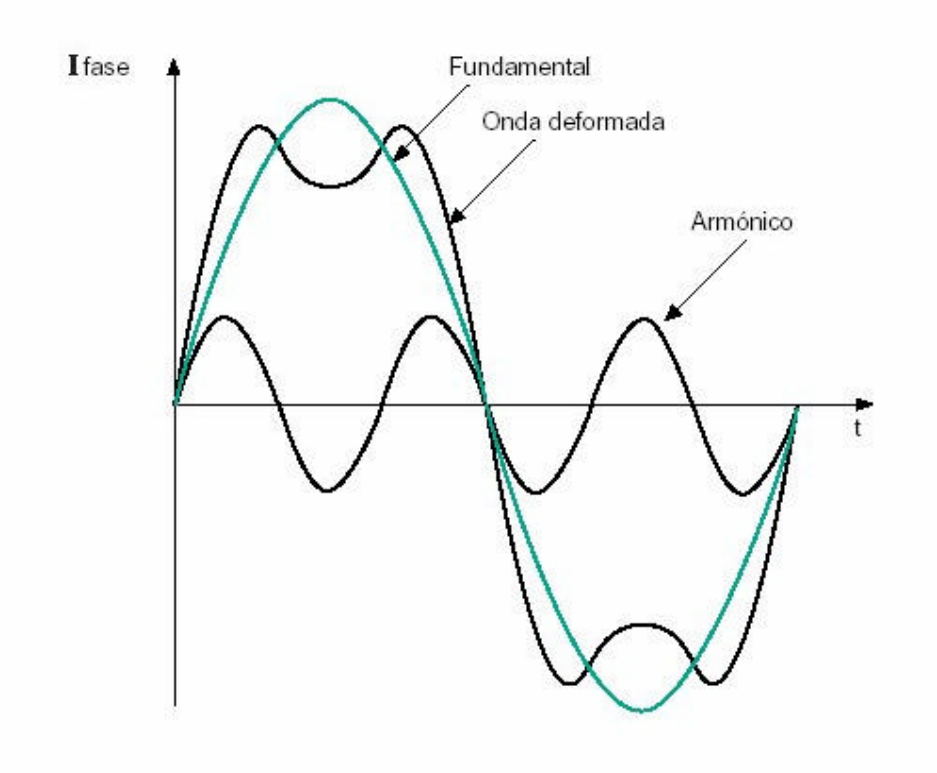
El proceso de monitorización de la calidad de la energía eléctrica comprende detectar, identificar, caracterizar y clasificar los fenómenos electromagnéticos de baja frecuencia que se presentan en los sistemas eléctricos. Para esto, es necesario evaluar algunos parámetros de las señales de tensión y corriente adquiridas del sistema eléctrico y compararlos con los valores umbrales escogidos según los objetivos de la monitorización [IEEE 1159, 95]. Entre los parámetros utilizados se encuentran: El valor pico de la señal, el valor eficaz de la señal, el valor eficaz de la componente fundamental, la frecuencia de la componente fundamental, las componentes de frecuencia predominantes, la distorsión armónica total para la señal de tensión (*THD, Total Harmonic Distortion*),

la distorsión armónica total para la señal de corriente (*TDD, Total Demand Distortion*), etc.

4.1 MONITORIZACION DE EVENTOS EN ESTADO ESTACIONARIO

Los fenómenos de baja frecuencia de interés para la implementación del prototipo son los armónicos e interarmónicos, los cuales se presentan es estado estacionario. De estos se va a estimar su amplitud. Los armónicos e interarmónicos son unos de los principales tipos de distorsión de la señal de tensión o de corriente. Esta distorsión es la desviación de la señal en estado estacionario de una onda sinusoidal ideal, como se puede observar en la Figura 29, caracterizada principalmente por su contenido espectral [IEEE 1159, 95].

Figura 29. Forma de onda: fundamental sinusoidal, componente armónica y distorsión [NTC 5000, 02].



Según [NTC 5000, 02] los armónicos son tensiones o corrientes sinusoidales, las cuales poseen frecuencias que son múltiplos enteros de la frecuencia a la cual el sistema de alimentación de energía eléctrica está diseñado para operar. Los armónicos se combinan con la tensión o corriente fundamental y producen la distorsión de la onda. La distorsión de armónicos existe debido a las características no lineales de los dispositivos y de las cargas del sistema de energía eléctrica [IEEE 1159, 95]. Los niveles de distorsión armónica pueden ser caracterizados por el espectro completo de los armónicos con sus magnitudes y ángulos de fases. Es común utilizar la distorsión armónica total (*THD*) para la señal de tensión o de corriente como una medida de la distorsión armónica [IEEE 1159, 95].

Las ecuaciones (1) y (2) corresponden al factor armónico, el cual está definido como la razón del valor *rss* (root-sum-square) de todos los armónicos entre el valor *rms* (root-mean-square) del fundamental [IEEE 519, 92].

$$Factor\ armónico(señal\ de\ tensión) = \frac{\sqrt{E_3^2 + E_5^2 + E_7^2 + ...}}{E_1} \qquad (1)$$

$$Factorarmónico(señal\ de\ corriente) = \frac{\sqrt{I_3^2 + I_5^2 + I_7^2 + ...}}{I_1} \qquad (2)$$

Otro valor utilizado es el factor de distorsión, el cual es el mismo factor armónico pero expresado en porcentaje. El término distorsión armónica total (*THD*) es usado comúnmente para definir el factor de distorsión.

Los interarmónicos aparecen como frecuencias discretas o como un espectro de banda ancha y pueden estar en cualquier sistema eléctrico. Las principales fuentes de interarmónicos son los convertidores de

frecuencia estática, los ciclo-convertidores, los motores de inducción y los dispositivos de arcos voltaicos [IEEE 1159, 95].

Considerando lo anterior, la estimación de los armónicos de corriente y de tensión es esencial para la calidad de la energía eléctrica. Según [IEEE 519, 92] dentro de los objetivos de la monitorización de armónicos se encuentran:

- Monitorizar niveles existentes de armónicos y verificarlos con los niveles recomendados o admitidos.
- Probar equipos que generan armónicos.
- Diagnosticar y solucionar problemas en situaciones en las cuales el desempeño del equipo es inaceptable para la utilidad o para el usuario.
- Determinar los niveles mínimos de distorsión y realizar un seguimiento de la tendencia en el tiempo de los armónicos de tensión y de corriente (con patrones diarios, mensuales y de estaciones).
- Verificar los resultados de las simulaciones del flujo de armónicos en un sistema eléctrico.
- Medir los armónicos de corriente y de tensión con su respectivo ángulo de fase. Estas mediciones pueden ser efectuadas con o sin una parte de las cargas no lineales conectadas al sistema.

4.2 EQUIPOS DE MONITORIZACIÓN

Los objetivos de la monitorización para un proyecto particular determinan la selección del equipo de monitorización, el método de recolección de datos, los umbrales de monitorización, la técnica de análisis de los datos y el nivel de rigurosidad del proyecto [IEEE 1159, 95]. Otro aspecto importante para la selección y el uso del instrumento de monitorización es que el usuario comprenda las capacidades y limitaciones del instrumento,

es decir, si es capaz de responder a las variaciones del sistema de energía eléctrica.

Los instrumentos utilizados para la monitorización son osciloscopios, monitores de perturbaciones, indicadores de eventos, medidores de tensión y de corriente con almacenamiento (*rvm, recording volt/ammeters*) y los monitores con visualización gráfica.

4.2.1 EQUIPOS UTILIZADOS PARA EL ANALISÍS DE TENSIONES Y CORRIENTES NO SINUSOIDALES

Como la implementación del prototipo de unidad de procesamiento se basa en la monitorización de armónicos, a continuación se describen los equipos básicos para esta monitorización:

- El osciloscopio: La visualización de la onda en el osciloscopio ofrece información cualitativa sobre el grado y tipo de distorsión. En algunos casos, es posible identificar resonancias mediante la distorsión que se observa en las formas de onda de corriente y de tensión [IEEE 519, 92].
- Los analizadores de espectro: Estos instrumentos visualizan la distribución de la potencia de una señal en función de la frecuencia. Un cierto rango de frecuencias es analizado, y todas las componentes, armónicos e interarmónicos, de la señal analizada son visualizadas [IEEE 519, 92].
- Los analizadores armónicos o analizadores de onda: Estos instrumentos miden la amplitud (y en unidades más complejas el ángulo de fase) de una función periódica. Estos instrumentos suministran el espectro de una señal observada. La salida puede ser grabada o puede ser monitorizada con un medidor análogo o digital [IEEE 519, 92].
- Los analizadores de distorsión: Estos instrumentos indican la distorsión armónica total (*THD*) directamente [IEEE 519, 92].

- Equipos digitales de medición de armónicos: Este es el tipo de equipo para el cual se desarrollo el prototipo de unidad de procesamiento de este proyecto. El análisis digital puede ser desarrollado con dos técnicas básicas: Utilizando filtrado digital o con la técnica de la transformada rápida de Fourier (*FFT*). Este es un método muy rápido, y en tiempo real, para desarrollar un análisis espectral que permite la evaluación de diferentes parámetros. La conversión análoga-digital y micro o mini computadores son empleados para la adquisición de los datos en tiempo real [IEEE 519, 92].

Una vez que la forma de la onda ha sido almacenada con un ancho de banda apropiado usando tanto técnicas análogas como digitales *online*, el cálculo de los componentes armónicos con la *FFT*, la conversión a unidades de ingeniería, el cálculo de estadísticas, la visualización e impresión de los resultados puede ser llevado a cabo *off-line* [IEEE 519, 92].

4.2.2 CLASIFICACIÓN GENERAL DE LOS EQUIPOS DE MEDICIÓN DE ARMÓNICOS

Se pueden diferenciar los equipos según tres parámetros: Las características de los armónicos de tensión y corriente a medir, el tipo de precisión del aparato y el tipo de medida a efectuar (tensión, corriente, etc.) [IEC 61000-4-7, 91].

Según [IEC 61000-4-7, 91] se consideran diferentes requisitos que han de cumplir los aparatos según las características de los armónicos a medir:

- Armónicos casi estacionarios (que varían lentamente): Se asume que un análisis continuo (sin intervalo de tiempo entre las medidas sucesivas) no es necesario. Es por ejemplo el caso de las

corrientes armónicas constantes (como aquellas producidas por los receptores de televisión o por los amortiguadores de luz) o el estudio de los armónicos de larga duración en la red de alimentación cuando los efectos instantáneos de los armónicos no se consideren importantes.

- Armónicos fluctuantes: Se requiere un análisis casi continuo en tiempo real. Una aplicación de este tipo es la medida de corrientes armónicas fluctuantes debidas a inversores de motores, cambio de velocidad, etc. También se aplica a los electrodomésticos con regulación electrónica y control de fase. Otro ejemplo sería el control de la tensión de la red en sistemas industriales, como por ejemplo las laminadoras.
- Armónicos que varían rápidamente (o ráfagas muy cortas de armónicos): Para estos, una medida continua en tiempo real es absolutamente necesaria. Un ejemplo de las fuentes de estos armónicos son los receptores de telemando en la red.

Considerando el segundo parámetro, la precisión, según [IEC 61000-4-7, 91] se consideran dos tipos de precisión (A y B) como se muestran en la Tabla 2. Los errores máximos permitidos se refieren a una señal con amplitud fija y de frecuencia determinada en el rango de frecuencias que se muestra en la Tabla 2.

Tabla 2. Requerimientos de exactitud para la medición de la amplitud de las componentes de frecuencia [IEC 61000-4-7, 91].

Clase	Medida	Condiciones	Error máximo permitido
A	Tensión	$U_m \geq 1\% U_N$ $U_m < 1\% U_N$	$5\% U_m$ $0.05\% U_N$
	Corriente	$I_m \geq 3\% I_N$ $I_m < 3\% I_N$	$5\% I_m$ $0.15\% I_N$
B	Tensión	$U_m \geq 3\% U_N$ $U_m < 3\% U_N$	$5\% U_m$ $0.15\% U_N$
	Corriente	$I_m \geq 10\% I_N$ $I_m < 10\% I_N$	$5\% I_m$ $0.5\% I_N$

U_m, I_m son los valores a medir de una componente de frecuencia

U_N, I_N son los valores nominales de entrada

Considerando ahora el tercer y último parámetro se encuentran los tipos de medidas, entre estas están: Los armónicos de corriente y de tensión, el ángulo de fase de los armónicos, la distorsión armónica total, la distorsión armónica ponderada, las componentes simétricas, etc [IEC 61000-4-7, 91].

4.2.3 REQUISITOS DE LOS EQUIPOS DE MEDIDA

Según [IEEE 519, 92] los siguientes requisitos son importantes y por lo tanto deben ser tenidos en cuenta para la medición correcta de armónicos:

- Precisión: El instrumento debe medir una componente armónica constante con un error compatible con el límite permitido. Es razonable usar un instrumento con una incertidumbre que no sea mayor del 5% del límite permitido.
- Selectividad: La selectividad de un instrumento es una indicación de su capacidad para separar componentes armónicos de

diferentes frecuencias. En cuanto a la selectividad, en la Tabla 3 se presentan los requerimientos mínimos de atenuación para una frecuencia inyectada, o frecuencia del armónico, cuando el equipo está configurado para medir la componente de frecuencia de 60 Hz.

Tabla 3. Requisitos mínimos de atenuación (dB).

Frecuencia inyectada (Hz)	Instrumento en el dominio de la frecuencia	Instrumento en el dominio del tiempo
60	0	0
30	50	60
120 a 720	30	50
720 a 1200	20	40
1200 a 2400	15	35

- Valor promedio o instantáneo: si los armónicos medidos varían en tiempo, es necesario suavizar las componentes que oscilan rápidamente en un período de tiempo. En este caso hay 2 factores que deben ser considerados:
    1. Respuesta dinámica: Capacidad para seguir las variaciones en la amplitud de los armónicos.
    2. Ancho de Banda: El ancho de banda de un instrumento afecta considerablemente la medición, especialmente cuando los armónicos están variando. Es recomendado un instrumento con un ancho de banda constante para un conjunto entero de frecuencias. El ancho de banda debe ser 3 ± 0.5Hz para una ganancia de -3bB, además de una atenuación mínima de 40dB para una frecuencia igual a la fundamental más 15Hz. En situaciones en las que los interarmónicos y transitorios están presentes, un ancho de banda más grande causará errores considerables [IEEE 519, 92].

- Presentación de los datos de los armónicos: Los datos medidos pueden ser presentados tanto en formas de tablas como en forma de gráficas.

Según [IEC 61000-4-7, 91] los aparatos que utilizan la transformada rápida de Fourier (FFT) deben tener unos requisitos especiales y unos básicos:

- Modos de operación:
  1. Modo disparo: La transformada de Fourier rápida es realizada a partir de las muestras de una sola ventana que es accionada por un gatillo externo y los resultados son almacenados en forma interna.
  2. Operación continua: El valor de repetición es seleccionado desde por ejemplo $1\times\text{min}$ hasta la operación en tiempo real sin interrupción; se proporciona también un almacenamiento de datos interno de por ejemplo 5000 ventanas.
  3. Operación continua y almacenamiento de los resultados que excedan el umbral: Los resultados son solamente almacenados si uno o más armónicos rebasan los valores límites preseleccionados.
- El equipo deberá suministrar los siguientes tipos de salidas para las amplitudes armónicas c_n, o sus valores eficaces $c_n/\sqrt{2}$, y de manera opcional el ángulo de fase j_n así como la parte real e imaginaria (a_n y b_n respectivamente) con los siguientes medios de salida:
  1. Pantalla numérica en escala lineal o logarítmica de los parámetros seleccionables de los diferentes armónicos, después de un mono disparo o durante la operación continua.

2. Pantalla gráfica del espectro lineal o logarítmico de todas las líneas preseleccionadas de valor de después de un disparo o en la operación continúa.
3. Impresora o ploter
4. Interfaz paralelo y/o interfaz serie para la conexión a un ordenador.

¦ Los requisitos básicos para los equipos que usan la FFT para la monitorización de armónicos de acuerdo con [IEC 61000-4-7, 91] se presentan en la Tabla 4.

Tabla 4. Requisitos básicos para equipos que utilizan la FFT según [IEC 61000-4-7, 91]

Tipo de armónicos	**Anchura de ventana recomendada**	**Requisitos adicionales**
Casi estacionarias	T_W = 0.1s-0.5 s	Puede haber intervalos entre ventanas
Fluctuantes (Según CEI 55-2)	T_W = 0.32 s (rectangular)	Sin intervalo
	T_W = 0.4s-0.5 s (Hanning)	Solapamiento mitad a mitad
Rápidamente variables	T_W = 0.08s-0.16s (rectangular)	Sin intervalo

En [IEC 61000-4-7, 91] se establecen también requisitos para los circuitos de entrada para las señales de tensión y de corriente así como para los transformadores de tensión y de corriente. En este proyecto no se consideran estos requisitos, ya que la implementación del prototipo de unidad de procesamiento no incluye la etapa de adecuación y adquisición de la señales de tensión y corriente.

El prototipo implementado puede ser usado para monitorización de armónicos casi estacionarios, con capacidad para ser mejorado para la monitorización de armónicos fluctuantes y armónicos rápidamente

variables. Basándose en los resultados obtenidos, puede estar clasificado según su precisión como clase B y con una incertidumbre menor al 5% para armónicos con amplitudes mayores al 9.09% de el valor nominal. Los resultados se presentan en forma gráfica y numérica. A esta visualización, se le puede agregar más adelante en nuevos desarrollos una opción para el almacenamiento de datos según el patrón a monitorizar, permitiendo la recolección de datos para estadísticas.

5 LA TRANSFORMADA DISCRETA DE FOURIER (*DFT*)

La transformada discreta de Fourier (*DFT*) es una técnica matemática basada en la descomposición de las señales en sinusoidales, y se aplica a las señales digitalizadas y de duración finita. La *DFT* es una secuencia que corresponde a muestras equiespaciadas en frecuencia de la transformada de Fourier de la señal y tiene una gran importancia en la realización de una gran variedad de algoritmos de tratamiento digital de señales [Oppenheim & Schafer, 00]. En este capítulo se explicara cómo funciona esta técnica así como el algoritmo utilizado para su implementación.

5.1 ¿CÓMO FUNCIONA LA DFT?

La utilización de esta técnica para la implementación del prototipo se debe principalmente a dos razones. La primera es que el algoritmo de la *DFT* se utiliza para estimar componentes de frecuencia de la señal a partir de sus muestras; permitiendo estimar así eventos en estado estacionario como son los armónicos, interarmónicos, subarmónicos, valor eficaz e incluso componentes de secuencia en sistemas trifásicos. La segunda razón es que las recomendaciones de la norma [IEC 61000-4-7, 91] y del estándar [IEEE 519, 92] contemplan a la transformada discreta de Fourier (*DFT* en tiempo real) como el algoritmo para la monitorización de armónicos.

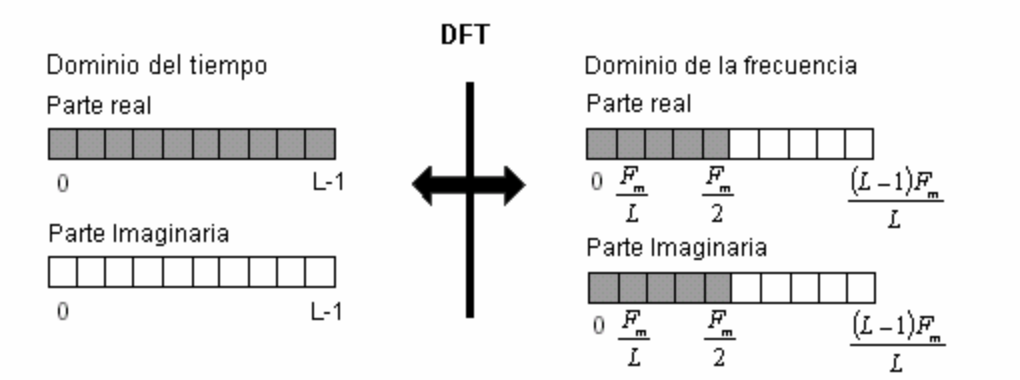
El fundamento para utilizar la transformada discreta de Fourier consiste en que: una cantidad *L* de muestras de la señal se puede representar o descomponer como la suma de *L* componentes de frecuencia

(exponenciales complejas de la forma $Be^{j(w\,n+b)}$). El algoritmo *DFT* calcula la magnitud y el ángulo de fase de tales componentes mediante la siguiente ecuación:

$$F[k]=\sum_{n=0}^{L-1}x[n]e^{-jk\frac{2p}{L}n} \quad \text{para } L \text{ valores enteros y consecutivos de } k \qquad (3)$$

F[k] es un valor complejo que contiene la información de magnitud y fase para la componente con frecuencia $k*F_{muestreo}/L$ [Hz] (ó $K2p/L$ [rad/muestra])*. De esta forma, la frecuencia de las componentes calculadas corresponde a los múltiplos enteros de $F_{muestreo}/L$ [Hz] que se encuentran ubicados en el rango $-0.5F_{muestreo}$ [Hz] a $0.5F_{muestreo}$ [Hz]. Por consiguiente, la máxima frecuencia que puede estudiarse es $0.5F_{muestreo}$ [Hz], como se ve en la Figura 30; y para una frecuencia de muestreo determinada, cuanto mayor sea la cantidad de muestras se calculará un mayor número de componentes de frecuencia con frecuencias más cercanas.

Figura 30. Transformada discreta de Fourier compleja.



* La frecuencia de una señal discreta puede interpretarse como una normalización de la frecuencia de la señal continua y es igual a la frecuencia de la señal continua dividida por la frecuencia de muestreo.

La reconstrucción de las muestras de la señal se realiza mediante la siguiente ecuación de síntesis:

$$x[n]=\frac{1}{L}\sum_{k=0}^{L-1}F[k]e^{jk\frac{2p}{L}n} \qquad para\ \ 0\le n\le L-1 \tag{4}$$

La señal *x[n]* recuperada a partir de la Ecuación (4) es periódica con periodo *L* muestras; y las muestras para $0 \le n \le L-1$ corresponden exactamente a las *L* muestras tomadas para calcular la información de las componentes de frecuencia mediante la Ecuación (3). En consecuencia, si la señal es periódica y las *L* muestras utilizadas en el algoritmo DFT corresponden a un periodo fundamental de la señal, las componentes de frecuencia estimadas serán los armónicos de la señal. Si se desean calcular subarmónicos o interarmónicos será necesario tomar más de un periodo fundamental de la señal para aumentar la resolución en frecuencia.

Cuando se toma un periodo fundamental de la señal, las amplitudes o valores pico de los armónicos y sus respectivos ángulos de fase corresponden a:

$$A_k=\frac{2}{L}F[k], q_k=\angle F[k] \quad para 1\le k\le L/2 \tag{5}$$

Como las muestras son de valor real, entonces la componente de continua (*K=0*) y la amplitud del armónico de orden *L/2* se calculan así:

$$A_0=\frac{F[0]}{L}, \quad A_{L/2}=\frac{F[L/2]}{L} \tag{6}$$

Por consiguiente, el algoritmo *DFT* puede interpretarse como un banco de filtros, uno por cada armónico [Bellanger, 94].

5.2 LA TRANSFORMADA RÁPIDA DE FOURIER (*FFT*)

Existen varias formas de calcular la transformada discreta de Fourier (*DFT, discrete Fourier transform*), entre las cuales están la solución de ecuación lineales simultaneas o el método de correlación. La transformada rápida de Fourier (*FFT, fast Fourier transform*) es otro método para calcular la DFT, el cual es considerablemente más eficiente, reduciendo el tiempo de calculo en cientos [Smith, 99]. Considerando esto, el algoritmo FFT es el que normalmente se implementa para el cálculo de la DFT, ya que con este algoritmo se realizan $(L/2)*\log_2(L)$ en lugar de L^2 multiplicaciones (si se realiza el cálculo directo de la *DFT*) [Proakis & Manolakis, 98].

5.2.1 ¿CÓMO FUNCIONA LA FFT?

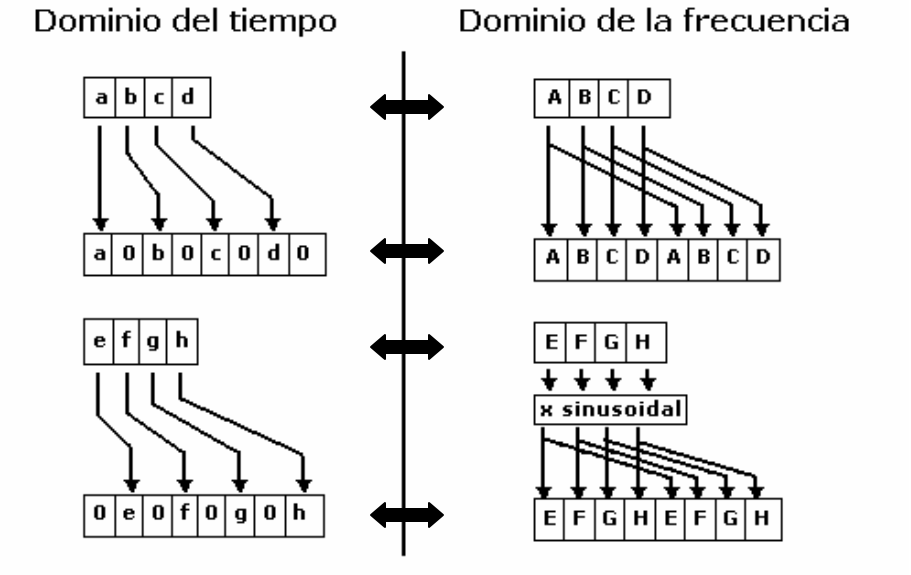
Según [Proakis & Manolakis, 98] la familia de algoritmos *FFT* utiliza la estrategia "divide y vencerás". Esta metodología es usada de la siguiente forma: Los algoritmos de *FFT* se basan en el principio fundamental de descomponer el cálculo de una transformada discreta de Fourier de *L puntos* en transformadas discretas de Fourier sucesivas con menor número de puntos [Oppenheim & Schafer, 00].

La *FFT* opera descomponiendo una señal de *L* puntos en el dominio del tiempo en *L* señales cada una compuesta de un solo dato. El segundo paso es calcular los *L* espectros de frecuencia correspondientes a estas *L* señales. Para este paso no hay que realizar nada, debido a que el espectro de frecuencia de una señal de un dato es igual al mismo dato. Por último, los *L* espectros son sintetizados en un solo espectro de frecuencia [Smith, 99].

Para ilustrar como se hace la síntesis de los L espectros se presentará un ejemplo de cómo dos espectros en frecuencia, cada uno compuesto de 4

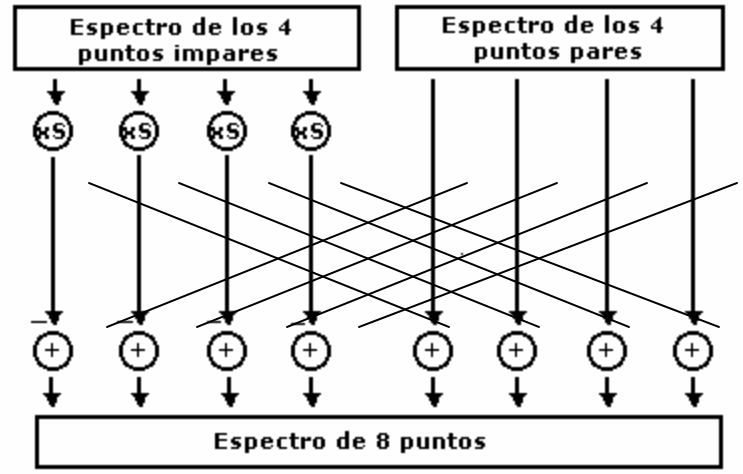
puntos, son combinados en un solo espectro de 8 puntos (ver Figura 31). La síntesis debe deshacer la descomposición realizada en el dominio del tiempo. En otras palabras, las operaciones en el dominio de la frecuencia deben corresponder al procedimiento realizado en el dominio del tiempo para la combinación de dos señales de 4 puntos [Smith, 99].

Figura 31. Síntesis realizada por la FFT.



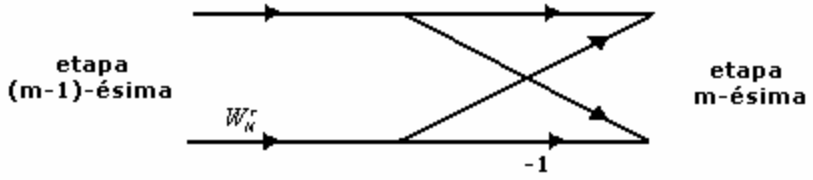
Cuando una señal en el dominio del tiempo es expandida con ceros, o expandida por un factor de ½, el espectro en frecuencia es comprimido o duplicado. Si la señal en el tiempo es también desplazada por una muestra durante la expansión, el espectro es multiplicado adicionalmente por una exponencial compleja o por un par de funciones senoidales en cuadratura.

Figura 32. Diagrama de flujo de la síntesis.



En la Figura 32 se muestra cómo se combinan dos señales de 4 puntos en el dominio de la frecuencia en un espectro de frecuencia de 8 puntos. En este punto aparece la *mariposa* como la operación básica para el cálculo de la FFT; este nombre es dado debido a la forma del grafo de flujo de la Figura 33 [Oppenheim & Schafer, 00].

Figura 33. Grafo del flujo del cálculo en mariposa.



5.2.2 ALGORITMO FFT IMPLEMENTADO

Para el prototipo se realizó el algoritmo *FFT* base-2 (*radix*-2), el cual requiere que el número de muestras (*L)* sea potencia entera de 2 ($L = 2^V$ con *v* entero). Este algoritmo se implementó utilizando la estrategia "divide y vencerás" y los resultados de la DFT se almacenan en la misma posición de memoria en donde se encuentra la información de la señal (*in-placed computation o in situ*).

Se realizó diezmado en tiempo para descomponer la señal de *L* muestras en el dominio del tiempo en *L* señales de una sola muestra. En la implementación del algoritmo se utiliza una descomposición entrelazada cada vez que la señal es partida en dos, como se ve en la Figura 34; lo que significa que la señal es separada en sus muestras con numeración par e impar [Smith, 99]. La descomposición no es más que un reordenamiento de las muestras en la señal. La Figura 35 muestra el patrón de reordenamiento requerido, el cual es implementado con un algoritmo de ordenamiento de inversión de bits con el fin de realizar el cómputo en el mismo lugar.

Figura 34. Diezmado de una señal de 8 muestras.

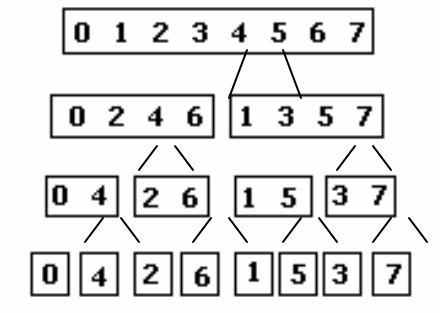


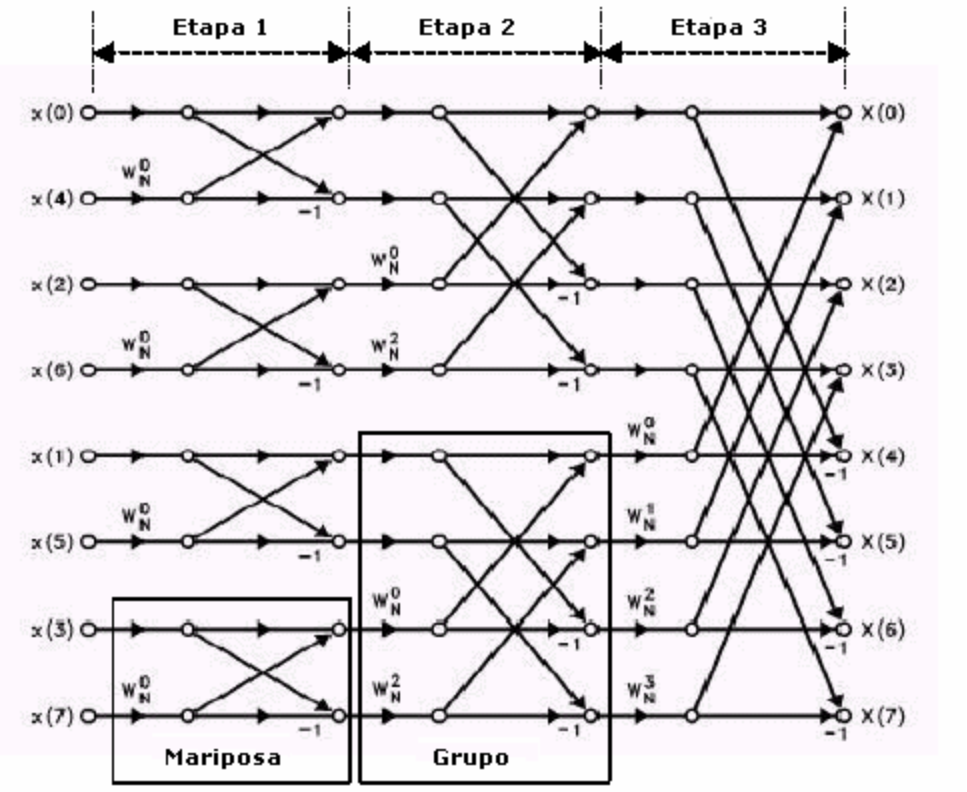
Figura 35. Ordenamiento de inversión de bits.

Números de las muestras en orden normal		Números de las muestras después de la inversión de bits	
Decimal	Binario	Decimal	Binario
0	000	0	000
1	001	4	100
2	010	2	010
3	011	6	110
4	100	1	001
5	101	5	101
6	110	3	011
7	111	7	111

5.2.3 CONSIDERACIONES PARA LA IMPLEMENTACIÓN DEL ALGORITMO

Para determinar estas consideraciones, en primera instancia consideres el grafo de la FFT de una señal de 8 muestras, ver Figura 36, el cual se basa en la operación de la mariposa de la Figura 33.

Figura 36. FFT de 8 puntos.



Este grafo para la FFT de 8 puntos proporciona el diagrama de flujo para la implementación del algoritmo. Hay tres lazos anidados: El lazo de las etapas, el lazo de los grupos y el lazo de las mariposas. Como se ve en la figura 30, hay $v = \log_2(L)$ etapas (el lazo externo), cada etapa *m* tiene 2^{v-m} grupos y cada grupo tiene 2^{m-1} mariposas.

Considerando este grafo y tomando como base el programa presentado en [Smith, 99] para la FFT, el paso siguiente es determinar el número de veces que se repite cada lazo. Para el lazo exterior, que es el de las etapas, el número de veces que esté se repite es el número de etapas. Por lo tanto el lazo exterior se realizara desde 1 hasta el número total de etapas $v = \log_2(L)$ con incrementos de uno.

Para el lazo intermedio, el de los grupos, es necesario identificar algunas características de cada grupo. Inicialmente, es necesario determinar el número de puntos de la *DFT* que se calcula en cada grupo. Observando la Figura 36 en la etapa 1 hay 4 *DFTs* de 2 puntos, en la etapa 2 hay 2 *DFTs* de 4 puntos y en la etapa 3 hay una *DFT* de 8 puntos. Por lo tanto, el número de puntos de las *DFTs* de la etapa *m* es igual 2^m. Como el cálculo de la *DFT* se basa en el cálculo de la mariposa que se muestra en la Figura 33, la cual solo requiere de dos puntos, es necesario determinar el número de mariposas en cada grupo para calcular la *DFT* requerida. Es decir, retomando la Figura 36 se observa que:

- Para la etapa 1, hay 4 grupos de *DFTs* de 2 puntos; por lo tanto hay que realizar una mariposa por grupo, es decir, se pasa solo una vez por grupo.
- Para la etapa 2, hay 2 grupos de *DFTs* de 4 puntos por lo que se debe pasar 2 veces por cada grupo, es decir, realizar dos mariposas para calcular la *DFT* de cada grupo.
- Para la etapa 3, hay 1 grupo de *DFT* de 8 puntos, por lo tanto hay que pasar por el grupo 4 veces para calcular las 4 mariposas de la *DFT*.

De todo lo dicho anteriormente se infiere que para cada etapa *m* hay 2^{v-m} grupos, en los cuales se llevan a cabo *DFTs* de 2^m puntos para lo cual se tiene que pasar por cada grupo, el número de puntos de cada DFT

de cada uno de los grupos sobre 2. Por lo tanto el rango para el contador del segundo lazo va desde 1 hasta $2^m/2 = 2^{m-1}$ también con incrementos de uno igual que el primer lazo.

Para el tercer y último lazo, que es el interno, se deben determinar tanto el rango del contador como su paso de incremento para llevar a cabo cada una de las mariposas de los grupos. Aquí es importante tener en cuenta un aspecto: Las mariposas de cada grupo se van calculando intercaladamente; esto quiere decir que se calcula la primera mariposa para cada uno de los grupos, luego se calcula la segunda para cada grupo, después la tercera y así sucesivamente. Por lo tanto, en este lazo no es un problema hallar el límite superior del contador ya que éste va a ser el número total de muestras (*L*). Ahora es necesario hallar el límite inicial y el paso de incremento para ir por cada una de las mariposas del mismo orden de cada grupo. De nuevo, si se observa la Figura 36 y analizando cada una de las etapas se induce que:

- En la primera etapa se recorre una sola vez cada grupo, ya que solo hay una mariposa por grupo, y se calculan las mariposas cada dos muestras comenzando desde la muestra cero.
- Para la etapa 2 se necesita recorrer cada grupo 2 veces. En la primera vez se comienza también desde la muestra cero, pero las mariposas se van calculando cada 4 muestras. La segunda vez que se recorren los grupos se comienza desde la muestra uno y también se van calculando las mariposas cada 4 muestras.
- Para la etapa 3, en la cual solo hay un grupo, se tiene que recorrer 4 veces el grupo. Para la primera vez se empieza en la muestra cero, para la segunda se comienza en la muestra uno, para la tercera vez en la muestra 2 y para la cuarta vez se comienza con la muestra 3.

De lo e anterior se puede deducir que para cada *j*-ésimo recorrido se debe comenzar el tercer lazo en *j*-1 con pasos de incremento de 2^m.

Para la implementación del algoritmo FFT hace falta hacer una consideración más para el límite inferior y para el incremento del tercer lazo. El grafo de la Figura 36 asume que los puntos a los que se les calcula cada una de las mariposas es un número complejo. En el DSP las muestras de la señal se almacenan en un vector en el cual se encuentran separadas la parte real y la parte imaginaria, y estas se encuentran en espacios contiguos de memoria. Es decir, el punto cero (*x(0)*) del grafo de la Figura 36 es en el DSP la posición cero (parte real) y la posición uno (parte imaginaria) del vector de muestras. Por lo tanto cada punto del grafo de la Figura 36 aparece como dos puntos (muestras) en la implementación. Realizando de nuevo el análisis del grafo teniendo en cuenta esta consideración se tiene que:

- Para la etapa 1 es necesario recorrer una sola vez los grupos, por lo tanto se empieza en el punto cero y se va avanzando cada 4 puntos.
- Para la etapa 2 se recorre 2 veces cada grupo. Para el primer recorrido se empieza en el punto cero y se va avanzando cada 8 puntos. En el segundo recorrido se empieza en el punto 2 y al igual que en el anterior se avanza cada 8 puntos.
- Para la etapa 3 se recorre 4 veces cada grupo. En el primer recorrido se empieza desde el punto cero, en el segundo recorrido se comienza en el punto 2, en el tercer recorrido se comienza en el punto 4 y en el cuarto recorrido se empieza desde el punto 6. Para todos los recorridos se van dando pasos de 16 puntos.

Lo anterior permite establecer los verdaderos valores de el límite inferior y del paso de incremento para el tercer lazo, lo cual lleva a la conclusión

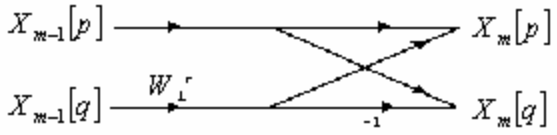
que para el *j*-ésima recorrido de la etapa *m*, el límite inferior de el tercer lazo es 2*(*j*-1) y los pasos son 2^{m+1}.

Una vez definidos los ciclos anidados, los cuales son los encargados de hacer el recorrido por cada uno de los puntos de cada etapa y así calcular las mariposas de cada grupo, lo único que resta es determinar las ecuaciones de este cálculo básico para la DFT. Utilizando una nueva forma de la mariposa de la Figura 33 ahora con unas etiquetas a la entrada y a la salida de la mariposa, como se observa en la Figura 37, se obtienen las ecuaciones (7) y (8) para el cálculo de la mariposa [Oppenheim & Schafer, 00].

$$X_m[p] = X_{m-1}[p] + W_N^r X_{m-1}[q] \tag{7}$$

$$X_m[q] = X_{m-1}[p] - W_N^r X_{m-1}[q] \tag{8}$$

Figura 37. Mariposa con entradas y salidas etiquetadas.



Con base en las Eecuaciones (7) y (8) y dado que estas ecuaciones asumen que tanto los puntos de entrada como los de salida son números complejos, se plantean las ecuaciones para la implementación de las mariposas en este proyecto. Considerando en primer lugar el factor W_L^r, el cual es un factor de ganancia (*twiddle factor*) encargado de realizar en el dominio de la frecuencia la operación del desplazamiento en tiempo y aplicando la identidad de Euler se tiene que

$W_L^r = Cos(2p\,r/L) - jSen(2p\,r/L)$. Con este factor de ganancia y recordando que en la implementación del algoritmo la parte real e imaginaria de cada muestra (punto) están separadas, las ecuaciones (7) y (8) quedan redefinidas de la siguiente forma:

$$X_{mreal}[p] = X_{mreal-1}[p] + Cos(2p\,r/L) * X_{mreal-1}[q] + Sen(2pr/L) * X_{mimag-1}[q] \quad (9)$$

$$X_{mimag}[p] = X_{mimag-1}[p] + Cos(2p\,r/L) * X_{mimag-1}[q] - Sen(2pr/L) * X_{mreal-1}[q] \quad (10)$$

$$X_{mreal}[q] = X_{mreal-1}[p] - Cos(2p\,r/L) * X_{mreal-1}[q] - Sen(2pr/L) * X_{mimag-1}[q] \quad (11)$$

$$X_{mimag}[q] = X_{mimag-1}[p] - Cos(2p\,r/L) * X_{mimag-1}[q] + Sen(2pr/L) * X_{mreal-1}[q] \quad (12)$$

En las ecuaciones (9), (10), (11) y (12) se observa que es necesario definir los parámetros *p, q* y *r* para la implementación del algoritmo.

El parámetro *p* indica el primer punto con el cual se calcula la mariposa en cada etapa; esto es llevado a cabo por el recorrido que realiza el tercer lazo. Como ya se determino para el tercer lazo, el *j*-ésimo recorrido de la etapa *m* va desde 2(*j*-1) hasta 2**L*, con pasos de 2^{m+1}. Al denominar *k* al contador que determina los límites del tercer lazo (*k*= 2*(*j*-1) hasta 2**L*, con un paso de 2^{m+1}), se tiene que *p*=*k* para la parte real y *p*=*k*+1 para la parte imaginaria.

Observando de nuevo la figura 36, para el parámetro *q* se tiene que:

- Para la etapa 1, *q* toma el valor *p*+1.
- Para la etapa 2, *q* toma el valor *p*+2.
- Para la etapa 3, *q* toma el valor *p*+4.

Con base en este patrón y teniendo en cuenta que se almacena la parte real e imaginaria de cada muestra en espacios de memoria contiguos se tiene que:

- Para la etapa 1, *q* toma el valor *p*+2.
- Para la etapa 2, *q* toma el valor *p*+4.
- Para la etapa 3, *q* toma el valor *p*+8.

En consecuencia para la etapa *m*, *q* va a ser igual $p+2^m$. Al igual que *p*, *q* tiene dos valores uno para la parte real del punto y otro para la parte imaginaria. Por lo tanto $q=k+2^m$ para la parte real y $q=k+1+2^m$ para la parte imaginaria.

A partir de la Figura 36, se puede establecer las siguientes observaciones con respecto al parámetro *r*:

- Para la etapa 1, se realiza un recorrido por cada grupo, cada grupo consiste de una mariposa, y *r* toma el valor de 0.
- Para la etapa 2, se realizan 2 recorridos por cada grupo, ya que es una *DFT* de 4 puntos, y *r* toma el valor de 0 en el primero y el valor de 2 en el segundo.
- Para la etapa 3, se realizan 4 recorridos por cada grupo, ya que es una *DFT* de 8 puntos, y *r* toma el valor de 0 en el primer recorrido, 1 en el segundo, 2 en el tercero y 3 en el cuarto.

En este caso ha sido necesario tener en cuenta las consideraciones prácticas sugeridas por [Oppenheim & Schafer, 00] en donde se establece que los factores de ganancia de cada una de las mariposas en cada etapa

m son potencias enteras de $W_N^{(N/2^m)}$. Teniendo en cuenta esta consideración se tiene que:

- Para la etapa 1, *r* es múltiplo de L/2=4.
- Para la etapa 2, r es múltiplo de L/4=2.
- Para la etapa 3, r es múltiplo de L/8=1.

Ahora, falta hallar cómo va variando el factor de multiplicación para *r*:

- Para la primera etapa, en donde solo hay un recorrido, el contador tiene que producir un cero para el valor de *r*.
- Para la segunda etapa, el factor de multiplicación tiene que producir un cero y un dos. Por lo tanto para el primer recorrido tiene que ser cero y para el segundo recorrido tiene que ser uno ya que *r* va a ser múltiplo entero de 2.
- Para la etapa 3, el factor de multiplicación de *r* tienen que producir un cero en el primer recorrido, un 1 en el segundo recorrido, un 2 en el tercer recorrido y un 3 en el cuarto recorrido. Por lo tanto los valores que producen estos resultados son 0, 1, 2 y 3, debido a que *r* debe ser múltiplo entero de 1.

En lo anterior el factor de multiplicación en cada *j*-ésimo recorrido es *j*-1. De esta forma queda completamente determinado el valor de *r* en la etapa *m* y en cada *j*-ésimo recorrido. Este valor es $r = (j\text{-}1)*(L/2^m)$.

Tomando los resultados para *p, q* y *r* las ecuaciones (9), (10), (11) y (12) quedan de la forma:

$$X_m[k] = X_{m-1}[k] + Cos\left((2p/N)*(j-1)*\left(L/2^m\right)\right)X_{m-1}\left[k+2^m\right] + Sen\left((2p/L)*(j-1)*\left(L/2^m\right)\right)X_{m-1}\left[k+1+2^m\right] \quad (13)$$

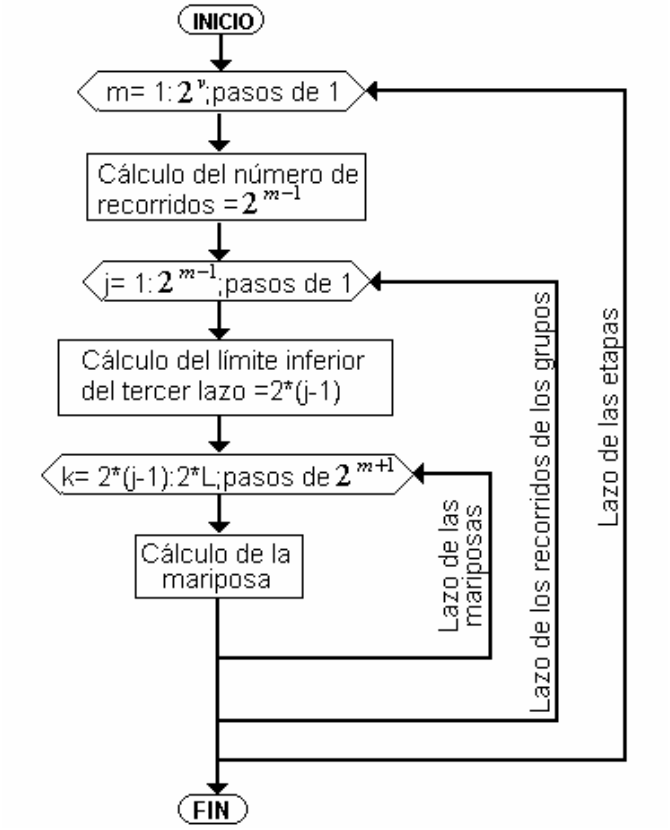
$$X_m[k+1] = X_{m-1}[k+1] + Cos\left((2p/N)*(j-1)*\left(L/2^m\right)\right)X_{m-1}\left[k+1+2^m\right] - Sen\left((2p/L)*(j-1)*\left(L/2^m\right)\right)X_{m-1}\left[k+2^m\right] \quad (14)$$

$$X_m\left[k+2^m\right] = X_{m-1}[k] - Cos\left((2p/N)*(j-1)*\left(L/2^m\right)\right)X_{m-1}\left[k+2^m\right] - Sen\left((2p/L)*(j-1)*\left(L/2^m\right)\right)X_{m-1}\left[k+1+2^m\right] \quad (15)$$

$$X_m\left[k+1+2^m\right] = X_{m-1}[k+1] - Cos\left((2p/N)*(j-1)*\left(L/2^m\right)\right)X_{m-1}\left[k+1+2^m\right] + Sen\left((2p/L)*(j-1)*\left(L/2^m\right)\right)X_{m-1}\left[k+2^m\right] \quad (16)$$

Estas últimas cuatro ecuaciones son las que finalmente fueron utilizadas para los cálculos de la *DFT* en la implementación del algoritmo *FFT* en el *DSP*. Hay que anotar que se utilizaron unas sentencias de asignación intermedias en los cálculos de las *DFT* debido a que se realizó un cómputo en el mismo lugar; y por lo tanto, el vector de almacenamiento de las muestras de entrada es el mismo para almacenar los resultados de salida. En la Figura 38 se muestra el diagrama de flujo del algoritmo implementado para el cálculo de la FFT en el prototipo. El lenguaje de programación de este algoritmo fue lenguaje C y algunas rutinas de inicio de tablas de datos en lenguaje *assembly*.

Figura 38. Diagrama de flujo del algoritmo FFT implementado.



5.2.4 ERRORES EN LA IMPLEMENTACIÓN

Durante la implementación del algoritmo FFT surgieron varios errores los cuales llevaban a un cálculo erróneo del espectro de la señal a analizar. De todos éstos cuatro fueron los más importantes. Dos de ellos eran externos a la implementación y los otros dos si eran producidos durante los cálculos de las mariposas de la DFT.

Inicialmente se tratarán los errores externos y se describirá cómo se corrigieron durante la implementación. Estos dos errores fueron:

- El primero surgía cuando se hallaba el número total de etapas (v) que se tenían que realizar, utilizado para el límite superior del lazo externo.
- El segundo aparecía cuando se calculaba el número de puntos de cada *DFT* en las respectivas etapas, cuyo valor era utilizado para

determinar el numero de recorridos para cada grupo en su respectiva etapa (límite superior del lazo intermedio), para el paso de incremento del lazo interno y para hallar la posición del segundo punto usado en el cálculo de la mariposa.

Para encontrar estos errores se utilizaron tres de las herramientas que están disponibles en el *software* de programación del *DSP* (*Code Composer Studio*), los *breakpoints*, la emulación en tiempo real y la ejecución del programa por pasos (*StepInto, StepOver y StepOut*). Estos errores eran producidos por las funciones intrínsecas de C, *double log10* (*double x*) y *double pow* (*double x, double y*), que se encuentran en la librería de soporte de *run-time* rts2xx.lib del *DSP*. La primera función (l*og10*), la cual calcula el logaritmo en base 10 de un número *x* y fue usada para hallar el número de etapas $v = \log_2 L$, presentaba error en el cálculo de *v* cuando *L*=128, obteniéndose un resultado de *v*=6. Para los demás valores de L (L= 256, 512, 1024 ó 2048) que el usuario puede seleccionar no presentaba ningún error. Para solucionarlo se adicionó un sentencia de decisión en la cual se pregunta si L=128, y si corresponde, se le suma 1 al valor de *v* obtenido con la función de log10(*L*)/log10(2). La segunda función (*pow*), la cual calcula la potencia x^y, era la única encargada de cometer el segundo error debido a que se usaba para calcular las potencias de 2. El resultado de estas potencias era el que se utilizaba para hallar el número de recorridos por cada etapa, el número de puntos de cada DFT y el paso de incremento del lazo interno. Para solucionarlo se uso un acumulador para multiplicaciones donde se va almacenando la multiplicación del número 2 consigo mismo durante cada ciclo del lazo externo.

Los dos errores restantes son implícitos a la implementación del algoritmo. Ambos errores tienen que ver con el desbordamiento de las operaciones en el DSP y al igual que los dos errores anteriores, la

detección y solución de ellos se realizó gracias a las herramientas de depuración del *Code Composer Studio*. El primero de ellos aparece cuando se llevan a cabo las multiplicaciones de los puntos para el cálculo de la mariposa. Lo que ocurre es lo siguiente: El vector que almacena los resultados del conversor análogo-digital es un vector definido como entero. Esta definición esta hecha debido a que se utilizaron unas librerías suministradas por *Texas Instruments* como base para la implementación, para poder llevar a cabo la adquisición de la señal para el cálculo de la *DFT*. Como el *DSP* es de punto fijo, en las librerías se usa notación en formato Q15 para trabajar los números, ofreciendo un rango desde -32768 para representar -1 hasta 32767 para representar 0.99999. Por lo tanto si se multiplican dos números en este formato el resultado va a sobrepasar el rango de una variable que fue declarada entera, cuyo rango está desde -32768 a 32768, produciendo un desbordamiento y por ende errores en el cálculo. Para corregir este error se convirtieron todos los números a decimales, dividiendo por el factor 32767, para luego si llevar a cabo las respectivas multiplicaciones y adiciones. Es aquí donde aparece el segundo error, cuya causa son las sumas realizadas en las mariposas que también pueden producir desbordamiento. Esto se debe a la siguiente razón: Si se tienen dos números que estén en el rango -1 a 0.9999 (-32768 a 32768 en Q15), existe la posibilidad de que su suma o resta sobrepase este rango, llegando a un máximo de 2*(-32768) o de 2*(32767). Por esta razón es que el resultado total de los cálculos de cada punto se multiplica por un factor igual a ½, manteniéndolo dentro del rango del formato Q15. Según [Proakis & Manolakis, 98], este es el escalado que se debe realizar ya que no afecta al nivel de la señal a la salida del algoritmo para la FFT y reduce significativamente la varianza de los errores de cuantificación a la salida. Al final de las *v* etapas se habrá conseguido un factor de escala global $(1/2)^v = 1/L$. Así, pues, se evita el desbordamiento en el cálculo de la DFT [Proakis & Manolakis, 98].

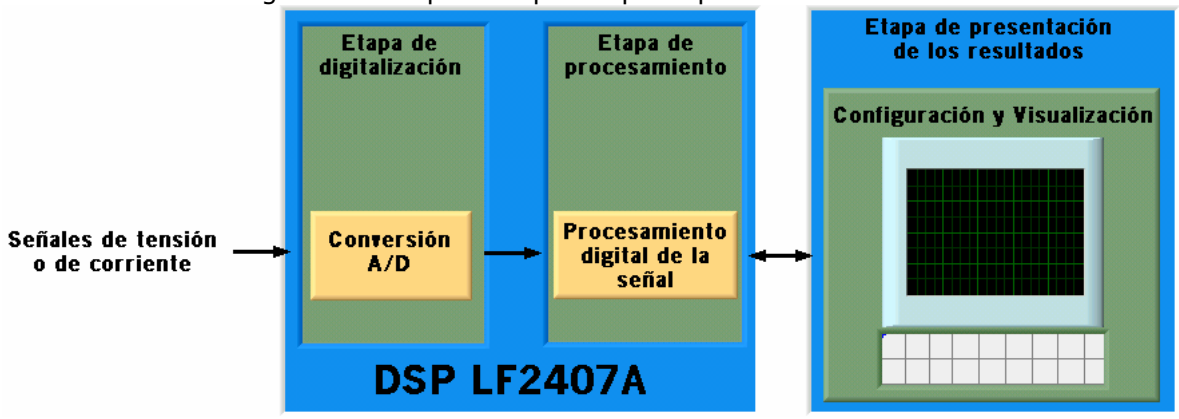
6 IMPLEMENTACIÓN DEL PROTOTIPO DE UNIDAD DE PROCESAMIENTO

En este capítulo se presentan las pautas principales para la implementación en tiempo real del algoritmo FFT. Se describen las librerías de *Texas Instruments* utilizadas como plantillas para desarrollar el algoritmo y las modificaciones que se les hicieron para cumplir con los objetivos propuestos para el prototipo. Este es un capítulo basado fundamentalmente en la experiencia adquirida durante la programación, de tal forma que se exponen los inconvenientes encontrados en el desarrollo de la implementación, así como los pasos realizados para solucionarlos.

6.1 DESCRIPCIÓN DEL PROTOTIPO IMPLEMENTADO

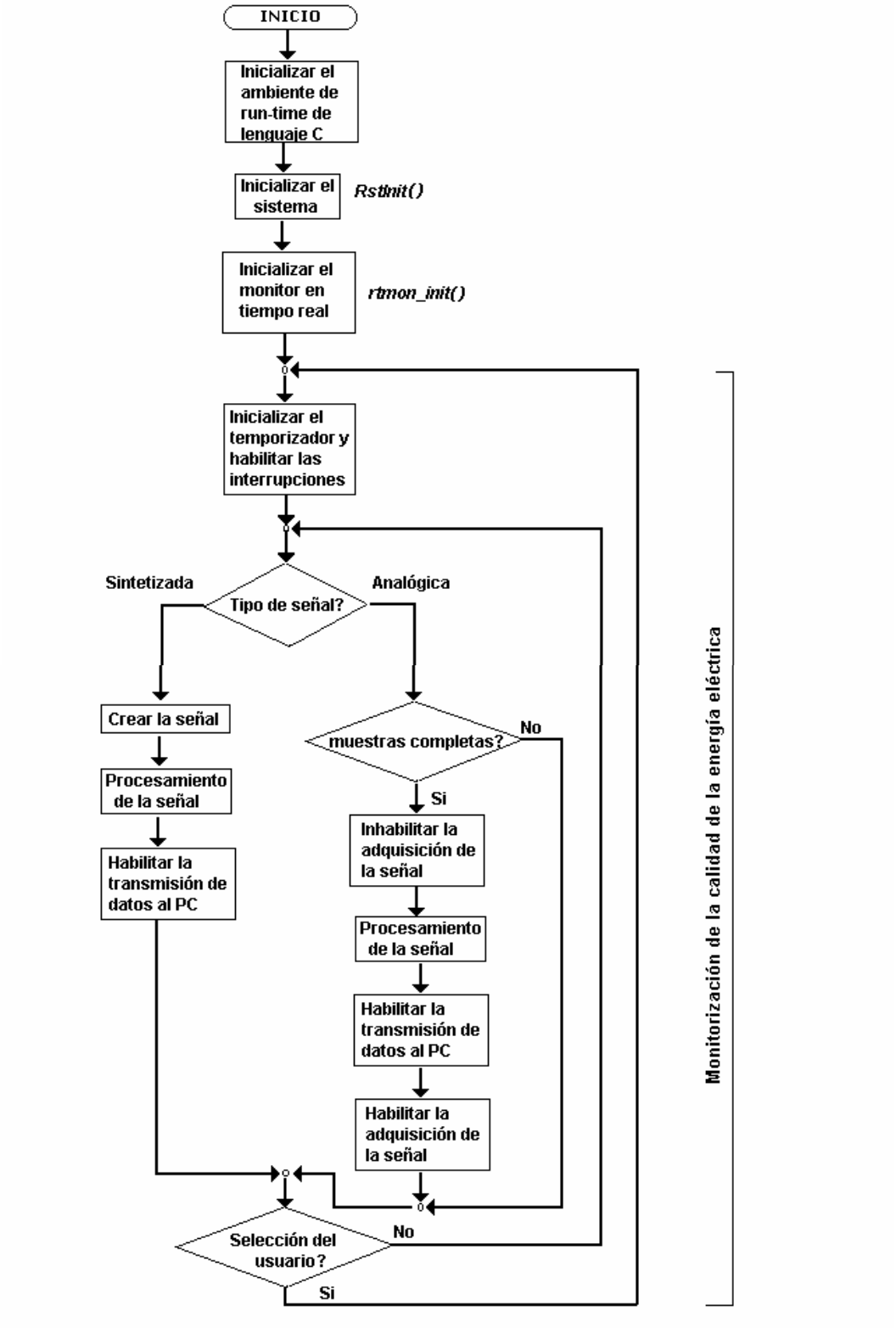
El prototipo implementado consta de tres partes principales: La etapa de digitalización de la señal, la etapa de procesamiento y la etapa de visualización de los resultados. Estas tres etapas pueden apreciarse en la Figura 39.

Figura 39. Etapas del prototipo implementado.



Teniendo estas tres etapas claramente identificadas, se pueden definir cada una de las tareas que se deben realizar para que cada una de ellas funcione correctamente. Estas tareas se pueden resumir de la siguiente forma: Una señal analógica presente en uno de los canales de entrada del conversor *A/D* se debe muestrear y almacenar. Con estas muestras se realiza un procesamiento, dependiendo de la configuración realizada por el usuario. Después de realizado el procesamiento, se envían los datos obtenidos al PC para ser visualizados. En el diagrama de flujo de la Figura 40 se puede identificar la secuencia de ejecución de cada una de estas tareas.

Figura 40. Algoritmo del prototipo de monitor de calidad.



En primer lugar se crea un entorno de *run-time* (para el momento de la ejecución) para el lenguaje C, necesario para poder ejecutar el programa. Este entorno es creado por la función *c_int0*, la cual se encuentra en la rutina de inicio *boot.asm* perteneciente a la librería de soporte de *run-time rts2xx.lib*. Esta función es llamada por el vector de interrupción 0 y las tareas desarrolladas para iniciar el entorno son: definir la sección *stack* o pila del sistema, en donde se almacenan variables locales y se pasan los argumentos a las funciones; inicializar las variables globales y locales y llamar la función *main* para ejecutar el programa.

Luego se necesita inicializar el sistema en una condición "sana" o estable por medio de la función *RstInit().* Esta condición estable se alcanza de la siguiente manera: Inhabilitando el temporizador *watchdog* y reiniciando su contador; estableciendo los estados de espera para acceder a las memorias externas; habilitando el reloj del manejador de eventos (encargado de controlar el temporizador 2, el cual determina el inicio de conversión); habilitando el reloj del conversor análogo digital y de la interfaz de comunicación serial; estableciendo un factor de 4 para el pre-escalamiento del reloj del *DSP;* habilitando la interrupción por desbordamiento del temporizador 2 y limpiando las banderas de interrupciones de los manejadores de eventos; y por último desenmascarando las interrupciones 1,3 y 7 para que puedan ser reconocidas por la *CPU* del *DSP* (la interrupción 1 para el receptor de la interfaz de comunicación serial, la 3 para el temporizador 2 y la 7 para el programa monitor del *DSP* en tiempo real).

La tercera tarea consiste en inicializar la monitorización del *DSP* en tiempo real, realizado por la función *rtmon_init()*.

La cuarta tarea consta de dos partes. La primera es habilitar e inicializar el temporizador 2 y la interfaz de comunicación serial, esto se realiza por

medio de la función *time_base_init(),* y la segunda es habilitar todas las interrupciones que están desenmascaradas, esto se realiza con la función *enable_ints().* Esta cuarta tarea junto con las siguientes se realizan continuamente y conforman el algoritmo de monitorización de la calidad de la energía eléctrica implementado en el prototipo.

Este algoritmo de monitorización se conforma de la siguiente forma: Inicialmente se determina si se van a monitorizar señales sintetizadas o señales análogas adquiridas con el conversor análogo-digital (*ADC*). Para el caso de las sintetizadas se crean estas señales; se procesan (Este proceso incluye enventanado, inversión de bits, hacer la parte imaginaria de la señal cero, calcular la FFT y hallar la magnitud de los coeficientes de *Fourier* obtenidos); y finalmente se transmiten los resultados.

Para las señales análogas se comprueba si se tiene la cantidad de muestras completas. Si están completas, se inhabilita la adquisición de la señal y se procede a realizar su procesamiento (enventanado, inversión de bits, hacer la parte imaginaria de la señal cero, calcular la FFT y hallar la magnitud de los coeficientes de *Fourier* obtenidos), luego se habilita el envío de datos y la adquisición de la señal. Si las muestras no están completas se espera hasta que se adquieran todas, y así el proceso se repite continuamente.

Del diagrama de la Figura 40 la rutina encargada de la selección que puede hacer el usuario es realizada por la interrupción de nivel 1 que controla el DSP. Dentro de este nivel de interrupciones está la interrupción generada por el receptor de la interfaz de comunicación serial. En esta interrupción se compara el valor del registro del buffer del receptor, para ejecutar diferentes opciones. Estas opciones permiten la transmisión de los resultados o la ejecución de sentencias para la configuración de la monitorización (Figura 41). Así mismo en el diagrama

de la figura 40, la adquisición de las muestras es realizada por la interrupción de nivel 3 que controla el *DSP*. Dentro de este nivel de interrupciones está presente la interrupción generada por el desbordamiento del temporizador 2, el cual es el encargado de dar el inicio de conversión *A/D* (genera la frecuencia de muestreo). En esta interrupción se limpian las banderas de interrupciones del manejador de eventos A y se adquiere la señal. El diagrama de flujo de la interrupción 3 se muestra en la Figuras 42.

Figura 41. Diagrama de flujo de la interrupción 1.

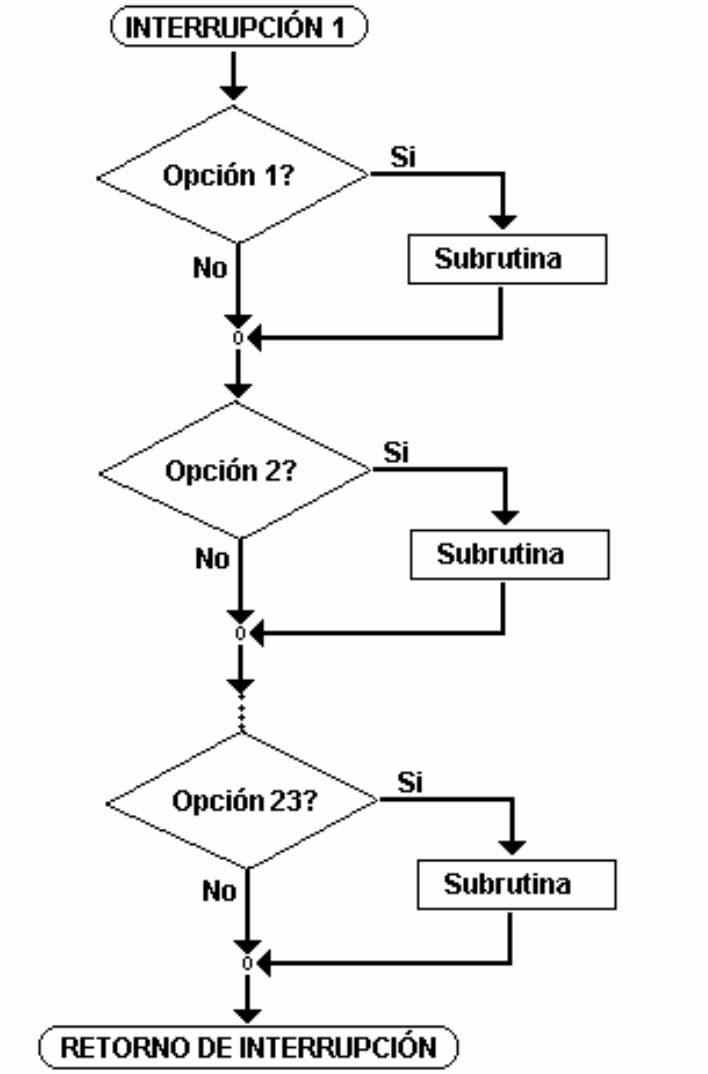
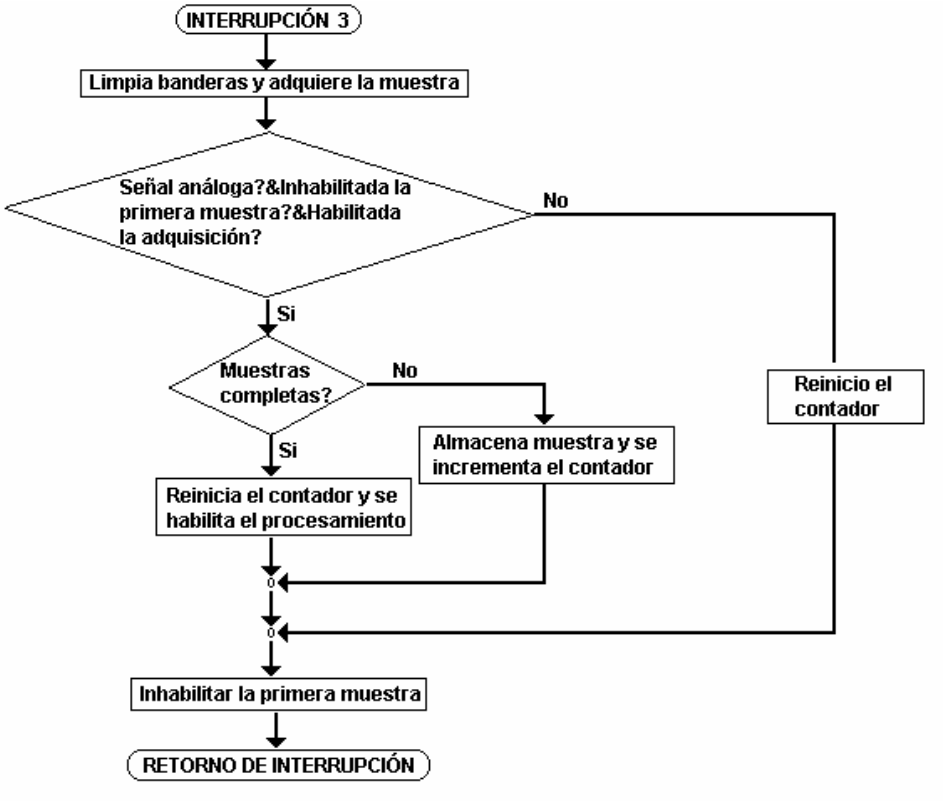


Figura 42. Diagrama de flujo de la interrupción 3.



6.2 REQUERIMIENTOS DEL PROTOTIPO

En la implementación del prototipo se tuvieron en cuenta los siguientes requerimientos de diseño: Se debe inicializar y configurar el DSP para que funcione continuamente permitiendo al usuario configurar el tipo de monitorización de la calidad de la energía eléctrica. Esta configuración debe permitir la selección del tamaño de muestras a procesar mediante la *FFT*, la selección del tipo de ventana que se va a utilizar para la adquisición de estas muestras y también dar la opción de visualizar la componente de continua.

La adquisición de las muestras debe realizarse con el conversor análogo-digital del LF2407A. El procesamiento de las muestras se debe realizar, según [IEC 61000-4-7, 91] en modo de disparo y los resultados deben

tener una incertidumbre que no sea mayor del 5%. Después de procesar las muestras el prototipo debe transmitir los resultados obtenidos a un PC para poder visualizarlos. Esta transmisión debe realizarse por vía serial, aprovechando el módulo de interfaz de comunicación serial que posee el LF2407A a través del conector RS-232 que se encuentra en su módulo de evaluación. El protocolo para esta comunicación es de 1 bit de parada, sin paridad, 8 bit de datos y una velocidad de transmisión de 115 200 baudios por segundo.

El prototipo debe estimar las amplitudes de los armónicos. Estas amplitudes deben ser visualizadas en una pantalla gráfica en valor eficaz. Además, debe incluir pantalla numérica para las amplitudes armónicas, para el valor eficaz de la fundamental y de toda la señal, para la distorsión armónica total (*THD*) y para la componente de continua.

6.3 LIBRERÍAS FFT UTILIZADAS

El software primario para la implementación del algoritmo FFT fue la librería para el procesamiento de señales desarrollada por *TI's Foundation Software*, en la cual se encuentran los diferentes algoritmos para calcular la FFT de una señal. Con esta librería se puede realizar la FFT real y compleja de una señal de 128 puntos, de 256 puntos y de 512 puntos. Para poder ejecutar y comprobar cada uno de estos algoritmos fue necesario descargar la librería *Software Test Bench (STB).* En esta librería se encuentran definiciones de registros y funciones ya desarrolladas para aprovechar al máximo las capacidades del módulo de evaluación; las cuales son utilizadas por la librería FFT.

6.3.1 MODIFICACIONES REALIZADAS A LAS LIBRERÍAS

En este numeral se especifican las modificaciones realizadas a las librerías durante el trabajo de la implementación del prototipo, las cuales fueron

necesarias para satisfacer los requerimientos establecidos para el proceso de monitorización de la calidad de la energía eléctrica.

De los requerimientos establecidos anteriormente los que demandaron modificación en las librerías fueron: La configuración de la memoria del *DSP*; el cálculo de la *FFT* de un número de *L* (*L*= 128, 256, 512, 1024 y 2048) muestras seleccionado por la configuración realizada por el usuario para la monitorización; realizar el enventanado a las muestras a procesar; la adquisición de las muestras con el conversor análogo-digital y la transmisión de los resultados por la interfaz de comunicación serial.

En primer lugar, para calcular la FFT de 128, 256, 512, 1024 y 2048 muestras no fue posible utilizar las rutinas encargadas del cálculo de la FFT, del enventanado, de la inversión de bits y de hacer la parte imaginaria cero; las cuales se encontraban en la librería para el tratamiento de señales. Las dos razones para no utilizarlas fueron: La primera, que la rutina que se encargaba del cálculo de la FFT era solamente para un número fijo de muestras y no permitía al usuario seleccionar la configuración para la monitorización de la calidad de la energía eléctrica. Como las muestras adquiridas se almacenan en un solo vector (cuyo tamaño es de 4096 muestras para permitir el cálculo de la FFT de 2048 muestras) para ahorrar memoria, el problema que surgió fue que las rutinas de enventanado, inversión de bits, de hacer la parte imaginaria cero y calcular la magnitud de los coeficientes de *Fourier* realizaban su función en todo el vector y no en la cantidad de muestras a procesar establecidas por la configuración del monitor, produciendo resultados erróneos.

Esto quiere decir que las funciones encargadas del ordenamiento de inversión de bits, de hacer la parte imaginaria cero del vector de muestras, de cambiar la forma de la ventana de adquisición, de calcular la

FFT y de hallar su magnitud no fueron usadas. Por lo tanto estas funciones fueron remplazadas por las siguientes funciones que fueron desarrolladas por el autor de este proyecto:

1. *int bitreversal(int *ipcb, int lim1, int muestras)*: Esta función calcula el ordenamiento de inversión de bits de cualquier número de muestras que sea igual a una potencia de 2. En esta función *ipcb* es el puntero de la posición inicial del vector de muestras. El parámetro *lim1* es el número de bits necesarios para expresar el número total de las muestras, esto quiere decir que si las muestras son iguales a 1024, *lim1* debe ser igual a 10, ya que $2^{10}=1024$. Por último esta el parámetro *muestras*, que es el número de puntos para el cual se lleva a cabo el ordenamiento de inversión de bits.
2. *int venHHB(int *ipcb,int muestras,volatile int *facthhb):* Con esta función se realiza el enventanado *Hanning, Hamming* o *Blackman* de las muestras si no se va a utilizar una ventana rectangular. El parámetro *ipcb* es un puntero a la posición inicial del vector de muestras adquirido. El segundo parámetro es el número de muestras adquirido. El tercero y último es el puntero a la posición inicial de las tablas en las cuales se encuentran los coeficientes de las ventanas *Hanning, Hamming* o *Blackman*.
3. *int imacero(int *ipcb,int muestras):* Esta función se encarga de volver cero la parte imaginaria de cada una de las muestras cero. El primer parámetro es el puntero a la posición inicial del vector de muestras y el segundo es el número de muestras adquirido.
4. *int FFTL(int *ipcb, int lim1, int muestras, int cosen, volatile int *twfactor):* Esta función es la encargada de calcular la FFT de una cantidad L de muestras adquiridas en tiempo real o sintetizadas por medio de *software*. La rutina que realiza esta función se basa en lo explicado en el capítulo 5 para la

implementación de la transformada rápida de Fourier. Por lo tanto usando la Ecuación (13), la (14), la (15) y la (16), los límites deducidos para cada uno de los ciclos y sus pasos de incremento se logró definir y declarar la función encargada del cálculo de la FFT. El primer parámetro es el puntero a la posición inicial del vector de muestras. El segundo parámetro es el límite para el ciclo externo, es decir, el número de etapas a recorrer. El tercero, es el número de muestras a calcula la FFT. El cuarto es el valor de desplazamiento que se tiene que usar para los coeficientes de la función coseno utilizados en los factores de ganancia (*twiddle factors*) para los cálculos de las mariposas. El último es el puntero a la posición inicial de los factores de ganancia.

5. *int magnitud(int *ipcb,int *salida,int muestras):* Esta se encarga de calcular la magnitud al cuadrado de la FFT. El resultado de esta función es la suma de los cuadrados de la parte real e imaginaria de cada uno de los coeficientes. El primer parámetro es el puntero a la posición inicial del vector en el cual se encuentra el resultado de la FFT. El segundo parámetro es el puntero a la posición inicial del vector donde se almacena la magnitud de estos coeficientes. El tercer y último parámetro es el número de muestras a las cuales se les calculó la FFT.
6. *int po2(int exponente):* Esta función se implementó para el cálculo de las potencias de 2, donde el único parámetro es el exponente al cual se eleva. Esta función se utiliza en la función *bitreversal*, y su implementación se debió a que la función estándar pow(x,y), la cual calcula x^y, producía resultados erróneos.

Para adquirir una cantidad de muestras de la señal por medio del conversor análogo-digital del *DSP*. Fue necesario también modificar el

módulo encargado del manejo del conversor A/D, perteneciente a la librería *STB*. Este módulo la adquisición por medio de la interrupción de desbordamiento por debajo (*underflow*) generada por el temporizador 2. cuando el temporizador llega a cero se adquiere la muestra y por lo tanto se debe colocar un valor preciso en el valor del registro T2PR para establecer una frecuencia de muestreo de 7680 Hz (128 muestras por ciclo para una señal de 60 Hz), en el registro T2PR. En consecuencia se realizaron 2 modificaciones:

1. La primera modificación consistió en cambiar el valor que se almacena en el registro T2PR, debido a que los algoritmos para la FFT de la librería se ajustaban a una frecuencia de muestreo de 20 kHz. Este valor se calculó de la siguiente forma: (1/7680)*40 MHz[*]. Este valor de 5208.33, el cual se aproximó a 5208 ya que en los registros del DSP se almacenan solo valores enteros. Este valor nos da una frecuencia de muestreo de 7680.491 Hz.
2. La segunda modificación se realizó en la rutina de atención de la interrupción del temporizador. Esta modificación consta de dos partes y se realizo en la función *void interrupt c_int03*(). La primera fue la elaboración de la rutina adquiera el número total de muestras seleccionado. Esta rutina ignora la primera muestra adquirida, no permite que se realice la rutina de calculo de la FFT sino hasta que se completen todas las muestras y evita que se vuelvan a adquirir muestras hasta que se haya realizado el calculo de la FFT y se hayan transmitido los datos, ya que el calculó se realiza sobre el mismo vector donde se almacenan los datos adquiridos por el conversor. La segunda modificación tiene que ver con el resultado obtenido por el módulo que maneja el conversor. Para llevar a cabo esta modificación se

[*] 40 Mhz es la frecuencia del reloj interno CLKIN el cual es el mismo

estudió el algoritmo utilizado para este módulo y se realizaron las operaciones necesarias para obtener un valor preciso en formato Q15 de la señal muestreada.

En cuanto al análisis y procesamiento en tiempo real no se realizó modificación alguna, aprovechando la configuración que venía establecida por el algoritmo FFT. Esta configuración es establecida por el archivo de encabezado *sysvecs.h* y los archivos *c200mnrt.asm* y *c200mnrt.i*. Además de estos archivo se encuentra la definición de REAL_TIME= 1, con la cual se habilita la interrupción para el manejo del monitor en tiempo real, el cual es el encargado de seguir atendiendo las rutinas de interrupciones aunque el *DSP* este detenido.

La transmisión de los datos obtenidos del procesamiento de la señal para ser visualizados y almacenados debió implementarse completamente. En primer lugar es necesario habilitar el reloj para el módulo de la interfaz de comunicación serial (*SCI*) en el registro SCSR1. El segundo paso consiste en configurar el módulo *SCI* por medio de sus respectivos registros. Esta configuración se realizó modificando la función *void time_base_init(void)*, de la librería para calcular la *FFT*. En esta se agregaron las sentencias para inicializar y establecer el funcionamiento de la interfaz de comunicación serial. Para esto se seleccionan los pines del receptor y del transmisor por medio del registro de control A (*MCRA*) del multiplexor para los pines de entrada y salida de propósito general (*GPIO*); se habilitan el transmisor y el receptor del módulo; se reinicia el módulo; se selecciona el modo de línea inactiva; se establece la velocidad de transmisión de los datos (la cual es de 115 200 baudios); se habilita la interrupción generada por el receptor de tal forma que sea manejada por la interrupción 1 del *DSP*; y se establece el protocolo de comunicación.

Este protocolo está compuesto por: un bit de parada, sin paridad, modo de prueba de lazo cerrado inhabilitado, modo de línea inactiva y 8 bits de datos.

También fue necesario desenmascarar la interrupción 1 del *DSP* cambiando el valor que se almacena en el registro de enmascaramiento de interrupciones (*IMR*), el cual se configura en la función *void RstInit(void),* de la librería base o plantilla.

Además de la interrupción 1 tienen que estar activadas o desenmascaradas dos interrupciones más: la del temporizador que se encuentra en la interrupción 3 y la del monitor de tiempo real que es la interupción 7.

Por último, para transmitir los datos obtenidos, se implementó una rutina que envíe los datos dentro de las sentencias que manejan las demandas de la interrupción 1. Esta rutina está declarada por la siguiente sentencia: *void interrupt c_int01().* Dentro de esta rutina se encuentran 23 subgrupos de código encargados de permitirle al usuario cambiar las opciones de monitorización, y uno de ellos tiene la finalidad de la transmisión de datos.

Los subgrupos son usados para la configuración de la monitorización y funcionan de la siguiente manera: para cada selección que el usuario realice está asignado un valor, el cual se envía por el puerto serial al *DSP*. En el momento que este valor llega al *buffer* del *DSP*, ocurre una demanda para la interrupción 1 y se dirige a la rutina de atención general de interrupción (*GISR, general interrupt service routine*) correspondiente (*void interrupt c_int01*).

Si en el *buffer* esta, por ejemplo, una A se ejecuta la transmisión de los datos; si hay una B (0x0042) el usuario seleccionó calcular la *FFT* de 2048 muestras adquiridas con una ventana rectangular, y así para las diversas opciones configuradas. Los datos que se transmiten son palabras (un tamaño de 16 bits), y deben ser divididos en dos *bytes* (8 bits). Además, es necesario agregar unos retardos entre cada una de las transmisiones de los datos, debido al modo de línea inactiva seleccionado, para que los datos pudieran ser reconocidos y enviados.

El archivo de enlace (**.cmd*) para la configuración de la memoria, la definición de las secciones de salida y el direccionamiento es el archivo *f128cd.cmd*. En este archivo se modificó la dirección de la porción de memoria *ExtRam,* la cual pertenece a la memoria externa. Así mismo, se modifico el valor de alineamiento de la sección *FFTipcb*, debido a que para nuestro algoritmo el tamaño máximo del vector de muestras es 4096, en donde se almacenan 2048 puntos cada uno con parte real e imaginaria, y por lo tanto el tamaño reservado en la memoria tiene que ser del doble. La última modificación del archivo *f128cd.cmd* fue asignarle a la sección de memoria *FFTwin* una porción de memoria en la cual se copia cuando se almacena el programa en el *DSP* y otra porción en donde se copia cuando el programa almacenado en el *DSP* se esta ejecutando.

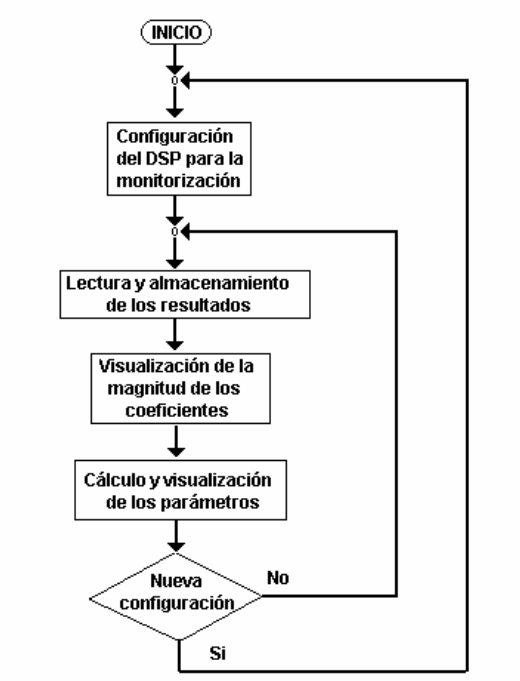
Para el funcionamiento del monitor de calidad de la energía eléctrica se agregaron cuatro archivos encargados de copiar las tablas de los coeficientes de los factores de ganancia de las mariposas para el cálculo de la *FFT* y los coeficientes de las ventanas *Hanning, Hamming y Blackman*. Estos datos se copian desde la memoria de programa a la memoria de datos para tener un acceso más rápido. En cada uno de estos archivos, elaborados en lenguaje *assembly*, se encuentran definidas y declaradas las funciones encargadas de copiar estos coeficientes de acuerdo con la ventana seleccionada y el número de puntos de la *FFT.*

6.4 INTERFAZ DE VISUALIZACIÓN

La plataforma seleccionada para la visualización de los datos obtenidos del procesamiento fue *LabView* debido a que es un ambiente gráfico que lo hace una herramienta poderosa, versátil y muy completa para el desarrollo de aplicaciones de control, análisis y visualización de resultados.

El algoritmo elaborado en este ambiente consiste de tres bloques. Este algoritmo se presenta en la Figura 43.

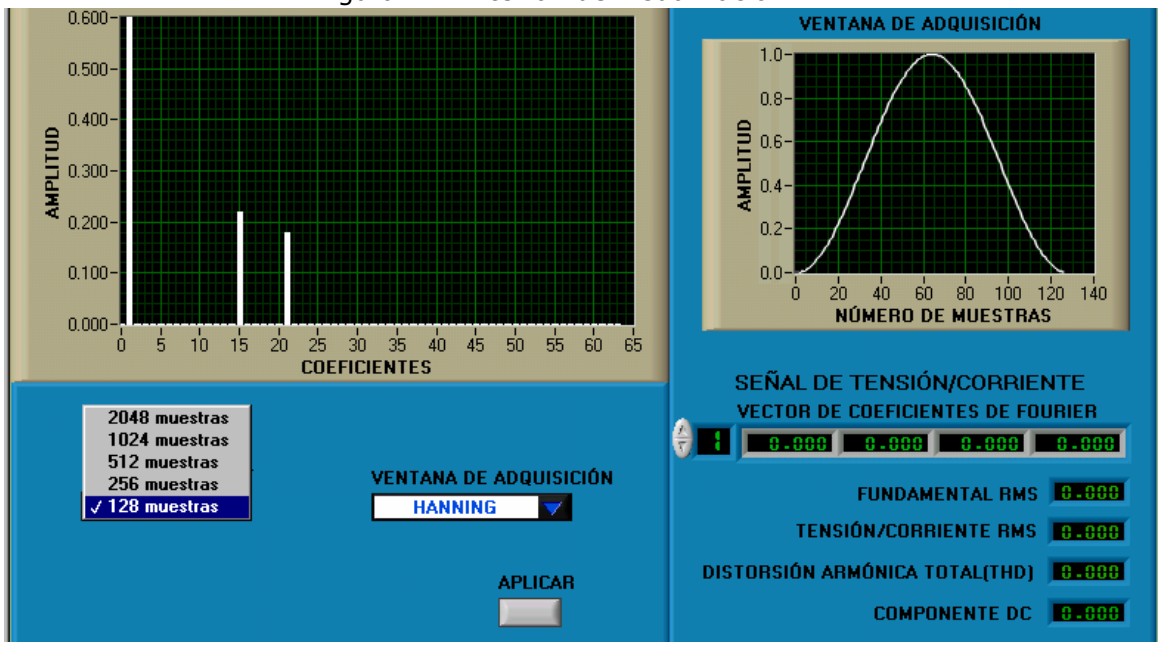
Figura 43. Algoritmo de la interfaz de visualización.



El primer bloque es el encargado de llevar a cabo la configuración del equipo para la monitorización. Esta configuración se hace por medio de la selección de las características de la transformada de *Fourier*. El segundo bloque es el que realiza la adquisición de los datos enviados por el puerto serial. Esta adquisición funciona de la siguiente manera: Primero el

computador espera que en el *buffer* se encuentren todas las muestras. Cuando los elementos en el *buffer* tienen la capacidad indicada, según los puntos de la *FFT* escogida, se van leyendo y almacenando los datos en un vector. El último bloque es el encargado de la visualización del vector de datos obtenidos del puerto serie y del cálculo de los parámetros de la señal. Estos tres bloques se repiten continuamente permitiendo una monitorización en tiempo real (*on-line*).

Figura 44. Interfaz de visualización.



En la Figura 44 se puede apreciar el ambiente gráfico implementado para la visualización y configuración del monitor. En esta interfaz se encuentra una gráfica de visualización de la magnitud de coeficientes de Fourier obtenidos por el procesamiento de las muestras en el *DSP*, una gráfica para visualizar la forma seleccionada de las ventanas de adquisición de las muestras, y los resultados de los valores eficaces del fundamental y de la señal completa, la distorsión armónica total (*THD*), la componente de continua y el vector con las magnitudes de los coeficientes de Fourier.

6.5 RESULTADOS DEL PROTOTIPO

Para comprobar y evaluar el desempeño del prototipo implementado se realizaron varias pruebas con señales sintetizadas y con señales análogas procedentes de un generador de señales. Las pruebas realizadas con las señales sintetizadas consistieron: primero en sintetizar señales con una sola componente de frecuencia con diferentes amplitudes y luego se crearon señales correspondientes a la suma de tres sinusoidales. En la Tabla 5 se muestran resultados de las pruebas con las señales sintetizadas con una componente de frecuencia para una FFT de 1024 muestras.

Tabla 5. Resultados de la señal sintetizada con una sola componente de frecuencia.

Orden de los armónicos	Amplitud de la señal sintetizada (p.u.)	Amplitud calculada por el monitor (p.u.)	Amplitud calculada en *Matlab* (p.u.)	Error con respecto a los resultados de Matlab(%)
1	1	1	1	0
3	0.8	0.8	0.8	0
5	0.67	0.67	0.67	0
7	0.51	0.51	0.51	0
10	0.73	0.73	0.73	0
15	0.26	0.26	0.26	0
19	0.505	0.505	0.505	0
21	0.231	0.231	0.231	0
23	0.16	0.16	0.16	0
25	0.31	0.31	0.31	0
27	0.42	0.42	0.42	0
30	0.218	0.218	0.218	0
31	0.184	0.184	0.184	0
33	0.126	0.125	0.126	0.79
35	0.142	0.141	0.142	0.7
37	0.175	0.175	0.175	0
39	0.153	0.153	0.153	0
41	0.148	0.147	0.148	0.67
43	0.1	0.099	0.1	1

En la Tabla 5 se puede ver que cuando la amplitud de la señal sintetizada es aproximadamente menor de 0.15 p.u. comienza a haber errores. Estos se deben a que el LF2407A es de coma fija, y por lo tanto aparecen los siguientes errores: el redondeo en el cálculo de las factores de ganancia (*twidle factors*) al pasar a formato Q15; el redondeo de los valores de la señal sintetizada al pasar a formato Q15; y el redondeo en las operaciones que realiza la *FFT* por el formato Q15.

En la Tabla 6 se encuentra uno de los resultados de las segundas pruebas con señales sintetizadas, la cual fue sintetizar una suma de tres sinusoidales. Estos resultados fueron obtenidos con una FFT de 1024 muestras.

Tabla 6. Resultados de la señal sintetizada compuesta de tres sinusoidales.

Orden de los armónicos	**Amplitud de la señal sintetizada (p.u.)**	**Amplitud calculada por el monitor (p.u.)**	**Amplitud calculada en *Matlab* (p.u.)**	**Error con respecto a los resultados de Matlab(%)**
1	0.6	0.6	0.6	0
15	0.22	0.22	0.22	0
21	0.18	0.18	0.18	0

En la Figura 45 se muestra una gráfica de amplitud contra el error cometido por el monitor con respecto a los resultados de *Matlab* y en la Figura 46 están estos mismos errores pero en escala logarítmica. Estos datos se calcularon de señales sintetizadas cuya amplitud se encuentra entre 0.1 y 0.015 p.u. En la Tabla 7 están los valores y sus respectivos errores.

Tabla 7. Resultados de la señal sintetizada compuesta de tres sinusoidales.

Orden de los armónicos	Amplitud de la señal sintetizada (p.u.)	Amplitud calculada por el monitor (p.u.)	Amplitud calculada en *Matlab* (p.u.)	Error con respecto a los resultados de Matlab(%)
1	0.100	0.099	0.100	1.00
2	0.095	0.099	0.095	1.05
3	0.090	0.090	0.090	0
4	0.085	0.084	0.085	1.17
5	0.080	0.080	0.080	0
6	0.075	0.074	0.075	1.33
7	0.070	0.070	0.070	0
8	0.065	0.064	0.065	1.53
9	0.060	0.059	0.060	1.67
10	0.055	0.054	0.055	1.81
11	0.050	0.049	0.050	2.00
12	0.045	0.044	0.045	2.22
13	0.040	0.040	0.040	0
14	0.035	0.033	0.035	5.71
15	0.030	0.029	0.030	3.33
16	0.025	0.025	0.025	0
17	0.020	0.019	0.020	5.00
18	0.015	0.011	0.015	26.67

Figura 45. Amplitud calculada por el monitor vs. error con respecto a los resultados de *Matlab*.

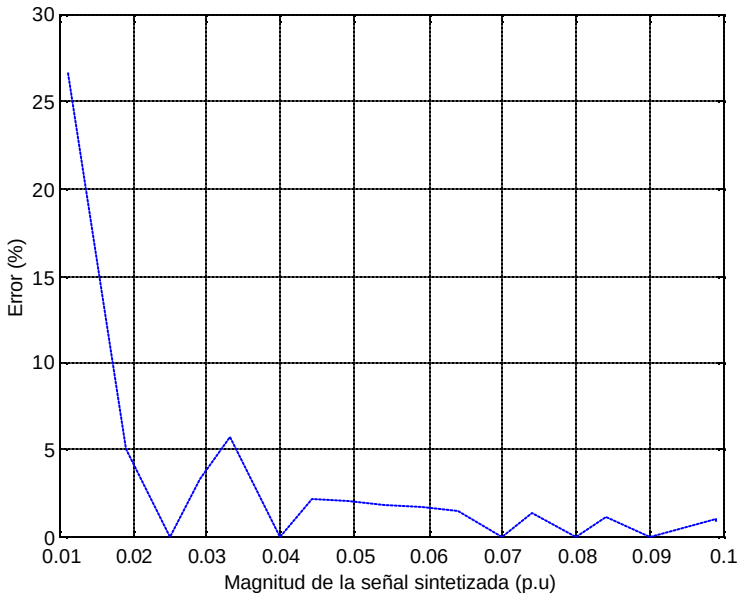
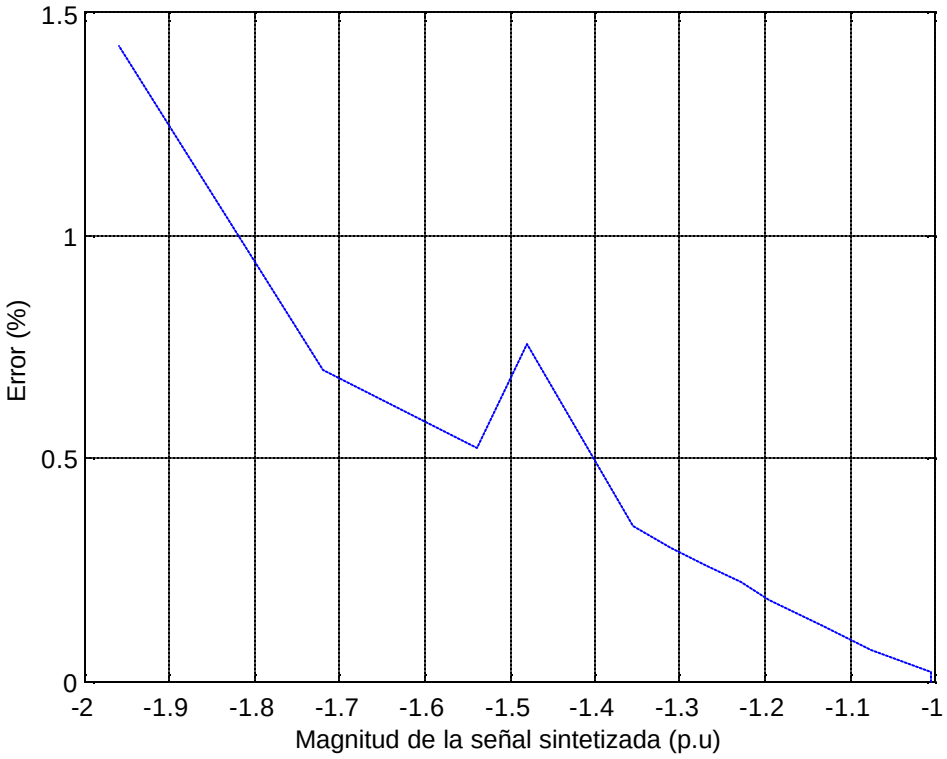


Figura 46. Amplitud calculada por el monitor vs. error con respecto a los resultados de *Matlab* (escala logarítmica).



Para las pruebas con el generador de señales la frecuencia de la señal generada no era estable, tenía una variación de ± 0.7 Hz, y por lo tanto la frecuencia de muestreo se estableció para adquirir 128 muestras de una frecuencia promedio de la señal.

En la Tabla 8 se muestran los resultados de unas pruebas con señales análogas y sus errores. Estas pruebas se hicieron para determinar la señal con menor magnitud que se puede procesar. Para la parte de la frecuencia y la componente de continua no se tiene ningún problema. El error obtenido para la componente de continua fue de 0.59%.

Para valores menores a 86 mV los resultados tenían errores mayores al 5%. Las fuentes de estos errores para las señales analógicas son: la variación de la frecuencia de la señal generada que produce un

deslizamiento de frecuencia; los errores de cuantificación debida a el conversor análogo-digital *A/D*; y los errores de redondeo debidos al formato Q15 en el cálculo de los *Twidle Factors* y en las operaciones que realiza la *FFT*.

Tabla 8. Resultados de la señal análoga.

Frecuencia de 60.2 Hz		
Magnitud de la señal generada (mV)	**Magnitud de la señal calculada (mV)**	**Error (%)**
980	949	3.1
900	886	1.5
330	326	1.2
278	270	2.8
150.5	146	2.99
114	109	4.38
86	82	4.65

7 CONLUSIONES Y TRABAJOS FUTUROS

En este capítulo se recopilan los principales resultados de los diferentes capítulos y se agregan algunas conclusiones generales; además, se propones algunos temas para continuar con este trabajo.

7.1 CONCLUSIONES

Se ha implementado un prototipo de unidad de procesamiento basado en *DSP* para la monitorización de la calidad de la energía eléctrica. Este prototipo permite a un usuario configurar el tipo de monitorización a realizar según el análisis requerido y visualizar los resultados en un PC de forma gráfica y numérica. Las opciones implementadas para la monitorización son: Seleccionar el tipo de señales a monitorizar (señales sintetizadas o analógicas); seleccionar el tipo de ventana para la adquisición de las muestras de la señal; seleccionar la cantidad de muestras a procesar; y en el caso de las señales análogas seleccionar si se quiere visualizar la componente de continua.

Este prototipo utilizó el *DSP* TMS320LF2407A de *Texas Instruments*. El *DSP* LF2407A cuenta con: un procesador de 16 bit; 128 k palabras de memoria (64k de memoria de programa y 64 k de memoria de datos); gran velocidad de procesamiento (40 millones de instrucciones por segundo, *MIPS*) que le permite procesar señales en tiempo real; una arquitectura *Hardware* mejorada por la utilización de dos buses independientes (uno para memoria de programa y otro para memoria de datos) y dos desplazadores en *hardware* disponibles para escalar números (los cuales permiten minimizar errores de cuantificación y de truncamiento), permitiendo el procesamiento de señales de control; y un

controlador de periféricos (conversor análogo-digital, dos puertos de comunicación serial, dos manejadores de eventos, un controlador de red local , entre otros). Todas estas características permiten que el LF2407A satisfaga los requerimientos de las aplicaciones de control, monitorización y evaluación de sistemas eléctricos.

El prototipo implementado permite monitorizar otras componentes de frecuencia diferentes a los armónicos, debido a que se utilizan ventanas de ponderación o adquisición mayores a un ciclo. Estas componentes de frecuencia son los interarmónicos y subarmónicos. Cuando se utilizan ventanas de adquisición mayores a un ciclo se logra un aumento en la resolución de la frecuencia. Esta resolución es igual a la frecuencia de muestreo sobre el número de muestras a procesar.

El módulo de evaluación TMS320LF2407 EVM de *Texas Instruments*, el cual pertenece a la Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones, fue el *hardware* utilizado para la implementación del prototipo de unidad de procesamiento. Este módulo permite examinar las características del LF2407A, debido a que por medio de él se pueden acceder a todos los periféricos y pines de control, de estado, de datos y de direcciones del LF2407A. Así mismo, permite la emulación y programación de algoritmos en el *DSP* a través del puerto *JTAG* del LF2407A. Por lo tanto fue la herramienta usada para el desarrollo, comprobación y evaluación del algoritmo a implementar en el *DSP*.

La adquisición de la señal análoga se realizó a través del conversor análogo-digital (*ADC*), a través de la interrupción de nivel 1 del LF2407A. Esta interrupción es demandada por el temporizador 2 del manejador de eventos. La frecuencia de muestreo que se requería para adquirir 128 muestras por ciclo asumiendo una frecuencia de 60 Hz para la señal a analizar, es de 7680 Hz. Esta frecuencia no se pudo obtener debido a que

el registro que controla la frecuencia de muestreo (este registro es el resultado de dividir el reloj del *DSP* sobre la frecuencia de muestreo), es de tipo entero. Por lo tanto la frecuencia de muestreo implementada fue de 7680. 49 Hz, que funcionaría correctamente para una señal análoga con una frecuencia de 60.0038 Hz. Además, la señal a muestrear tiene que ser unipolar y su rango va de 0 a 3.3 Voltios.

La transmisión de los resultados y la selección de la configuración de la monitorización se realizó por medio de la interfaz de comunicación serial (*serial communications interface, SCI*) del LF2407A. Esta comunicación es controlada por la interrupción de nivel 3 del LF2407A. El protocolo de comunicación que se implementó para el prototipo fue: 8 bit de datos; 1 bit de parada; sin paridad y con una velocidad de transmisión de 115200 baudios por segundo. El *SCI* se configuró para funcionar en modo de comunicación de línea inactiva para un transmisor y receptor asíncrono universal (*universal asynchronous receiver/transmitter*, *UART*).

La visualización de los resultados y el control para le selección de la monitorización se realiza a través de un PC, el cual está conectado vía serial con el módulo de evaluación. Esta interfaz de comunicación se implementó con el *software* de programación *LabView*. Las características de esta interfaz son: una pantalla gráfica y una pantalla numérica para visualizar las magnitudes de los armónicos; una pantalla gráfica para visualizar la forma de la ventana de adquisición; botones de configuración de la monitorización; y pantallas numéricas para visualizar los valores *rms* de la fundamental y de la señal completa, la distorsión armónica total (*THD*) y la componente de continua.

Se desarrollaron 8 rutinas para el procesamiento de las señales a monitorizar con el prototipo. Estas rutinas son: La rutina encargada de calcular la FFT para un número L de muestras, donde L es una potencia

de 2, y se implementó para 128, 256, 512, 1024 y 2048 muestras; la rutina que realiza el ordenamiento de inversión de bits para una cantidad L de muestras, donde L es potencia de 2; la rutina que hace la parte imaginaria de la señal cero; la rutina que calcula la magnitud al cuadrado de los coeficientes de *Fourier* de la señal monitorizada; y 3 rutinas encargadas de realizar el enventanado de las muestras de la señal, las ventanas implementadas fueron *Hanning*, *Hamming* y *Blackman*.

El algoritmo diseñado corrobora la versatilidad y eficiencia para la implementación de aplicaciones con el módulo de evaluación, estableciendo las bases necesarias para la implementación de aplicaciones más complejas, que usen toda la capacidad de procesamiento del LF2407A y un manejo de más periféricos.

El *Code Composer Studio* presenta herramientas de monitorización que permiten realizar el seguimiento de la aplicación en tiempo real visualizando: el programa cargado en la *CPU* del *DSP*; la ejecución de las interrupciones y rutinas desarrolladas; los registros de control y estado del *DSP*; las memorias de programa, de datos y espacio de entrada y salida (*I/O*); y variables usadas en el algoritmo. Estas herramientas sirven para el desarrollo, depuración, comprobación y evaluación del algoritmo a implementar en el *DSP*.

7.2 TRABAJOS FUTUROS

El prototipo de unidad de procesamiento implementado establece el punto de partida para el desarrollo de un monitor de calidad de la energía eléctrica. A continuación se presentas algunas recomendaciones para mejorar este prototipo y desarrollar un monitor que cumpla con todos los requerimientos establecidos por las normas y estándares internacionales.

El algoritmo implementado en el prototipo no produce buenos resultados con señales muy pequeñas, con amplitudes menores a 150mV cuando son análogas y menores a 0.03 p.u. cuando son sintetizadas. Es conveniente mejorar este límite si se pretende desarrollar un equipo para realizar una monitorización real de la calidad de la energía eléctrica, donde la magnitud de un armónico puede llegar a ser hasta el 0.7% de la amplitud de la fundamental.

Es conveniente agregar la opción de visualización de la señal monitorizada y la opción de almacenamiento de los resultados según los patrones a monitorizar. Estos patrones pueden ser: Los promedios de las magnitudes de los armónicos en un intervalo de tiempo; monitorizar niveles umbrales de armónicos permitidos en cierta área; y hacer un seguimiento de tensión.

Es posible mejorar el modo disparo de operación del prototipo utilizando rutinas de adquisición que le permitan al usuario operar el prototipo en modo continuo y con solapamiento de ventanas. Con estos modos de operación se puede realizar la monitorización de armónicos fluctuantes y rápidamente variables.

Es importante también implementar una rutina de sincronía para la frecuencia de la señal a monitorizar, según los valores permitidos por el registro de control del inicio de conversión para la adquisición de la señal análoga. Estos valores se encuentran en la Tabla 10 del Anexo D.

Para adquirir muestras de las señales del sistema eléctrico es necesario desarrollar la etapa de adecuación de la señal y así implementar un prototipo completo de monitor de la calidad de la energía eléctrica. Esta etapa de adecuación debe cumplir con los requisitos de las normativas internacionales.

Para monitorizar otros tipos de eventos es posible implementar más algoritmos, aprovechando la capacidad de memoria y de procesamiento que posee el LF2407A, para desarrollar más técnicas para el procesamiento de las señales; y así, poder monitorizar otras perturbaciones (huecos de tensión, elevaciones de tensión, fluctuaciones de tensión, muescas de tensión, etc). Entre estas técnicas están: el filtrado *Kalman* y la transformada *Wavelet*.

REFERENCIAS BIBLIOGRÁFICAS

[T.I spru357b, 01] *Texas Instruments*. "TMS320LF/LC240xA *DSP Controllers Reference Guide: System and peripherals*". 2001.

[T.I evm2407b, 00] *Texas Instruments*. "TMS320LF2407 *Evaluation Mode: Technical Reference*". 2000.

[T.I spru160c, 99] *Texas Instruments*. "TMS320F/C24x DSP *Controllers Reference Guide: CPU and instruction set*". 1999.

[T.I spru357b, 01] *Texas Instruments*. "TMS320LF/LC240xA *DSP Controllers Reference Guide: System and peripherals*". 2001.

[T.I spru018d, 95] *Texas Instruments*. "TMS320C1x/C2x/C2xx/C5x *Assembly LanguageTools User's Guide*". 1995.

[T.I spru024e, 99] *Texas Instruments*. "TMS320C2x/C2xx/C5x *Optimizing C Compiler User's Guide*". 1999.

[IEEE 1159, 95] IEEE *Standards coordinating committee 22 on power quality*, USA. IEEE Std 1159-1995: "IEEE *Recommended practice for monitoring electric power quality*", IEEE *Standards boards*, 1995.

[NTC 5000, 02] Norma Técnica Colombiana 5000: "Calidad de la potencia eléctrica (CPE). Definiciones y términos fundamentales", Instituto Colombiano de Normas Técnicas (ICONTEC), 2002.

[IEEE 519, 92] IEEE Std. 519. "*Recommended Practices and Requirements for Harmonic Control in Electric Power Systems*". 100 p., New York, U.S.A, 1992.

[IEC 6 1000-4-7, 91] COMISIÓN ELECTROTÉCNICA INTERNACIONAL. IEC 61 000-4-7. "Compatibilidad Electromagnética (EMC)-Parte 4: Técnicas de ensayo y medida. Sección 7: Guía general relativa a las medidas de armónicos e interarmónicos, asÍ como a los aparatos de medida, aplicable a las redes de alimentación y a los aparatos conectados a éstas". 32 p., Ginebra, Suiza, 1991.

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

[Bellanger, 94] Bellanger M. "*Digital Processing of Signals: theory and practice*". Segunda edición, *John Wiley & Sons*, 1994. Pp. 388.

[Smith, 99] Smith, Steven. "*The Scientist and Engineer's Guide to Digital Signal Processing*". Segunda edición, *California Technical Publishing*, 1999.

[Proakis & Manolakis, 98] Proakis, John y Manolakis, Dimitris. "Tratamiento digital de señales". Tercera edición, *Prentice Hall*, Madrid, 1998.

BIBLIOGRAFÍA

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

[Proakis & Manolakis, 98] Proakis, John y Manolakis, Dimitris. "Tratamiento digital de señales". Tercera edición, *Prentice Hall*, Madrid, 1998.

[Smith, 99] Smith, Steven. "*The Scientist and Engineer's Guide to Digital Signal Processing*". Segunda edición, *California Technical Publishing*, 1999.

[Bellanger, 94] Bellanger M. "*Digital Processing of Signals: theory and practice*". Segunda edición, *John Wiley & Sons*, 1994. Pp. 388.

[Deitel & Deitel, 92] Deitel, H.M y Deitel, P.J. "*C How to program*". Primera edición, *Prentice Hall Internacional Editions*, 1992.

[Páez, 99] Páez, Manuel. "C y C++ de afán". Primera edición, Editorial Universidad de Antioquia, 1999.

[T.I spru328b, 00] *Texas Instruments*. "*Code Composer Studio User's Guide*". 2000.

[T.I spru018d, 95] *Texas Instruments*. "TMS320C1x/C2x/C2xx/C5x *Assembly LanguageTools User's Guide*". 1995.

[T.I spru024e, 99] *Texas Instruments*. "TMS320C2x/C2xx/C5x *Optimizing C Compiler User's Guide*". 1999.

[T.I spru160c, 99] *Texas Instruments*. "TMS320F/C24x *DSP Controllers Reference Guide: CPU and instruction set*". 1999.

[T.I spru357b, 01] *Texas Instruments*. "TMS320LF/LC240xA *DSP Controllers Reference Guide: System and peripherals*". 2001.

[NTC 5000, 02] Norma Técnica Colombiana 5000: "Calidad de la potencia eléctrica (CPE). Definiciones y términos fundamentales", Instituto Colombiano de Normas Técnicas (ICONTEC), 2002.

[IEEE 1159, 95] IEEE Standards coordinating committee 22 on power quality, USA. "IEEE Std 1159-1995: IEEE Recommended practice for monitoring electric power quality", IEEE Standards boards, 1995.

[IEC 6 1000-4-7, 91] COMISIÓN ELECTROTÉCNICA INTERNACIONAL. IEC 61 000-4-7. "Compatibilidad Electromagnética (EMC)-Parte 4: Técnicas de ensayo y medida. Sección 7: Guía general relativa a las medidas de armónicos e interarmónicos, asÍ como a los aparatos de medida, aplicable a las redes de alimentación y a los aparatos conectados a éstas". 32 p., Ginebra, Suiza, 1991.

[IEEE 519, 92] IEEE Std. 519. "*Recommended Practices and Requirements for Harmonic Control in Electric Power Systems*". 100 p., New York, U.S.A, 1992.

[Martínez et al, 02] Martínez, Diego E.; Peña, Maicohol A.;Barandica, Asfur; Loaiza, Humberto. "Monitoreo y Registro de los Parámetros de

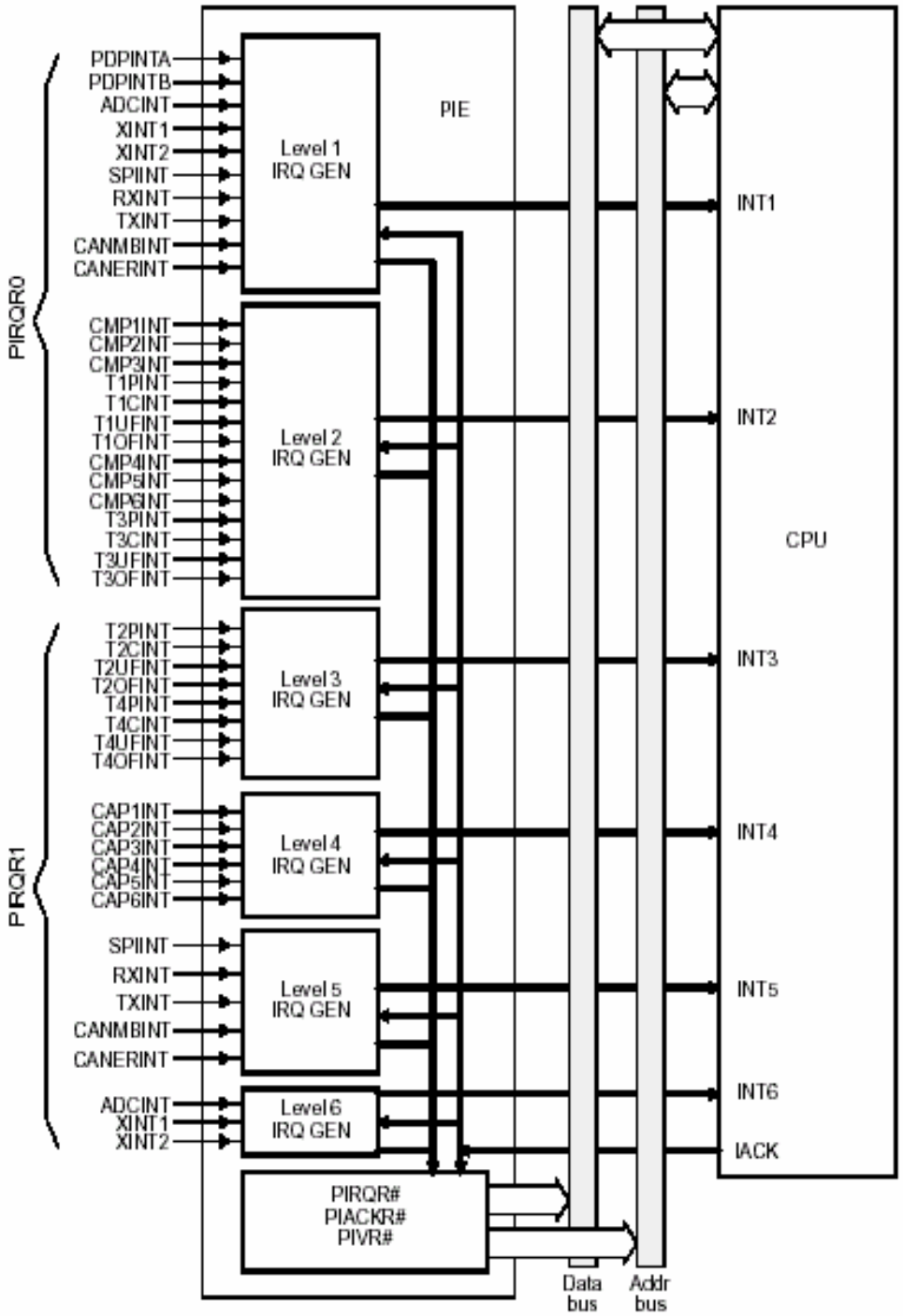
Calidad de Energía de una Red Trifásica con el DSP TMS320VC33". VII Simposio de tratamiento de señales, imágenes y visón artificial, páginas: 202-207. Bucaramanga, 2003

[Lakshmikanth & Morcos, 01] Lakshmikanth, A. y Morcos, Medhat. "*A Power Quality Monitoring System: A Case Study in DSP-Based Solutions for Power Electronics*". *IEEE Transactions on Instrumentation and Measurement*, Vol. 50, No. 3, paginas: 724-731, Junio 2001.

[Ruiz et al, 95] Ruiz, J.; Ortuondo, J.; Palacios, N:; Izquierdo, J.; Leturiondo, L.A.; Aramendi, E.; Amantegui, Javier. "*Real Time Power Quality Measurement and Monitoring Multichannel System*". IEEE *Transactions on Power Delivery*, Vol: 10, No. 3, paginas: 1190-1199, Julio 1995.

ANEXO A. DIAGRAMA DE BLOQUES DE LA EXPANSIÓN DE INTERRUPCIONES PERIFÉRICAS

Figura 47. Diagrama de bloques de la expansión de interrupciones periféricas.



ANEXO B. REGISTROS UTILIZADOS EN LA IMPLEMENTACIÓN

Tabla 9. Registros del LF2407A utilizados en la implementación

NOMBRE DEL REGISTRO	NEMOTCNIA DEL REGISTRO	DIRECCIÓN EN LA MEMORIA DE DATOS
Registro de enmascaramiento de las interrupciones de nivel	**IMR**	**0004h**
Registro de control y estados del sistema 1	SCSR1	7018h
Registro de *reset* del *watchdog*	WDKEY	7025h
Registro de control del *watchdog*	WDCR	7029h
Registro de control de comunicación *SCI*	SCICCR	7050h
Registro de control 1 del *SCI*	SCICTL1	7051h
Registro de selección de baudios del *SCI*, parte alta.	SCIHBAUD	7052h
Registro de selección de baudios del *SCI*, parte baja.	SCILBAUD	7053h
Registro de control 2 del *SCI*	SCICTL2	7054h
Registro del *buffer* de los datos del receptor del *SCI*	SCIRXBUF	7057h
Registro de control de prioridad del SCI	SCIPRI	705Fh
Registro de control A de multiplexación de entradas y salidas	MCRA	7090h
Registro de control 1 del *ADC*	ADCTRL1	70A0h
Registro de control 2 del *ADC*	ADCTRL2	70A1h
Registro de selección de la cantidad de conversiones del *ADC*	MAXCONV	70A2h
Registro de control de la secuencia de selección de los canales a muestrear por el *ADC*	CHSELSEQ1	70A3h
Registro del buffer de los resultados del canal uno del *ADC*	RESULT0	70A8h
Calibración del *ADC*	CALIBRATION	70B8h
Registro del período de conteo del temporizador 2	T2PR	7407h
Registro de control del temporizador 2	T2CON	7408h
Registro A de enmascaramiento de interrupciones del manejador de eventos A	EVAIMRA	742Ch
Registro A de las banderas de interrupciones del manejador de eventos A	EVAIFRA	742Fh

Registro B de las banderas de interrupciones del manejador de eventos A	EVAIFRB	7430h
Registro C de las banderas de interrupciones del manejador de eventos A	EVAIFRC	7431h
Registro A de las banderas de interrupciones del manejador de eventos B	EVBIFRA	752Fh
Registro B de las banderas de interrupciones del manejador de eventos B	EVBIFRB	7530h
Registro C de las banderas de interrupciones del manejador de eventos B	EVBIFRC	7531h
Registro generador de estados de espera	WSGR	I/O-FFFFh

ANEXO C. DESARROLLO DE UN PROYECTO UTILIZANDO EL CODE COMPOSER STUDIO

Los pasos para desarrollar una aplicación con el módulo de evaluación TMS320LF2407 EVM, a través del software *Code Composer Studio* son los siguientes:

- Crear un archivo enlazador *(*.cmd*) encargado de controlar la configuración de la memoria, la definición de secciones de salida y el direccionamiento de las secciones de salida en la memoria. Para este archivo se puede utilizar la plantilla que se muestra en la Figura 48. Si no se esta programando en lenguaje C la sección *.cinit* no es necesaria. Si no se esta desarrollando una aplicación con emulación en tiempo real las secciones *mon_pge0, mon_rgst* y *mon_main* no son necesarias.

Figura 48. Plantilla para el archivo enlazador.

```
MEMORY
{
        VECS:           org=0h,       len=40h          ] Memoria de programa
        EXT_PROG:       org=0044h     len=0FDBCh       ]

        PAGE 1:
        B2:             org=60h,      len=20h          ]
                                                       ] Memoria de
        B1:             org=300h,     len=100h         ] Datos
        B0:             org=200h,     len=100h         ]
        ExtRam   :      origin = 0x8000, length = 0x7fff ]
}
SECTIONS
{
        .text   :  {} >       EXT_PROG      PAGE 0  ── Sección para el código del programa
        .cinit   >       EXT_PROG      PAGE 0  ── Sección para las tablas de inicio de variables
        .switch  >       EXT_PROG      PAGE 0  ── Sección para las sentencias switch

        .const   >       B0            PAGE 1  ── Sección para constantes
        .bss     :  {} > B0            PAGE 1  ── Sección para las variables
        .stack   >       B1            PAGE 1  ── Sección para la pila
        .sysmem  >       B1            PAGE 1  ── Sección para la memoria dinámica del sistema.
                                                  Funciones malloc() y calloc() .
         vectors >       VECS          PAGE 0  ──── Sección para la tabla de vectores
        .data    :  {} >      B0        PAGE 1
        FFTipcb ALIGN(4096): { } >     ExtRam       PAGE 1  ]
        FFTmag    >      ExtRam        PAGE 1               ] Secciones creadas
        FFTwin : load=EXT_PROG PAGE 0,  RUN=ExtRam PAGE 1   ] por el usuario
        FFTtf  : load=EXT_PROG PAGE 0,  RUN=ExtRam PAGE 1   ]

    /* Additional sections for real time monitor */

        mon_pge0   : { } > B2     PAGE 1            /*  ]
        mon_rgst   : { } > B2     PAGE 1            /*  ] Secciones usadas por el
        mon_main   : { } > EXT_PROG  PAGE 0       /*  ] monitor en tiempo real
}
```

- Crear un archivo fuente en el cual se encuentra los algoritmos programados ha ser ejecutados por el *DSP*. Estos archivos pueden ser de dos clases: cuando se programa en lenguaje C es de extensión **.c*; y cuando se programa en lenguaje *assembly* es de extensión **.asm*.
- Agregar archivos de encabezado, con extensión **.h*, en donde se encuentran definidas constantes, etiquetas y se declaran las funciones que están definidas en librerías. Estos archivos de encabezado deben estar elaborados en el mismo lenguaje de programación del archivo fuente.
- Agregar un archivo de encabezado si se está programando en lenguaje C o un archivo fuente si se está programando en lenguaje *assembly*; donde se define la tabla de los vectores de interrupción. Para ver un ejemplo de esta tabla vaya a la dirección *C:\tic2xx\c2000 \MONITOR* o a *C:\tic2xx\c2000\tutorial\realtime*. Estas carpetas aparecen en el computador cuando se instala el *Code Composer Studio*.
- Agregar las librerías en donde se definen las opciones que se están usando en el algoritmo. Si se esta programando en lenguaje C, es obligatorio incluir la librería de soporte de *run-time rts2xx.lib* para que inicialice el entorno de programación en C.
- Configurar las opciones del enlazador, compilador y ensamblador. Las opciones a configurara son: La opción –i del compilador en donde se ubican las direcciones de búsqueda (separadas con punto y coma) de los archivos de encabezado que se han incluido; la opción –i del enlazador donde se ubican la direcciones de búsqueda (separadas con punto y coma) de las librerías pertenecientes al proyecto; la opción –l del enlazador donde se escriben el nombre de las librería utilizadas (separadas con punto y coma); y las opciones –o y –m en donde se colocan el nombre del archivo de salida a cargar en el DSP y el nombre del archivo del mapa de

memoria que se configuró respectivamente. Es recomendable dejar las demás opciones que aparecen por defecto para los proyectos. Si se desean cambiar o seleccionar otras opciones, es recomendable consultar los capítulos 3 y 8 de la referencia [T.I spru018d, 95] para entender las opciones del ensamblador y del enlazador respectivamente y la referencia [T.I spru024e, 99] para las opciones del compilador.

- Un archivo **.gel* en el cual se le indica al depurador del *Code Composer* cuáles áreas de memoria el puede acceder o no. Si se está programando el *DSP* para que funcione en modo microprocesador (*JP6* en la posición 1-2), este archivo es cargado apenas se inicia el *Code Composer Studio* (archivo *init.gel*). Si el *DSP* va a funcionar en modo *stand-alone* o microcomputador (*JP6* en la posición 2-3) se selecciona la opción *F2407_ROM_memory_map()* del archivo *init.gel* para que configure el respectivo mapa de memoria.
- Si el proyecto va a permitir la emulación en tiempo real deben agregarse dos archivos: el archivo *c200mnt.i* donde se encuentra la información para la configuración del monitor en tiempo real; y el archivo *c200mnrt.asm* encargado del funcionamiento del monitor en tiempo real.

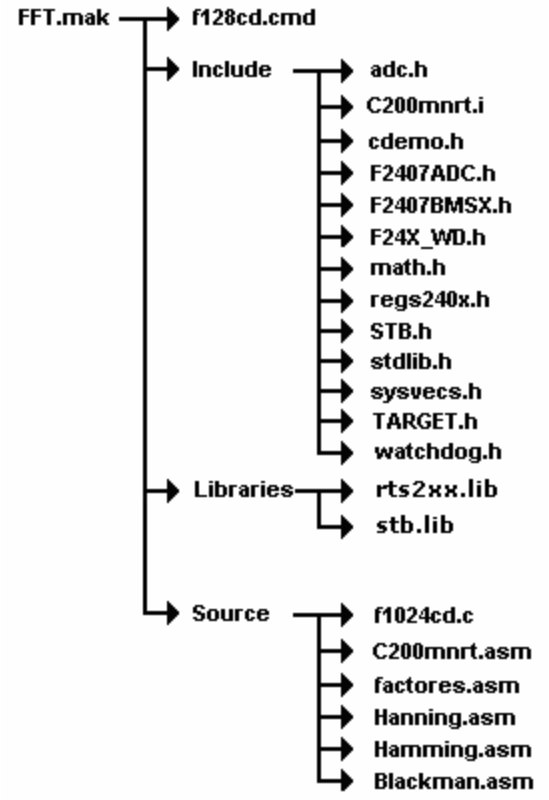
Después de desarrollar un proyecto, con el *DSP* en modo microprocesador, en el menú *Option->Program Load...* de la ventana principal del *Code Composer Studio* seleccione la opción *Load Program After Build*. De esta forma, después de que se compila el programa, esté es cargado en el *DSP*. En este punto ya puede ejecutar el programa desde el menú *Debug->Run*. Si el proyecto posee la característica de emulación en tiempo real debe: ejecutar el programa; luego detenerlo desde el menú *Debug->Halt*; después reiniciar el DSP con el menú *Debug->Reset*

DSP; seleccionar el modo en tiempo real en el menú *Debug->Real Time Mode*; y ahora si ejecutar el programa.

Si el proyecto se desarrollo para ejecutarse en modo de microcomputador, primero debe instalarse un *plug* en el *Code Composer Studio* que permite la programación de la memoria *Flash* interna de programa que posee el *DSP*. Este *plug* se encuentra en el sitio *web* de *Texas Instruments* y su nombre es *API 1.3*.

En la Figura 49 se encuentra el diagrama del proyecto implementado para el prototipo.

Figura 49. Proyecto implementado para el prototipo



ANEXO D. SINCRONÍA CON LA FRECUENCIA DE 60 HZ

Los resultados de la Tabla 10 se calcularon para adquirir 128 muestras por ciclo con el reloj del DSP LF2407A en 40MHz. Estos datos se determinaron para saber cuantas opciones de sincronía permite el LF2407A en el rango de 59.8 Hz a 60.2 Hz que puede variar la frecuencia del sistema en condiciones normales de operación (Resolución CREG 025 de 1995).

Tabla 10. Sincronía con la frecuencia de 60 Hz.

T2PR	Frecuencia de muestreo (Hz)	Fsistema [Hz]	Deslizamiento
5191	7705,644385	60,20034675	0,3328
5192	7704,160247	60,18875193	0,3136
5193	7702,67668	60,17716156	0,2944
5194	7701,193685	60,16557566	0,2752
5195	7699,711261	60,15399423	0,256
5196	7698,229407	60,14241724	0,2368
5197	7696,748124	60,13084472	0,2176
5198	7695,267411	60,11927664	0,1984
5199	7693,787267	60,10771302	0,1792
5200	7692,307692	60,09615385	0,16
5201	7690,828687	60,08459912	0,1408
5202	7689,35025	60,07304883	0,1216
5203	7687,872381	60,06150298	0,1024
5204	7686,395081	60,04996157	0,0832
5205	7684,918348	60,03842459	0,064
5206	7683,442182	60,02689205	0,0448
5207	7681,966583	60,01536393	0,0256
5208	7680,491551	60,00384025	0,0064
5209	7679,017086	59,99232098	-0,0128
5210	7677,543186	59,98080614	-0,032
5211	7676,069852	59,96929572	-0,0512
5212	7674,597084	59,95778972	-0,0704
5213	7673,12488	59,94628813	-0,0896

5214	7671,653241	59,93479095	-0,1088
5215	7670,182167	59,92329818	-0,128
5216	7668,711656	59,91180982	-0,1472
5217	7667,24171	59,90032586	-0,1664
5218	7665,772327	59,8888463	-0,1856
5219	7664,303506	59,87737114	-0,2048
5220	7662,835249	59,86590038	-0,224
5221	7661,367554	59,85443402	-0,2432
5222	7659,900421	59,84297204	-0,2624
5223	7658,43385	59,83151446	-0,2816
5224	7656,967841	59,82006126	-0,3008
5225	7655,502392	59,80861244	-0,32
5226	7654,037505	59,79716801	-0,3392