DISEÑO Y CONSTRUCCIÓN DE UN PROTOTIPO DE UN MEDIDOR DE ARMÓNICOS DE CORRIENTE BASADO EN UN PROCESADOR DE SEÑALES DIGITALES (*DSP*)

AUTORES:

JAIRO IVÁN FLÓREZ BARRERA

SHIRLEY PAOLA HERRERA HERNÁNDEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERIAS FÍSICO-MECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

BUCARAMANGA

2004

DISEÑO Y CONSTRUCCIÓN DE UN PROTOTIPO DE UN MEDIDOR DE ARMÓNICOS DE CORRIENTE BASADO EN UN PROCESADOR DE SEÑALES DIGITALES (*DSP*)

AUTORES:

JAIRO IVÁN FLÓREZ BARRERA

SHIRLEY PAOLA HERRERA HERNÁNDEZ

PROYECTO DE GRADO

DIRECTORES:

Msc. JORGE RAMÓN

MPE. CÉSAR DUARTE GUALDRÓN

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERIAS FÍSICO-MECÁNICAS

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

BUCARAMANGA

2004

DEDICATORIA

A mi familia, por supuesto.

Jairo Iván Flórez Barrera

Solo hay en el mundo dos personas a quienes puedo dedicarles este trabajo; indudablemente mis padres.

Shirley Paola Herrera Hernández

AGRADECIMIENTOS

En primera instancia agradecemos a Dios por permitirnos culminar ésta etapa.

Por otro lado agradecemos especialmente a nuestros directores de proyecto; los profesores César Antonio Duarte Gualdrón y Jorge H. Ramón Súarez; además a los profesores Gabriel Ordóñez, Jaime Barrero, Alfredo Acevedo, Oscar Gualdrón y al ingeniero Javier Mier a todos ellos muchas gracias por su tiempo, su conocimiento, su orientación y en especial su voluntad.

A nuestras familias por su apoyo incondicional, su paciencia y por hacer de nosotros lo que hoy somos.

A nuestros más sinceros amigos porque contamos con ellos siempre; por su tiempo, su experiencia, su comprensión y sus palabras de aliento, que reafirmaron nuestra amistad.

Jairo y Shirley

A mi amigo y compañero de batallas, Jairo Iván Flórez, por tenerme tanta paciencia, por ayudarme a entender y resolver tantas cosas, por mostrarme mis errores, por su confianza y finalmente por su sincera amistad.

A Dany por distraerme y alegrarme tanto, en el mejor momento.

Shirley Herrera

TABLA DE CONTENIDO

Pág.

TABLA DE FIGURAS

Pág.

TABLA DE TABLAS

Pág.

TABLA DE ANEXOS

INTRODUCCIÓN

El procesamiento de señales digitales es una de las herramientas de mayor interés en ingeniería, pues se adapta a la tecnología digital que presentan los sistemas electrónicos; brindando rapidez, exactitud, baja probabilidad de error y adaptabilidad para trabajar con cualquier señal analógica luego de ser digitalizada.

Hoy en día se cuenta con nuevas tecnologías para ejecutar este trabajo, mediante dispositivos integrados que se encargan de realizar la conversión de las señales analógicas adquiridas a digitales, su procesamiento y el almacenamiento de resultados; estos dispositivos llamados *DSPs (Digital Signal Processors)* son la base de este proyecto**.**

El principal objetivo en este trabajo es, implementar un prototipo que estime los armónicos de corriente, presentes en un sistema monofásico cuando se trabaja con cargas no lineales, para lo cual se ha seleccionado el *DSP56F801* fabricado por Motorola. La adquisición de la señal se hará por medio de sensores de efecto *hall* que proporcionan una tensión correspondiente al campo magnético que genera el paso de una corriente; un circuito de adecuación de la señal adquirida que la llevará al nivel necesario para la entrada del *DSP* y finalmente la visualización se obtendrá con una pantalla gráfica que permitirá ver el espectro en frecuencia de la señal eléctrica tomada en tiempo real y los valores de los armónicos medidos en la etapa previa.

1. FUNDAMENTOS TEÓRICOS PARA LA IMPLEMENTACIÓN DEL PROYECTO

En este capítulo se exponen los diferentes temas relacionados con este proyecto involucrando la base teórica del efecto *hall*, las técnicas utilizadas en el procesamiento de señales para estimar armónicos y las especificaciones del *hardware* empleado, haciendo alusión propiamente al ***DSP56f801***, sus características más significativas en cuanto a memoria, manejo de periféricos, arquitectura y *software*, asimismo se consideran las características de la pinza **SONDA AEMC MN375** que es utilizada en el sensado de la señal de corriente, finalmente se concluye con conceptos relacionados con la pantalla gráfica **LCD[1] HYUNDAI HG25601-C**.

1.1 EFECTO *HALL*

El efecto *hall* fue descubierto por Edwin Herbert Hall in 1879; este efecto permite conocer el valor de la corriente que circula por un conductor o la tensión que hay en sus terminales gracias al flujo magnético que se genera por esta corriente [GRANADOS, et al, 99], ó por la acción de un campo magnético incidente perpendicularmente sobre un conductor. Depende del tipo de sensor con que se trabaje, y del tipo de señal que se requiere sensar, sea corriente o tensión.

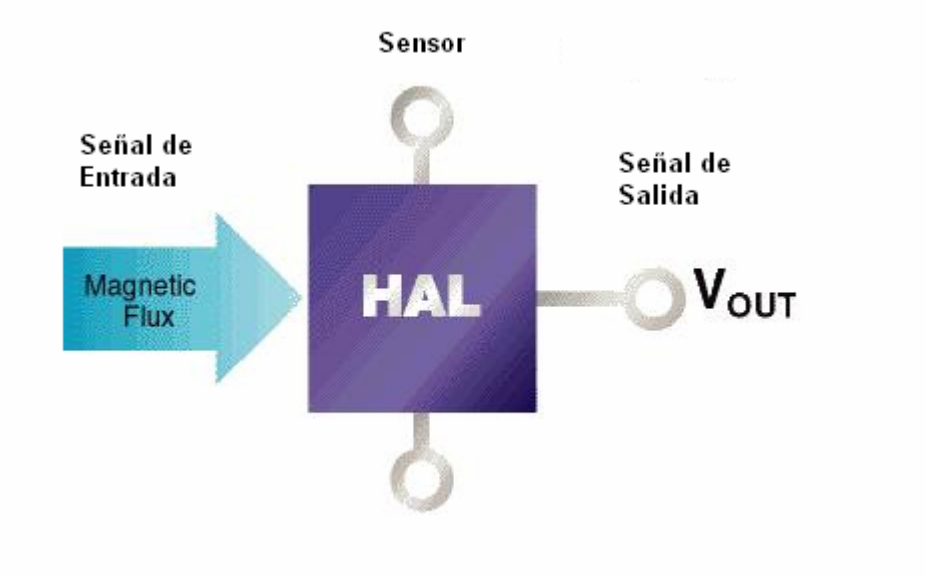
En primera instancia al trabajar con el principio de flujo magnético producido, se pretende medir la tensión presente en los terminales de un conductor, para esto es necesario un sensor de circuito integrado, el cual es puesto en el entrehierro de un núcleo de ferrita, que posee un arrollamiento con un

[1] Lyquid Cristal Digital

número determinado de vueltas. A los terminales de este bobinado se conectan los terminales del conductor y así se estima su campo eléctrico; el fenómeno consiste en que la corriente que circula por el conductor es llevada a la bobina produciendo entonces un flujo magnético que atraviesa perpendicularmente el área sensora del elemento; la tarea del integrado es detectar y procesar este flujo magnético para entregar una señal eléctrica llamada tensión *hall*, que tiene la misma forma de onda de la corriente y es proporcional a ella en magnitud, sin embargo la proporcionalidad depende de cada sensor.

Se observa en la Figura 1, en esencia el uso del sensor, el flujo magnético se presenta como la entrada al mismo y una tensión a la salida, que es proporcional a la corriente que genera este flujo, es la salida del conversor.

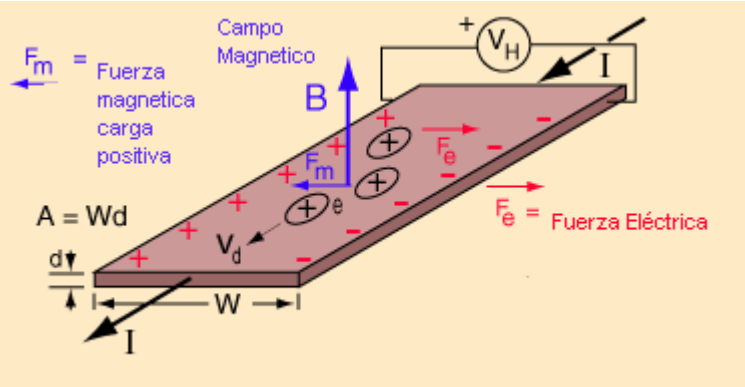
Figura 1. Diagrama esquemático del sensor de efecto *hall*



Ahora, si se enfoca la atención en el segundo tipo de sensado, se debe tener en cuenta que este patrón es idóneo para sensar corriente por lo cual fue el

empleado en este proyecto, el principio es como sigue: si se hace circular una corriente por un conductor y éste se somete a la acción de un campo magnético perpendicular a la dirección de la corriente, la fuerza electromotriz (f*em*) que se deriva logra hacer que las cargas positivas y negativas de la corriente se agrupen de tal forma, que se genera una diferencia de potencial, básicamente este es un campo eléctrico llamado tensión *hall* (esto se puede observar en la Figura 2), así trabaja la Pinza SONDA AEMC MN375, después de detectar el campo magnético generado por la corriente que pasa a través del conductor y que es perpendicular a la dirección de la corriente de sus placas sensoras, esta exhibe en sus terminales el campo eléctrico (tensión *hall)* suscitado, que es proporcional al valor de corriente que generó este campo; la forma de onda de la tensión es exactamente la misma forma de onda de la corriente, cabe aclarar de nuevo que la proporcionalidad de la tensión de salida con respecto a la corriente depende de las características del sensor.

Figura 2. Efecto *hall*



1.1.1 Pinza SONDA AEMC MN375. (Figura 3)

El sensor de la pinza tiene las siguientes especificaciones**:**

• Rango nominal de 10A

• Rango de medida de 0.1A a 10A

• Rango de salida 100mV/A. (1Vrms equivale a 10A. Relación lineal entre la tensión de salida y la corriente que circula por el conductor.)

• Precisión para el rango de 10A

  • [0,1- 1] A 1% ó +- 2mV

  • [1- 10] A 1% ó +- 1,5mV

• Cambio de Fase

  • [0,1- 1] A no especificada

  • [1- 5] A menor a 1° @ 60Hz

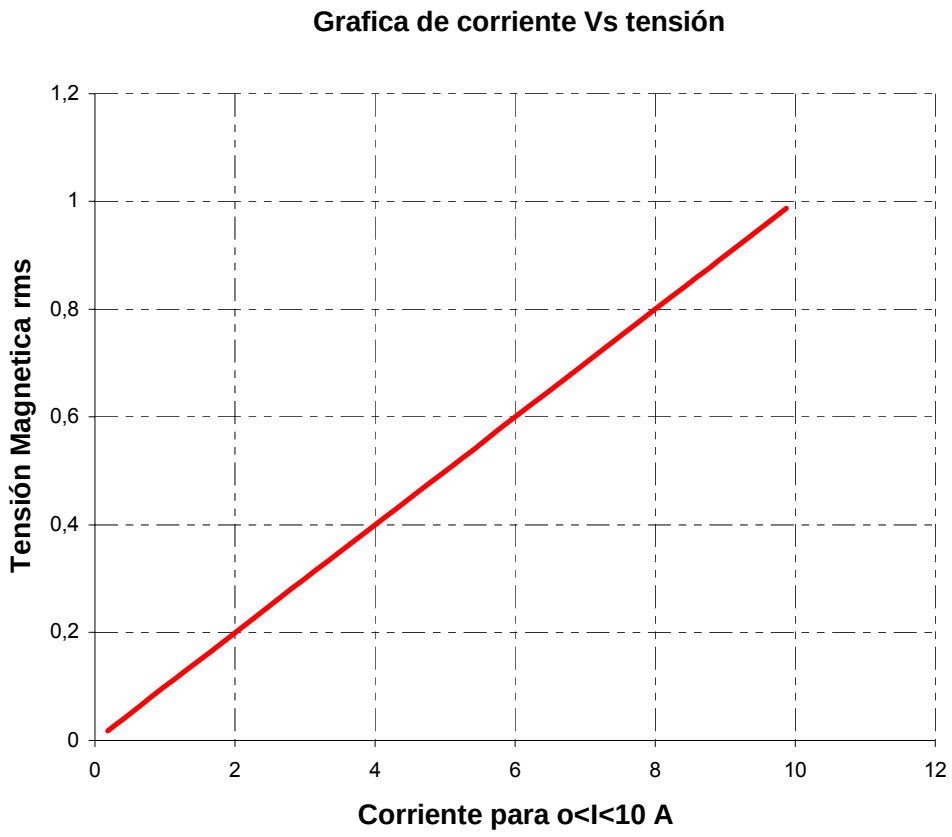
  • [5-10] A menor a 1,5° @ 60Hz

• Rango de frecuencia: 40Hz a 3KHz

Figura 3. SONDA AEMC MN375

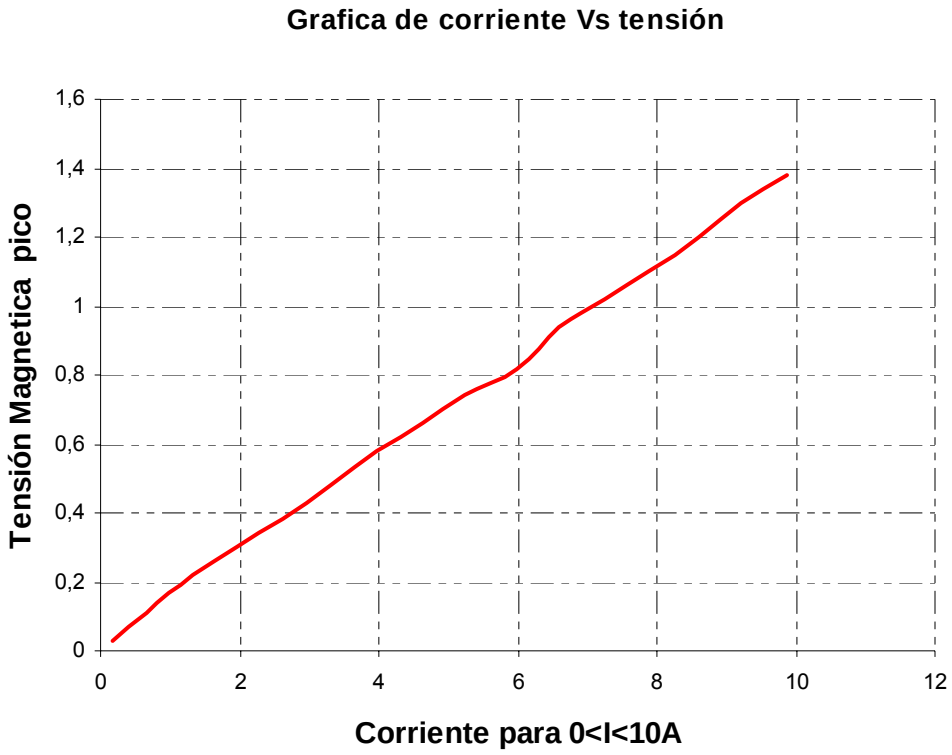
Tomado de la hoja de datos AEMC MN375 que se encuentra en el anexo C.

Figura 4. Curva característica para corriente en el conductor entre 1 y 10A Vs tensión eficaz de salida. (Pruebas realizadas en el laboratorio UIS)



Como se puede observar en la figura 4 la respuesta de la pinza es lineal al trabajar con la tensión eficaz de salida, sin embargo al trabajar con la tensión pico, la línea no es tan recta como se esperaba, para solucionar este inconveniente se debe obtener la ecuación característica de la curva, y así lograr una aproximación lineal, luego en la programación en el *DSP* se hacen los cambios pertinentes, asimismo se debe tener en cuenta el error que pueda introducir esta aproximación en la estimación. La figura 5 es la curva corriente Vs tensión pico de salida antes mencionada.

Figura 5. Curva característica para corriente en el conductor entre 1 y 10A Vs tensión pico de entrada. (Pruebas realizadas en el laboratorio UIS)



1.2 MEDICIÓN DE ARMONICOS

La medición de los armónicos presentes en los sistemas de energía es un aspecto de gran relevancia en la calidad de la energía eléctrica ya que determina en muchos casos el funcionamiento adecuado de las redes y los equipos conectados. Este problema se debe afrontar no solo desde el punto de suministro de energía sino también desde el lado del cliente quien al conectar cargas no lineales introduce armónicos al sistema.

Para realizar la medición de armónicos se deben tener en cuenta dos aspectos fundamentales, la variación de la frecuencia de la onda de corriente y el valor eficaz de tensión y corriente del sistema eléctrico. “Una señal eléctrica se puede describir en estado estacionario como la sumatoria de armónicos con frecuencias múltiplo entero de la frecuencia fundamental” [Duarte, Ordoñez, 2003], pero además existe un desplazamiento de frecuencia (δ) introducido en el equipo de medida por el error que existe al hacer la medición de frecuencia del sistema o al trabajar con la frecuencia nominal, que no siempre es exactamente 60Hz, según esto la señal se define como:

$$x(t) = A_0 + \sum_{k=1}^{N} A_k Cos\left(k 2\pi f_m (1-\delta) t + \theta_k\right) \quad (1)$$

Al calcular los parámetros de la señal eléctrica para la medición de los armónicos normalmente los equipos utilizan algoritmos como la *DFT*[2][IEC 61000-4-7, 91], entonces si se considera f_m como la frecuencia medida por el equipo, M el número de muestras por ciclo, N es un número entero menor que $M/(2(1-\delta))$. Y se tiene en cuenta el efecto anti-solapamiento[3] el modelo de las muestras se describe así:

$$x[n] = A_0 + \sum_{k=1}^{N} A_k Cos\left(k \frac{2\pi}{M} (1-\delta) n + \theta_k \right) \quad (2)$$

Cuando se realizan los cálculos en presencia del desplazamiento de frecuencia la estimación puede ser errónea ya que podrían no eliminarse correctamente las componentes armónicas que no hacen parte de la medida y además la componente que se está estimando podría no filtrarse con ganancia máxima; para solucionar estos inconvenientes se deben adaptar los

[2] *Discre Fourier Transfor* (Transformada Discreta de *Fourier*)
[3] La frecuencia de muestreo debe ser por lo menos dos veces la frecuencia más alta de la señal para evitar solapamiento de componentes de frecuencia.

algoritmos utilizando por ejemplo ventanas de ponderación, desarrollando algoritmos más robustos, más exactos pero por tal motivo más lentos; asimismo las diferentes metodologías de procesamiento de la *DFT* agregan demoras en el tiempo de ejecución dependiendo de la exactitud requerida. Es por esto que es necesario buscar un tipo de procesador rápido y si es posible con procesamiento en paralelo que agilice el proceso, y de esta forma mantener la medida en tiempo real si se busca diseñar un prototipo independiente del computador.

1.2.1 Transformada Discreta de Fourier.

La Transformada Discreta de Fourier (DFT) es un algoritmo utilizado en el procesamiento de señales, cuando se requiere estimar las componentes de frecuencia de un sistema partiendo de sus muestras, además se puede hallar el valor eficaz, evaluar armónicos, interarmónicos, subarmónicos, entre otros.

Básicamente la *DFT* parte del siguiente principio: *L* muestras de un señal se pueden representar mediante la suma de *L* componentes de frecuencia que sean exponenciales complejas y de la siguiente forma: $Be^{j(wn+\beta)}$. La tarea de la *DFT* consiste en calcular la magnitud y la fase de tales componentes haciendo uso de la siguiente expresión: [Duarte, 2004]

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

Ahora la reconstrucción de la señal partiendo de sus *L* muestras se realiza por medio de la siguiente ecuación de síntesis:

$$x[n] = \frac{1}{L}\sum_{k=0}^{L-1}\left[F[k]e^{jk\frac{2\pi}{L}n} \right] \quad para\ 0 \le n \le L-1 \quad (4)$$

Es posible concluir que si la señal es periódica y las *L* muestras utilizadas para calcular las componentes de frecuencia de la *DFT* corresponden a un

periodo fundamental de la señal, estas serán las componentes armónicas de la señal.

1.2.2 Transformada Rápida de Fourier.

Transformada Rápida de Fourier FFT4 calcula exactamente los mismos valores que la DFT pero de forma más eficiente ya que no realiza L2 multiplicaciones sino (L/2)*Log2(L) [Proakis, Manolakis, 98], pero por tal motivo el algoritmo se vuelve más complejo necesitando entonces un procesador ágil. Por otro lado el número L de muestras debe ser una potencia entera de dos, es decir (L=2v siendo v un entero). [Proakis, 98].

1.3 PROCESADORES DE SEÑALES DIGITALES

El procesamiento de señales digitales ha incrementado su utilización debido a la rapidez de respuesta y a la adaptabilidad para trabajar algoritmos dedicados al procesamiento de señales, es importante resaltar la versatilidad que se obtiene al trabajar con procesadores de señales digitales pues con un simple convertidor analógico/digital[5] se puede tratar cualquier tipo de señal analógica logrando resultados adecuados en corto tiempo.

Para este trabajo se cuenta con el *DSP56F801* fabricado por Motorola y que pertenece a la Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones, este *DSP* es de los de menor jerarquía en su familia,

---

[4] Fast Fourier Transform

[5] En las siguientes páginas se hará referencia al Conversor con la abreviatura *ADC(Analog-Digital Conversor*)

sin embargo posee una arquitectura suficiente para el trabajo que se pretende realizar; en realidad se puede describir como un híbrido entre un microcontrolador y un procesador de señales digitales tomando del primero la capacidad en el manejo de periféricos y del segundo la velocidad para procesar señales digitales. Es importante resaltar la facilidad de adaptación que posee el *DSP* para trabajar sin su *demoboard*[6]*;* es decir, la facilidad para independizarlo y construir una aplicación específica, a diferencia de otros *DSPs* que existen en el mercado y que no brindan esta alternativa, sin embargo en este prototipo se trabaja el *chip* incorporado a su *demoboard* y así se aprovecha la estructura y las ventajas que conlleva tener el *chip* listo para procesar. Una de las recomendaciones que se realizan en este proyecto, es precisamente trabajar con el integrado independientemente.

Cabe mencionar que esta *demoboard* fue muy importante para el proyecto (a pesar de no poseer memoria de almacenamiento externa), pues contaba con la implementación en *hardware* de los periféricos de entrada y salida del *DSP*, necesarios para la adquisición de datos y la visualización de resultados. Una descripción más específica de esta *demoboard* se encuentra en la referencia [Motorola, Dsp56f800dbum, 2003].

El *DSP56F801* esta construido según la arquitectura *HARVARD* que permite trabajar independientemente la memoria de datos y la memoria de programas, agilizando su respuesta [Motorola, into 56F801 Arquitecture], por otro lado cuenta con dos canales *ADC* de cuatro entradas operando como independientes o diferenciales[Motorola, DSP56F801 User Manuals, 2003] [Motorola, AN1936/D (pdf), 2003]. A continuación se presenta un resumen de la arquitectura, la memoria, periféricos y las características de *software* más significativas.

---

[6] Tarjeta de prueba.

1.3.1 Arquitectura del chip.

La arquitectura HARVARD, separa físicamente la memoria de programas y de datos, tanto en memoria externa como memoria interna (este DSP no posee memoria externa al chip). Dentro de las características más relevantes del chip se tienen:

- Seis canales para comunicar con el modulo de *PWM*[7]
- Interfase serial de comunicación (*SCI*)
- Interfase serial de periféricos. (*SPI*)
- Dos canales analógicos/digitales de cuatro conversores cada uno (12 bits)
- Cuatro temporizadores de propósito general
- Puerto JTAG/OnCE™ para *debuger*[8]
- Un oscilador *On-chip* de relajación
- Puerto *GPIO* (propósito general) con 11 pines
- En total 48 pines.

1.3.2 Arquitectura del núcleo del procesador[9].

Componentes como la unidad aritmético lógica, la unidad generadora de señales, la unidad controladora de programas y la unidad de manipulación de "bus" y bits, entre otras, hacen parte del núcleo del procesador. Por otra parte la frecuencia del procesador en MIPS10 es la mitad de la de la frecuencia del reloj, es decir que trabaja máximo a 40MIPS si se tiene una frecuencia de reloj de 80MHz (En este caso se trabaja a 30MIPS). Posee además una serie de registros y buses de operación, almacenamiento y direccionamiento que se describen a continuación:

---

[7] Pulse width modulator (Modulación por el ancho del pulso)
[8] Seguimiento de la señal
[9] Núcleo del *Chip* DSPF56801
[10] MIPS: millones de instrucciones por segundo

- Dos acumuladores de 36 bits en total para almacenar datos (A y B)
- Tres registros de entrada de 16 bits (X0, Y0 y Y1)
- Cuatro registros de 16 bits para direccionamiento (R0-R3)
- Un registro de pila (*SP*)
- Un registro de estados (*SR*)
- Un registro para modo de operación (*OMR*)
- Barril bidireccional de desplazamiento de 16 bits
- Acumulador multiplicador paralelo de ciclo sencillo 16x16-bits (*MAC*)
- Un bus externo y cuatro buses internos para transferencia de datos
- Un bus externo y tres buses internos para direccionamiento
- 15 modos de direccionamiento

1.3.3 Memoria.

La Arquitectura HARVARD del núcleo del procesador permite hasta tres accesos simultáneos de datos y programas a la memoria, lo cual resulta muy conveniente para las necesidades de programación, pues se obtiene una mayor velocidad en el procesamiento.

Por otra parte, la *demoboard* con la que trabaja el *DSP56F801* no posee memoria, es decir solo cuenta con la memoria *on-chip*, sin embargo es importante mencionar que posee dos tipos de memoria, la memoria *flash* y la memoria RAM, las dos divididas en memoria de programas y de datos debido a su arquitectura, la memoria *flash* se encuentra en el núcleo del procesador.

- Memoria *Flash*: 8Kx16-bits de palabra para programas.
  2Kx16-bits de palabra para datos.
  2Kx16 bits de palabra para *boot.*
- Memoria RAM: 1Kx16 bits de palabra para programas
  1Kx16 bits de palabra para datos.

1.3.4 Periféricos.

EL manejo de periféricos en este DSP es una de sus mayores ventajas. El DSP56F801 posee diferentes tipos de periféricos que se detallan a continuación, vale la pena mencionar que estos periféricos cuentan con puertos ya definidos en la demoboard del DSP.

- Modulación por ancho de pulso (*PWM*) con seis salidas y dos entradas por defecto.
- Conversor analógico/digital de 12 *bits* con dos canales de 4 entradas para operar multiplexadas o independientemente, el *ADC* y el *PWM* se pueden sincronizar. Éste controlador posee una frecuencia de muestreo determinada por el usuario (Máximo se puede muestrear a 5MHz según las especificaciones del fabricante); los niveles de entrada alto y bajo al conversor se encuentran entre 0 V y 3.3 V respectivamente.
- Cuatro temporizadores de propósito general, el temporizador D con tres pines o tres líneas adicionales para el puerto GPIO.
- Interfase serial de comunicación (*SCI*) con dos pines o dos líneas adicionales para el puerto GPIO.
- Interfase serial de periféricos (*SPI*) con 4 pines programables o 4 líneas adicionales para el puerto GPIO
- Puerto GPIO (propósito general) con 11 pines
- *COP(computing operating properly* temporizador *watchdog*
- Pin dedicado a interrupciones externas.
- Pin dedicado a reset externo.
- Puerto JTAG/On-Chip velocidad del procesador independiente del *debuger.*
- *Software* programable para el *PLL*[11] (reloj del *core*)

[11] PLL (phase lock loop)

Para más información sobre estos tópicos consultar las siguientes referencias bibliográficas [Motorola, DSP56F801FA60, 2003] [Motorola, DSP56f800 *Family Manuals*, 2003]

1.3.5 Características de programación del *DSP.*

Con respecto al modo de programación del *DSP* es importante resaltar el *SDK* (Software Development Kit). Esta herramienta de programación permite el desarrollo de aplicaciones específicas mediante bibliotecas y funciones que facilitan el uso del *DSP* dependiendo de la aplicación que se busca implementar. Las bibliotecas y funciones incluyen: manejo de periféricos (entradas/salidas), temporizadores, administración de memoria, rutinas del *DSP*, interrupciones, herramientas, entre otras. [Motorola, *view pdf*, 2003]

El *DSP* cuenta con el *CodeWarrior*; ambiente de trabajo o compilador en el cual se puede desarrollar la programación en lenguaje C o *assembler* para una determinada aplicación, y asimismo su compilación, depuramiento y almacenamiento en la memoria flash o *RAM* del chip. En el Apéndice A se realiza una descripción más detallada sobre el *CodeWarrior* y el SDK. Referencias [*Metrowerks*, IDE_5.1_*Users_Guide*, 2003] [*CodeWarrior* (PDF), 2003]

Se debe tener presente también que el *set* de instrucciones soporta tanto funciones del *DSP* como funciones de control, las cuales pueden ser programadas mediante lenguaje de alto nivel como el lenguaje C, pero también por medio de la programación de bajo nivel; *assembler*[12].

[12] Lenguaje de programación de bajo nivel

Otra ventaja que presenta este procesador es que los lazos de DO y REP13 están implementados en hardware directamente,[Motorola, DSP56f800 Family, 2003]. Esto incrementa la velocidad de respuesta del mismo.

1.4 VISUALIZACIÓN DE LOS RESULTADOS

La visualización de los resultados en la estimación de armónicos de corriente, requiere que sea lo suficientemente clara y específica, es decir, se necesita visualizar el valor de un armónico determinado hasta el 30^{avo} y el espectro en frecuencia para tener una buena apreciación y así el profesional encargado, pueda hacer un análisis consistente de la calidad del sistema eléctrico visto desde la perspectiva de la carga, este es el punto en el cual podrían aparecer armónicos de corriente, lo que conlleva a efectos nocivos en la calidad de la señal eléctrica que es consumida por el cliente; quien inyecta las cargas no lineales al sistema.

Para desarrollar esta etapa, se cuenta con la pantalla gráfica **LCD HYUNDAI HG25601-C** que pertenece a le Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Se adoptó esta solución para que prototipo sea totalmente independiente del computador, a continuación se presentan las características que posee.

[13] Instrucciones de programación de alto nivel.

1.4.1 Pantalla gráfica LCD HYUNDAI HG25601-C.

Esta pantalla trabaja con el circuito integrado controlador LDC SED1330F14 el cual presenta ciertas características y especificaciones que se expondrán a continuación.

1.4.2 Características y especificaciones.

Las características más sobresalientes de esta pantalla se contemplan a continuación:

- **Texto y gráficos**. Una de las mayores ventajas que se tienen al trabajar con esta pantalla gráfica es la facilidad para combinar texto y gráficos; pues posee tres capas de pantalla simultáneas que pueden organizarse sólo de la siguiente forma:
    - La primera capa de texto y la segunda de gráfico. En esta forma no se trabaja la tercera capa, y el texto solo se visualiza en la primera capa.
    - La primera, segunda y/o tercera capa de gráficos

La resolución que permite esta pantalla es de 640x256 *pixeles*[15] la cual es suficiente para una clara visualización.

- **Memoria.** La memoria con que cuenta el circuito integrado SED1330F es de tipo ROM y RAM. Ambas memorias permiten generar y almacenar caracteres en una aplicación específica. Asimismo, también puede generar caracteres internos que se almacenan en la memoria ROM, por otro lado resulta relevante mencionar que el *buffer* de la memoria RAM tiene una

---

[14] Fabricado por **S-MOS Systems, Inc.**

[15] Unidad mínima de información que se puede mostrar en pantalla, hace referencia a la resolución de la misma.

capacidad de almacenamiento de hasta 64Kbytes, esto le indica al programador con que capacidad puede transmitir.

- **Control del cursor.** El cursor puede moverse en cualquier dirección (arriba, abajo, izquierda o derecha), se programa utilizando lenguaje de programación C o C++.

- **Distribución de pines:** Cuenta con 60 *pines QFPs.* a continuación se presenta la distribución de los *pines* que se utilizaron en el prototipo:
  - 16 *pines* para direccionamiento del bus de memoria RAM
  - 1 *pin* para *RESET*
  - 8 *pines* para el bus de datos
  - 8 *pines* para el bus de datos de la RAM
  - Además de los *pines* utilizados para alimentación y tierra.
  - En la comunicación con el *DSP* se trabaja con 11 pines de los cuales 3 son para control y 8 para datos.

- **Consumo de energía.** El consumo de corriente del SED1330F es relativamente bajo, siendo este de 3,5mA si se alimenta con 3,5 V, sin embargo la alimentación para este integrado es de 5 V, si se hace referencia a la pantalla en si, ésta debe alimentarse con una señal de corriente continua entre -25 V y -10 V (para esta aplicación se alimentó con -12v)

La función que realiza la pantalla en este prototipo conjunto con lo concerniente a la programación e inicialización se describe detalladamente en el capítulo 4.

2. ADQUISICIÓN DE SEÑAL

En este capítulo se describe el *hardware* necesario para implementar la tarjeta de adquisición de los armónicos de corriente, partiendo del sensado por medio de la pinza, hasta llegar a la entrada del conversor analógico-digital, que hace parte del *DSP.* Se abordan cada una de las etapas de adquisición, adecuación, filtrado y alimentación que hacen parte de esta tarjeta, se mencionan y se resaltan algunos de los circuitos integrados utilizados (en los anexos se encuentra en la hoja de datos de cada uno de ellos) y se muestran gráficas de los diseños de los circuitos realizados tanto de su esquemático como del circuito real. En conclusión al abordar este capítulo el lector podrá conocer los pormenores de la implementación de la tarjeta de adquisición de una señal, en este caso la señal de los armónicos de corriente.

2.1 ADQUISICIÓN DE LOS ARMÓNICOS DE CORRIENTE

La estimación de los armónicos de corriente mediante el *DSP56F801* requiere que la señal sea acondicionada previamente, antes de ser procesada. Para esto se implementó una tarjeta de adquisición que estima la señal por medio de la Pinza SONDA MN 375, luego se ajusta el rango de tensión de salida de la pinza a un valor apropiado para el conversor, el cual se encuentra entre -0,1 V y 3,3 V. Además es necesario implementar un filtro antisolapamiento con una respuesta en frecuencia máximamente plana, cuyo desempeño se complementa mediante filtrado digital.

2.1.1 Armónicos de corriente.

La adquisición de los armónicos de corriente se realiza por medio de la Pinza SONDA MN 375 fabricada por AEMC. Este sensor utiliza el efecto hall para medir la corriente que pasa a través de un conductor, posee su propio núcleo, haciéndola suficientemente útil para los requerimientos de este proyecto, en el capítulo uno se presentó una imagen de la misma con sus respectivas características y especificaciones.

La relación entre la corriente y la tensión magnética que posee este sensor es de 1Vrms a 10A, según las indicaciones del fabricante. Las pruebas realizadas en el laboratorio lo confirman; por otro lado, como se mencionó en el capítulo anterior la corriente máxima que puede medir es de 10A, esto se debe tener en cuenta para no afectar el comportamiento del prototipo.

Luego de efectuar el sensado del campo magnético por medio de la pinza, es necesario tener en cuenta los requerimientos necesarios para trabajar con el *ADC,* este presenta un rango de valores permitido entre -0,1 y 3,3V, estas especificaciones definen ciertos parámetros de diseño que se expondrán a continuación:

La señal de salida de la pinza en condiciones extremas es una curva senoidal de amplitud (+/-)1,4 Vpico; como no se permiten valores negativos en la entrada al *ADC,* se debe sumar un nivel de *offset*[16] positivo, de tal forma que la señal senoidal se ubique por encima de cero, esto está en concordancia con las condiciones de entrada del *ADC.* El valor de *offset* sumado es igual a 1,5 V, y el rango de valores de entrada al conversor entonces estará entre 0,1 V y 2,9 V; esto supone que los valores que se

[16] Señal de corriente continua que se mantiene constante, es llamado también nivel de continua, se emplea para ajustar señales de corriente alterna.

encuentran por debajo o por encima de este rango (hasta -0,1 V ó hasta 3,3 V, respectivamente) serán picos de corriente negativos o positivos. De esta forma se reconocerá si el dispositivo esta trabajando a 10 A correctamente ó si se esta presentando algún pico de corriente; lo cual lleva, a que la relación entre la corriente y la tensión magnética en la entrada del conversor, es de 0,1 V a 1 A para el mínimo valor y 2,9 V a 10 A para el máximo valor de señal de corriente. Vale la pena aclarar que la adquisición será correcta aunque se sobrepase este rango; el problema se presentará en la pinza porque ésta se sobrecarga; es decir, se estará midiendo mas corriente de la que la pinza pueda soportar, lo cual no asegura una correcta medida; en términos cuantitativos se podrá adquirir una señal de hasta 12 A; por encima de este valor la salida se saturará.

Cabe aclarar también que el rango de valores tomado de esta forma se debe a que la calidad de respuesta del *ADC* se ve afectada en los límites de entrada, así que para evitar errores en la estimación a causa de este factor, se decidió no tomar el rango permitido por el *ADC* en su totalidad. Sin embargo, si se hace la medición con una pinza que permita mayor corriente, el prototipo podrá hacer la estimación sin error hasta 12 A.

En este momento, la señal de corriente ha sido sensada, pero necesita un circuito que regule y filtre la señal para su posterior procesamiento.

2.1.2 Protección del dispositivo.

Para proteger el dispositivo (específicamente el conversor) de sobrecargas o picos de tensión en la entrada se cuenta con la saturación que presentan todos los circuitos integrados de la tarjeta de adquisición; pues se alimentan con 3,3v, por consiguiente la salida se limita a este valor; (pruebas de

laboratorio lo demostraron, ya que los elementos se saturaron a 3,3v cuando la señal superó el umbral), ofreciendo una protección inmediata al conversor. No obstante es importante aclarar que la pinza está diseñada para mediciones de corriente máxima de 15A, si se sobrepasa este valor, es muy probable que se reduzca la vida útil de la misma y por tanto el resto del equipo se aísla de estas señales, conservando el ADC.

2.1.3 Adecuación de la señal (offset).

A la señal sensada por la pinza se le debe sumar una señal de offset que ubique el pico mas negativo de la señal por encima de cero, como se había planteado originalmente este valor debe ser 0,1v que corresponde a –10A, asimismo el pico mas positivo no debe sobrepasar los 3,3v, para lo cual a 10A se tiene 2,9v (-10A y 10A son los valores mínimo y máximo de corriente que debe medir la pinza para conservar intactas sus características de medición). Para realizar esta tarea se implementó un sumador no inversor, mediante un OPAM17, resistencias y un regulador de tensión18 cuya función es fijar a 1,5V el nivel de la señal de continua que se sumará a la tensión magnética, de forma no que no se presenten variaciones en ésta a causa de pequeñas variaciones en la fuente la cual por supuesto no es ideal.

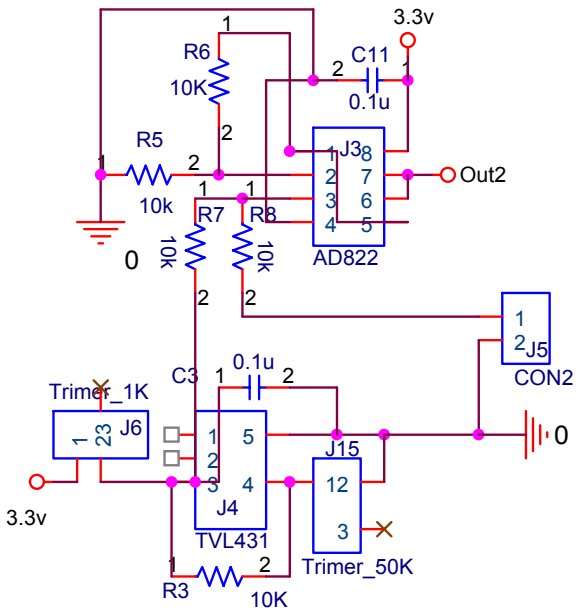
Antes de llegar al filtro es necesario llevar la señal a través de un seguidor de tensión y así evitar las corrientes de carga que se pueden presentar a la entrada del filtro. Los seguidores de tensión se implementan con circuitos amplificadores operacionales integrados, uniendo la entrada inversora a la salida, así se mantiene la realimentación pero sin resistencias y a la entrada no inversora se conecta la señal de tensión magnética ya adecuada. Gracias

[17] Circuito Amplificador Operacional Integrado, se utilizó el AD822 fabricado por Analog Devices.
[18] El regulador utilizado fue el TLV431 fabricado por Texas Instruments.

a la alta impedancia de entrada con que cuentan los *OPAMs* se asegura que la señal de entrada al filtro sea netamente tensión y que este libre de corrientes de carga que introduzcan error a la estimación.

La figura 6 presenta una gráfica esquemática de la etapa de adecuación:

Figura 6. Esquemático de la etapa de adecuación.

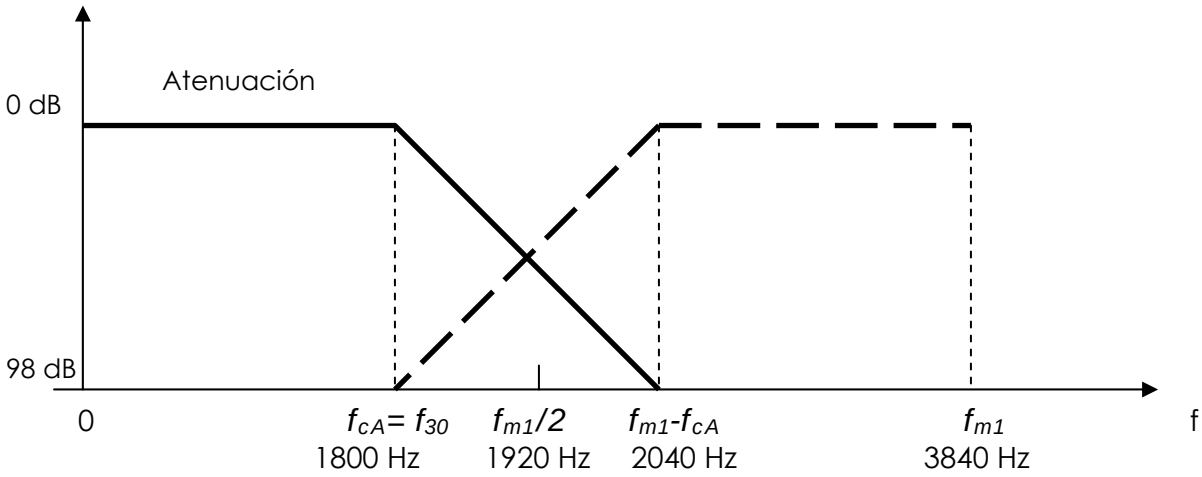


2.1.4 Filtrado anti-solapamiento.

El filtro anti-solapamiento en hardware se implementó con el fin de restringir el paso de frecuencias no deseadas, es decir frecuencias por encima del ultimo armónico a evaluar y de esta forma disminuir el tiempo de procesamiento y la aparición de frecuencias redundantes; sin embargo más adelante se verá que mediante software también debe realizarse este filtro, de lo contrario la implementación en hardware sería muy exigente.

La forma ideal del filtro que se requiere implementar se muestra en la figura 7.

Figura 7. Filtro anti-solapamiento ideal



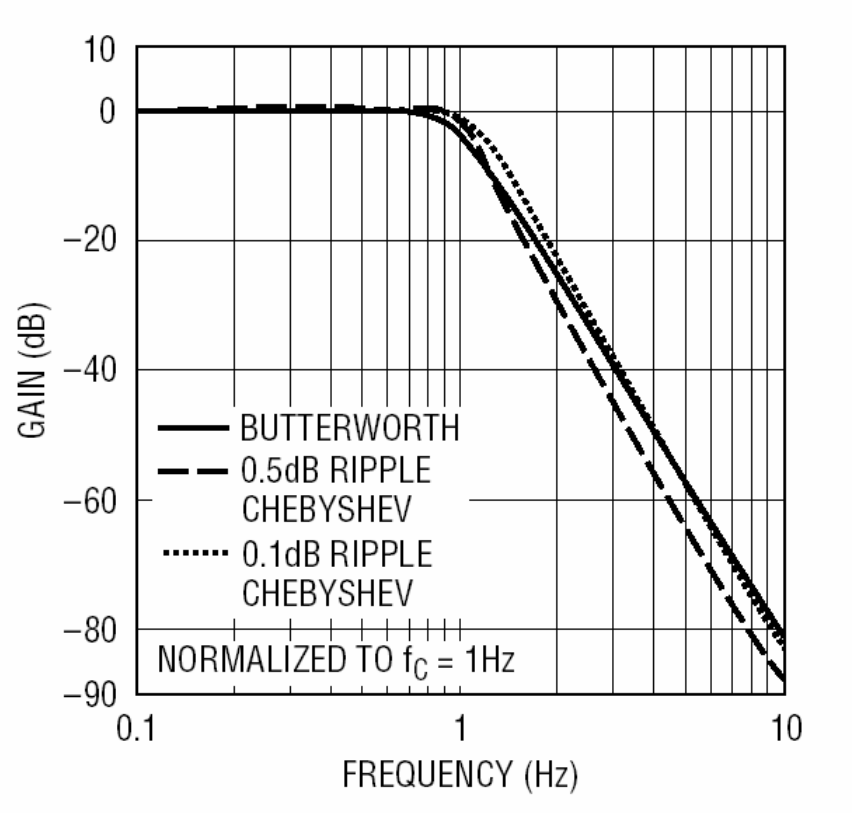
Aporte de los autores

Básicamente se trabaja con un circuito operacional integrado[19], el cual es un filtro *Butterworth*[20] de cuarto orden [Sedra, Smith, 2002], que tiene entre sus características: respuesta máximamente plana, caída por década aproximadamente de -80dB, ajustándose a los requerimientos del diseño para filtrar la señal, además de resistencias y condensadores.

La Figura 8 es una gráfica de la respuesta en frecuencia del circuito integrado utilizado (LTC1563-2 fabricado por *Linear Technologies*) junto con otros integrados que poseen otro tipo de filtros; comprobando que el integrado escogido es idóneo para este trabajo.

[19] Referencia LTC1563-2 fabricado por Linear Technologies
[20] Filtro pasa bajas, su nombre se debe a su descubridor, sin embargo también recibe el nombre de filtro máximamente plano ó plano-plano

Figura 8. Respuesta en Frecuencia del Filtro LTC1563-2



Tomado de la hoja de datos LTC1563-2

En términos ideales el filtro anti-solapamiento necesario para estimar hasta el armónico 30^{avo} debe tener las siguientes características (Véase la figura 7)

- Ser pasa bajas con una frecuencia de corte aproximadamente de 1,8KHz.
- Respuesta máximamente plana.
- Ganancia unitaria.
- Caída de -98db a una frecuencia de 2,04kHz
- Caída de –100db a una frecuencia de 3,84kHz.

El criterio de *Nyquist* especifica que la frecuencia de muestreo debe ser por

lo menos dos veces la frecuencia máxima a analizar, para este caso se determina que esta frecuencia debe ser entonces de 3,6KHz, sin embargo como se trabaja con potencias de 2, este valor es un poco mayor, es decir que se tiene una frecuencia de muestreo de 3,84KHz si se tienen 64 muestras por ciclo de 60Hz. Para situaciones en las que se estime un mayor número de armónicos por ejemplo, si se requiere la medición hasta el armónico 50, esta frecuencia de muestreo se hace mayor (específicamente 7,68kHz, esto es equivalente a tomar 128 muestras por ciclo de 60Hz [Oppenheim, Schafer, 2000] pero siempre cumpliendo el principio de Nyquist y la tasa de muestreo potencia de 2.

Ahora bien, la atenuación de –98 db a 2,04kHz es un criterio de diseño definido por la resta de la frecuencia de muestreo (3,84kHz) y la máxima frecuencia a evaluar (1,8kHz) [Ordoñez, 2002].

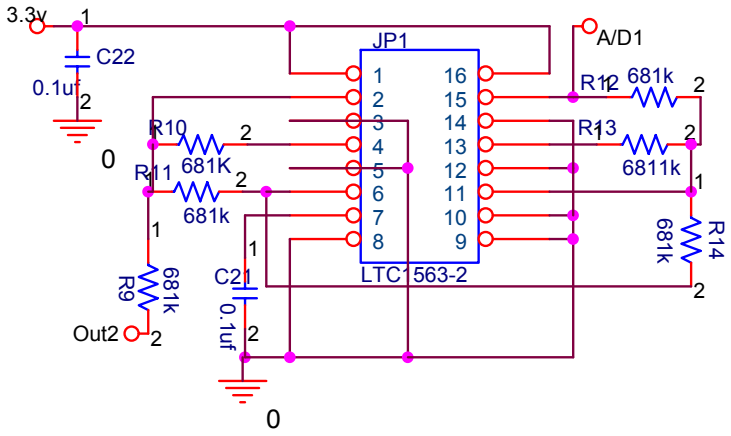
No obstante, estos requerimientos pueden ser menos exigente si se hace sobremuestreo; es decir, en lugar de muestrear al doble de la máxima frecuencia a analizar, según la teoría de *Nyqist*, se multiplica esta frecuencia de muestreo por un valor X (siendo éste un múltiplo entero de la frecuencia fundamental) de forma tal que la frecuencia de muestreo es superior a la recomendada.

Para este prototipo, se tiene una frecuencia de muestreo de 15,36kHz; es decir, se ha sobremuestredo por un factor de 4. Con esta frecuencia, la atenuación de –98db debería presentarse en 11,76kHz y a 15,36kHz se debería tener una atenuación de –100db conservando el criterio de diseño inicial; sin embargo, el filtro finalmente implementado tiene una frecuencia de corte de 3,6kHz y una atenuación de –80db por década. Lo que no permite satisfacer los requerimientos de diseño. Esto en primera instancia no presenta mayores problemas ya que mediante procesamiento digital se logra

una atenuación de –98db a 2,04kHz como idealmente se requiere; asimismo con un factor de sobremuestreo de 4 los resultados finales son apropiados, para una buena estimación.

La figura 9 es el esquemático del filtro implementado:

Figura 9. Esquemático del filtro anti-solapamiento usando el LTC1563-2



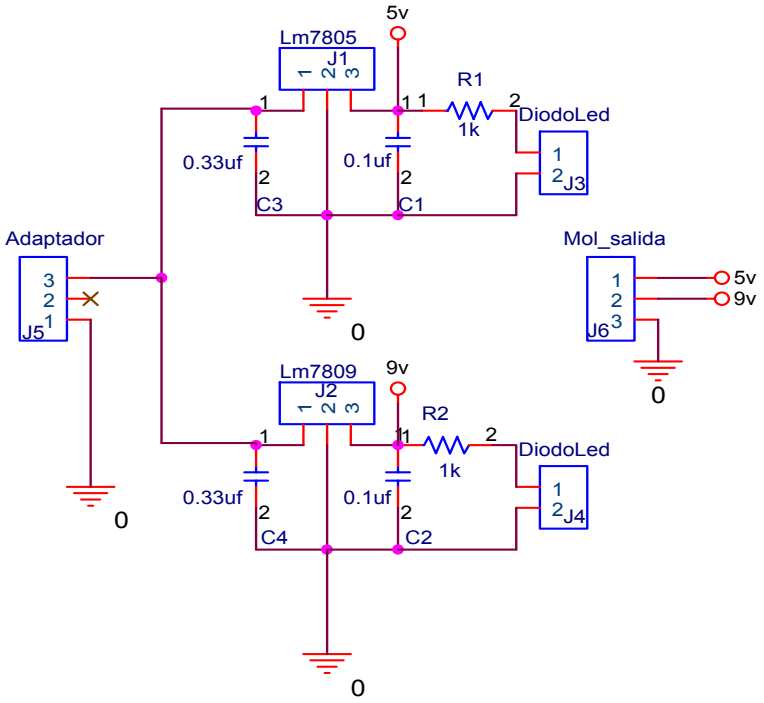
2.1.5 Fuente de alimentación.

La fuente de alimentación para la tarjeta de adquisición de los armónicos se implementó mediante un circuito integrado regulador de tensión (LD1117)21, resistencias y condensadores. Se toma como entrada al regulador, una señal de 5v de corriente continua tomada de un diseño de fuente basado en dos reguladores tipo TO22022; uno de 5 V que provee tensión y corriente necesarias para alimentar la tarjeta y al SED1330F (Integrado que se encarga de la visualización en la pantalla), el otro de 9v que alimenta la demoboard del DSP; a su vez estos reguladores toman la tensión de entrada y la corriente que requieren, de un adaptador mínimo de 5W, pues la

[21] Referencia LD1117 *SOIC* de 3.3v, fabricado por *STMicroelectronics*
[22] Empaquetamiento de circuitos electrónicos.

potencia total que puede consumir el equipo es de 3,9 W, el esquemático de este circuito se observa en la figura 10.

Figura 10. Esquemático de los reguladores para la fuente de 5 V y 9 V.



De otro lado, este LD1117 es de vital importancia, pues además tiene como función la protección en cuanto a picos de tensión en la alimentación se refiere, ya que su magnitud en la salida será siempre una señal de continua constante a 3,3v así se presentasen picos de tensión en la entrada (esto se contrastó con los resultados de las pruebas realizadas en el laboratorio).

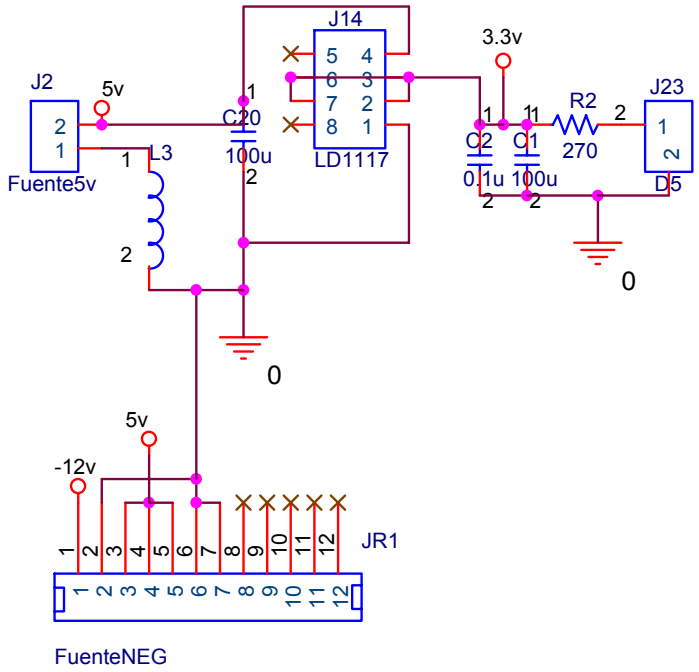
Ahora bien para alimentar la pantalla se necesita una fuente de corriente continua entre -25 V y -10 V, para este fin se utilizó un regulador integrado *ISR*[23] este regulador tiene como característica principal que aunque es alimentado con 5v(Pines de alimentación múltiples) puede proveer una tensión de salida de +/-12 V hasta +/-15 V (la tensión de salida se ajusta

[23] *Integrated Switching Regulator*. Referencia PT5061 fabricado por *TEXAS INSTRUMENTS.*

mediante resistencias externas), por razones obvias utilizar este regulador fue muy conveniente, pues los pines de entrada se conectaron a los 5v con que se dispone en la fuente de los TO220*,* la salida que arrojó fue de -12v como se esperaba; se hace mención al respecto porque en un principio se pensó que serían necesarias dos tipos de fuente para alimentar el prototipo, una para la *demoboard* del *DSP* y otra para alimentar tanto la tarjeta de adquisición como la pantalla; vale la pena mencionar que no se hizo uso del adaptador que trae la *demoboard* consigo, pues éste no tiene la suficiente potencia como para alimentar todo el prototipo. Esto simplificó el trabajo y redujo el tamaño físico del mismo.

Es consecuente mencionar el problema que se puede presentar al mezclar la tarjeta de adquisición, la pantalla y la *demoboard,* pues en este caso se trabaja con dos tipos de tierra, una analógica y otra digital, es posible que se inyecte ruido sobre la tarjeta analógica a causa de la tarjeta digital (debido a las altas frecuencias que esta ultima manipula) o viceversa; para evitar este inconveniente se unen ambas tierras a través de una bobina de choque (preferiblemente torroidal) de modo que a altas frecuencias se obtiene una impedancia muy alta en la bobina, comportándose ésta como un circuito abierto, que no permite el paso de este tipo de señales hacia ningún circuito, pero cuando se alimenta el circuito con una señal de continua, la bobina se comporta como un corto circuito, que no ocasiona problemas de tensión en la tierra. La figura 11 muestra el circuito implementado para la alimentación de la tarjeta.

Figura 11**.** Implementación de la fuente de alimentación para la tarjeta.



Asimismo esta tarjeta es el medio de comunicación entre el *DSP* y la pantalla, pues ambos componentes envían la señal de salida o entrada según sea el caso, desde este *hardware*, no se hace directamente a causa de la etapa antes mencionada, que necesitaba ser implementada en una tarjeta, además como se trabaja comunicación serial (en el capítulo 4 se explica esto con detalle, incluido el *hardware*), se trabaja con correas que van del *DSP* a la tarjeta y de la misma a la pantalla y al registro de desplazamiento necesario para este tipo de comunicación.

Para terminar en este capítulo se presentan dos graficas, la figura 12 con el esquemático del diseño final, que incluye: la etapa de sensado de la pinza, el sumador de *offset*, el seguidor de tensión, el filtro y la fuente y la figura 13 con la imagen de la tarjeta implementada.

Figura 12. Esquemático de la tarjeta de adquisición.

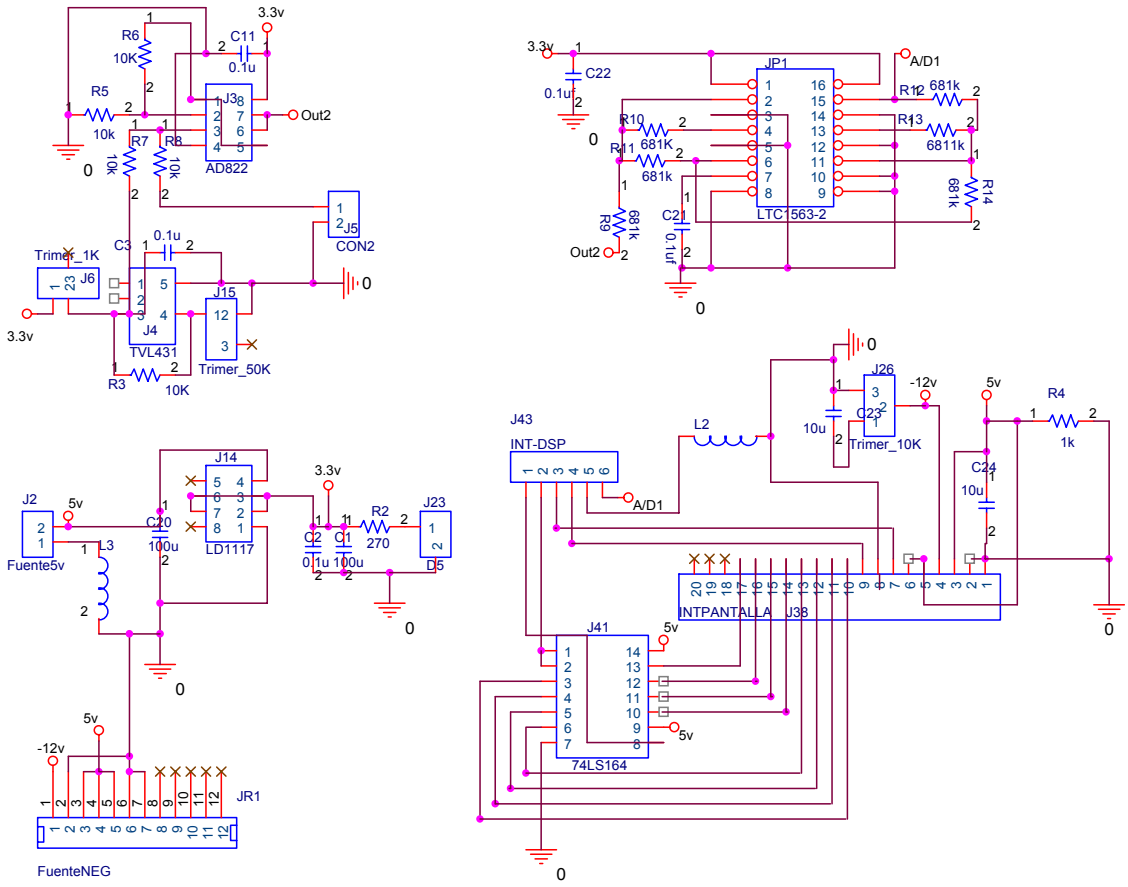
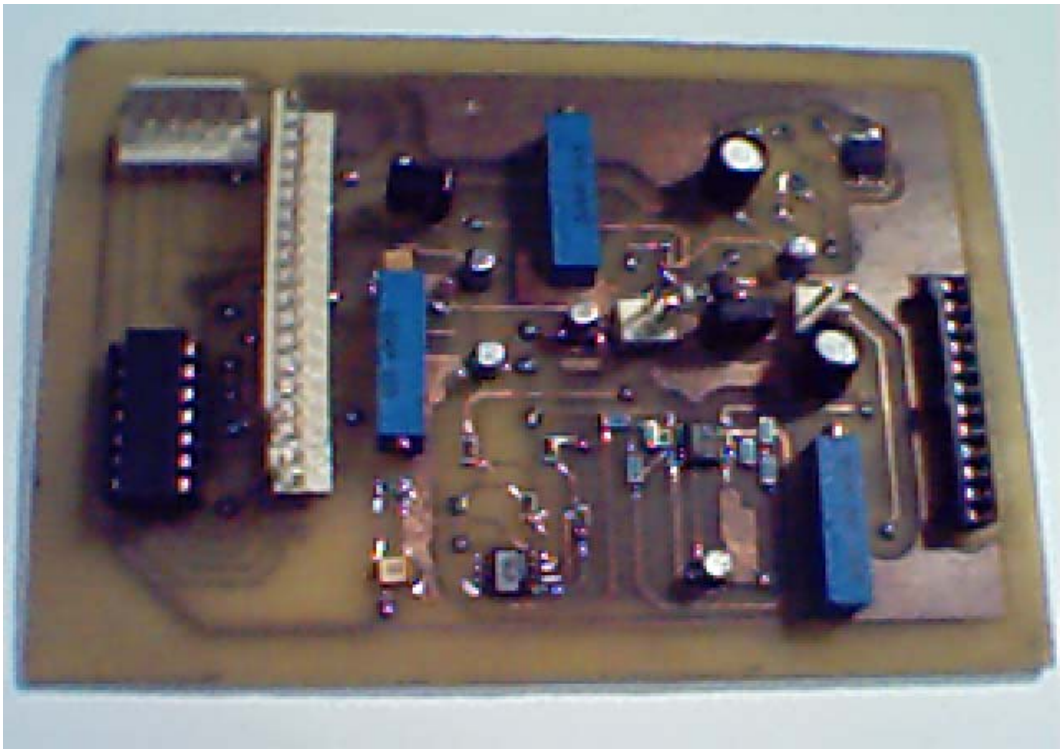


Figura 13. Tarjeta de adquisición de señal implementada.

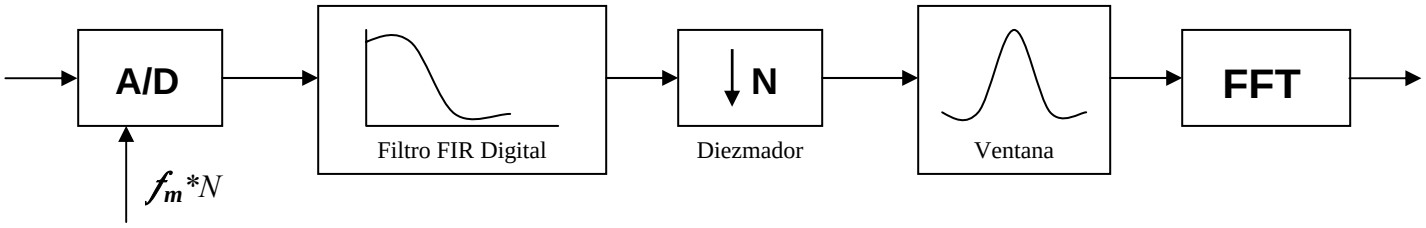


3. PROCESAMIENTO DIGITAL

En este capítulo se aborda el problema del procesado digital de señal a partir del muestreo de una señal analógica que ya ha sido adaptada para poder ser empleada por el *DSP*[24], tal como se explicó en el capítulo 2.

Las diferentes etapas que conforman el procesamiento digital se muestran globalmente en la figura 14, donde se distinguen los siguientes bloques:1) Muestreo, 2) Filtrado digital y diezmado, 3) Enventanado y 4) Transformada rápida de Fourier.

Figura 14. Diagrama de bloques del procesamiento digital



Fuente: Investigación de los autores.

Al final del capítulo se explica cómo se implementó cada una de las etapas del procesamiento en el *DSP* y el por qué de las restricciones que se realizaron en cada una de las etapas mencionadas.

[24] *Digital Signal Processor*

3.1 MUESTREO

La frecuencia de muestreo debe seleccionarse tal como lo establece el teorema *Nyquist,* es decir, la frecuencia de muestreo debe ser mayor al doble de la frecuencia más alta que contenga la señal a analizar.

Según [Ordoñez, 2002], en el análisis de componentes armónicas en estado estacionario el análisis espectral debe permitir estimar componentes hasta el 50^{avo} armónico, razón por la cual la máxima frecuencia a tenerse en cuenta es de 3KHz en el sistema eléctrico colombiano, pero por razones de capacidad de memoria del procesador el mayor armónico a estimar será el 30^{avo}, y por ende la máxima frecuencia a estimar es de 1,8KHz.

La frecuencia de muestreo utilizada en el prototipo sería entonces de 3840 Hz, lo que corresponde a tomar 64 muestras por ciclo de 1/60 seg., ya que con este número de muestras por ciclo se garantiza que el análisis espectral se puede realizar hasta el armónico 30^{avo}.

Con la frecuencia de muestreo de 3840 Hz., el filtro anti-solapamiento que se debe diseñar analógicamente es muy exigente, lo que puede generar distorsión en la señal a estimar, debido al alto grado del filtro analógico; es por eso que la frecuencia de muestreo se debe incrementar por un factor de sobre-muestreo que según la referencia [Ordoñez, 2002] puede ser de 8, para así diseñar el filtro anti-solapamiento analógico no muy exigente y complementar el proceso de filtrado digitalmente.

En el diseño final, el factor de sobre-muestreo se fijó en 4, debido al alto consumo de memoria que se producía usando un factor de 8 y a que no hay diferencias relevantes en los resultados utilizando un factor de 4.

3.2 FILTRADO DIGITAL Y DIEZMADO

La implementación del filtro digital se convirtió en el cuello de botella del trabajo en cuanto a *software* se refiere, debido a las exigencias del algoritmo ante las limitantes propias del *DSP* como lo son la memoria RAM de datos, la aritmética de coma fija y el limitado *set* de instrucciones que maneja.

La respuesta en frecuencia ideal del filtro digital a implementar se muestra en la Figura 7 del Capítulo 2; en donde lo más importante es alcanzar una atenuación de -98 dB, en la banda de rechazo del filtro.

El filtro que se decidió implementar fue un filtro *FIR*[25], en lugar de un filtro *IIR*[26], debido a que los filtros *FIR* pueden tener una fase lineal (generalizada) de forma precisa, que para análisis de componentes armónicas estacionarias resulta fundamental [Oppenheim & Schafer, 2000] .

Un filtro *FIR* está definido por la siguiente expresión:

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k] \qquad (5)$$

[25] Finite Impulse Response
[26] Infinite Impulse Response

donde *x[n]* representa la entrada del filtro, b_k representa los coeficientes del filtro, *y[n]* representa la salida del filtro y N la longitud del filtro. Si la señal *x[n]* es reemplazada por un impulso $\delta[n]$ la Ecuación (5) quedaría:

$$y[n] = \sum_{k=0}^{N-1} b_k \delta[n-k] = h[n] \qquad (6)$$

donde *h[n]* es la respuesta al impulso del filtro, dado que:

$$\delta[n-k] = \begin{cases} 0 & \text{para } n \neq k \\ 1 & \text{para } n = k \end{cases}$$

De la ecuación (6) se deduce que:

$$\begin{aligned} b_0 &= h[0] \\ b_1 &= h[1] \\ &\vdots \\ b_k &= h[k] \end{aligned} \qquad (7)$$

De la ecuación (7) se puede concluir que los coeficientes del filtro son muestras de la respuesta al impulso, lo que significa que la base del problema del filtrado se dirige al método utilizado para el cálculo de los coeficientes. Entre los métodos más conocidos para el cálculo de los coeficientes se encuentran el método de ventanas y muestreo en frecuencia Park-McClellan [Oppenheim & Schafer, 2000], [Proakis & Manolakis 98]. En esta aplicación se ha seleccionado el método de ventanas porque es un procedimiento directo y general. Ahora se debe tomar la decisión sobre el tipo de ventana que más se adecue a las exigencias requeridas por el filtro, dentro de las cuales las más comunes son: la ventana rectangular, *Hanning,*

Hamming, Blackman y *Kaiser*. [Oppenheim & Schafer, 2000], [Dahnoun, 2000].

A excepción de la ventana *Kaiser*, las demás ventanas presentan un límite de atenuación en la banda de rechazo fijo tal como se indica en la tabla 1, de donde se puede apreciar que la máxima atenuación lograda con estas ventanas es de 74 dB. al utilizar una ventana *Blackman*.

Tabla 1. Características de las ventanas más conocidas.

Ventana	Ancho de transición normalizado (ΔF (Hz))	Rizo en la banda de paso (dB)	Atenuación en la banda de rechazo (dB)
Rectangular	0,9/N	0,7416	21
Hanning	3,1/N	0,0546	44
Hamming	3,3/N	0,0194	53
Blackman	5,5/N	0,0017	74

N: longitud del filtro

Tomado de [Dahnoun, 2000]

3.2.1 Ventana Kaiser.

En vista de que ninguna de las ventanas presentadas ofrece una atenuación como la requerida por el filtro, -98 dB, se utilizó la ventana Kaiser, cuyo diseño parte de la atenuación necesaria en la banda de rechazo y en el ancho de la banda de transición. A continuación se presenta la ventana Kaiser y las ecuaciones que se emplean para el diseño de filtros mediante este método.

La ventana de *Kaiser* se define como**:**

$$w[n]=\begin{cases}\dfrac{I_0\left[\beta\left(1-\left[(n-\alpha)/\alpha\right]^2\right)^{1/2}\right]}{I_0(\beta)}, & 0\le n\le M-1\\ 0, & en\ \ el\ resto\end{cases} \qquad (8)$$

siendo $\alpha = M/2$ e $I_0(\cdot)$ la función de *Bessel* modificada de primera especie. Variando *M* (longitud) y β (parámetro de forma), se puede cambiar la longitud y la forma de la ventana, ajustándose a su vez el compromiso entre amplitud de los lóbulos laterales y el ancho del lóbulo principal. [Oppenheim & Schafer, 2000].

Kaiser determinó empíricamente que el valor de β necesario para obtener una atenuación *A* está dado por:

$$\beta=\begin{cases}0{,}1102(A-8.7), & A>50\\ 0{,}5842(A-21)^{0,4}+0{,}07886(A-21), & 21\le A\le 50\\ 0, & A\le 21\end{cases} \qquad (9)$$

Además *Kaiser* descubrió que una vez estén especificados los valores de *A* y $\Delta\omega$ (ancho de la banda de transición), *M* se puede calcular mediante la siguiente ecuación:

$$M=\frac{A-8}{2{,}285\Delta\omega} \qquad (10).$$

La Ecuación (10) permite calcular el valor de *M* con una precisión de ±2 muestras, para un amplio margen de valores de *A* y $\Delta\omega$, por tanto el método de diseño basado en una ventana *Kaiser* casi no requiere de iteraciones o

ensayos de prueba y error [Oppenheim & Schafer, 2000]. $\Delta\omega$ está definido como:

$$\Delta\omega = \omega_s - \omega_p \qquad (11)$$

en donde ω_p es la frecuencia de corte en la banda de paso y ω_s es la frecuencia de corte de la banda de atenuación, en radianes.

Para efectos del diseño y partiendo de una frecuencia de sobremuestreo f_{sm}= 4 * 3840 $\therefore$ f_{sm}= 15360, se obtienen los siguientes parámetros:

- A = 98 [dB]
- $\omega_p = \frac{1800 * 2\pi}{15360} = 0{,}234375\pi$ [rad] , $\omega_s = \frac{2040 * 2\pi}{15360} = 0{,}265625\pi$ [rad]
- $\Delta\omega = 0{,}03125\pi$ [rad]
- *M = 403*

Se realizó un programa de simulación en MATLAB 5.3, el cual obtiene la respuesta en frecuencia de un filtro pasabajas basado en la ventana *Kaiser*. Mediante simulaciones se estudió la respuesta en frecuencia del filtro diseñado, variando los factores de sobremuestreo. Los resultados de dichas simulaciones se pueden apreciar en las Figuras 15 a 17. Cabe resaltar que el barrido en frecuencia mostrado en estas figuras se realiza hasta la mitad de la frecuencia de sobremuestreo, que en términos de radianes equivale a hacer un barrido en el intervalo [0 , π], suficiente para apreciar su calidad.

Figura 15. Respuesta en frecuencia del filtro pasabajas con $f_{sm} = 8*f_m$

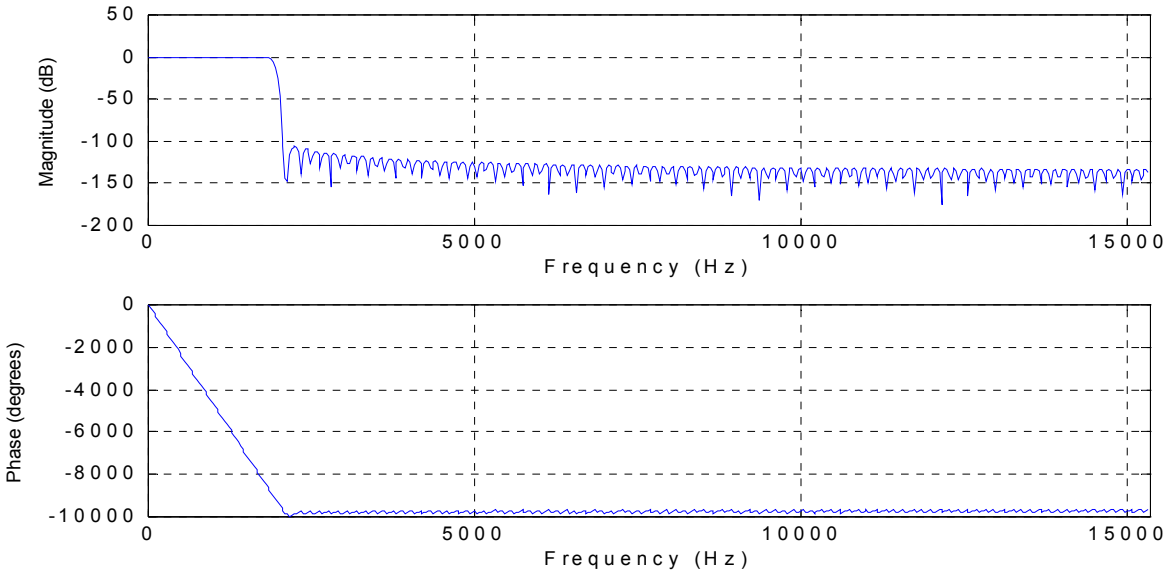


Figura 16. Respuesta en frecuencia del filtro pasabajas con $f_{sm} = 4*f_m$

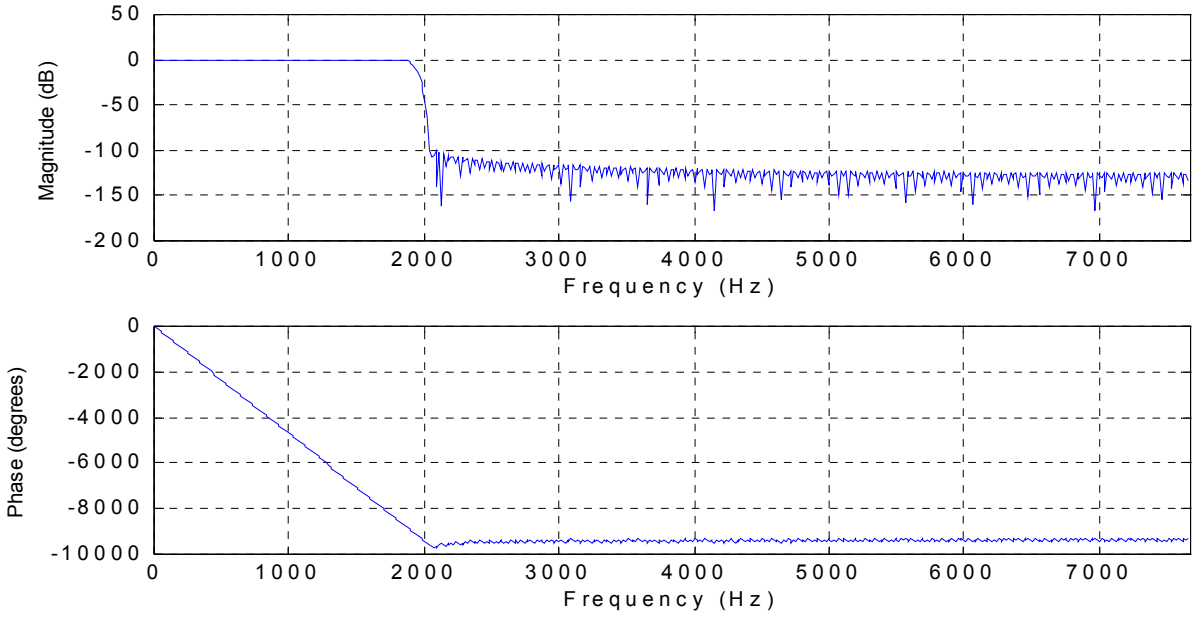
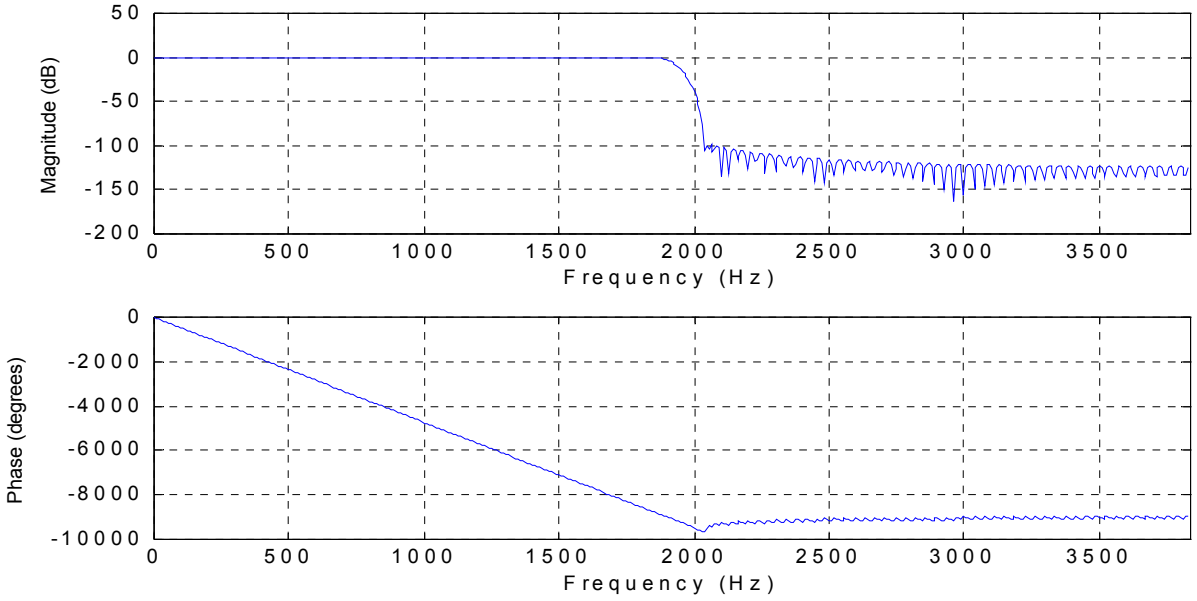


Figura 17. Respuesta en frecuencia del filtro pasabajas con $f_{sm} = 2*f_m$



Fuente: Investigación de los autores.

Las diferencias tanto en magnitud como en fase de cada una de las figuras no son significativas, debido a la gran eficiencia del filtro diseñado; las diferencias se hacen notables al revisar la longitud de cada uno de los filtros a diferentes frecuencias tal como se muestran en la tabla 2.

3.2.2 Efecto de la cuantificación en el diseño del filtro.

Los resultados mostrados anteriormente en las figuras 15 a 17, fueron calculados en Matlab. Utilizando aritmética de coma flotante y variables de doble precisión, es decir 64 bits. De otra parte, el DSP56F801 trabaja con 16 bits y tiene aritmética de coma fija. En la tabla 2 se presenta una relación de cómo esto afecta críticamente la relación entre la atenuación deseada y la atenuación alcanzada con 16 bits y 32 bits. Si el filtrado se realiza a 32 bits, la relación mejora ostensiblemente.

Tabla 2. Relación entre frecuencias de muestreo, longitud del filtro y fenómeno de cuantificación

Atenuación deseada [dB]	2*FM			3*FM			4*FM			8*FM		
	N	Aq_{16}	Aq_{32}	N	Aq_{16}	Aq_{32}	N	Aq_{16}	Aq_{32}	N	Aq_{16}	Aq_{32}
60	117	61	60	175	60	60	233	62	60	465	62	62
70	139	69	70	208	66	70	277	67	70	554	62	75
75	150	72	74	225	69	77	300	68	77	598	64	80
80	161	73	79	242	70	79	322	67	82	643	63	86
90	184	72	90	275	68	90	367	66	92	732	63	90
95	195	70	95	292	70	96	389	67	98	777	64	97
98	**202**	**71**	**98**	**302**	**72**	**99**	**403**	**67**	**98**	**803**	**62**	**98**
105	217	69	107	325	68	105	433	64	106	866	65	110
110	228	72	111	342	70	111	456	67	111	910	63	112

Fm = 3 840 Hz *Aq16 =Atenuación resultante16 bits.*
N = longitud del filtro *Aq32= Atenuación resultante 32 bits.*
Fuente: Investigación de los autores.

Ante la necesidad de almacenar los coeficientes del filtro en un formato de 32 bits en la memoria del *DSP* se determino que la frecuencia de sobremuestreo no fuera de $8*f_m$ sino de $4*f_m$ tal y como explicó al comienzo de este capítulo.

3.2.3 Cálculo de los coeficientes.

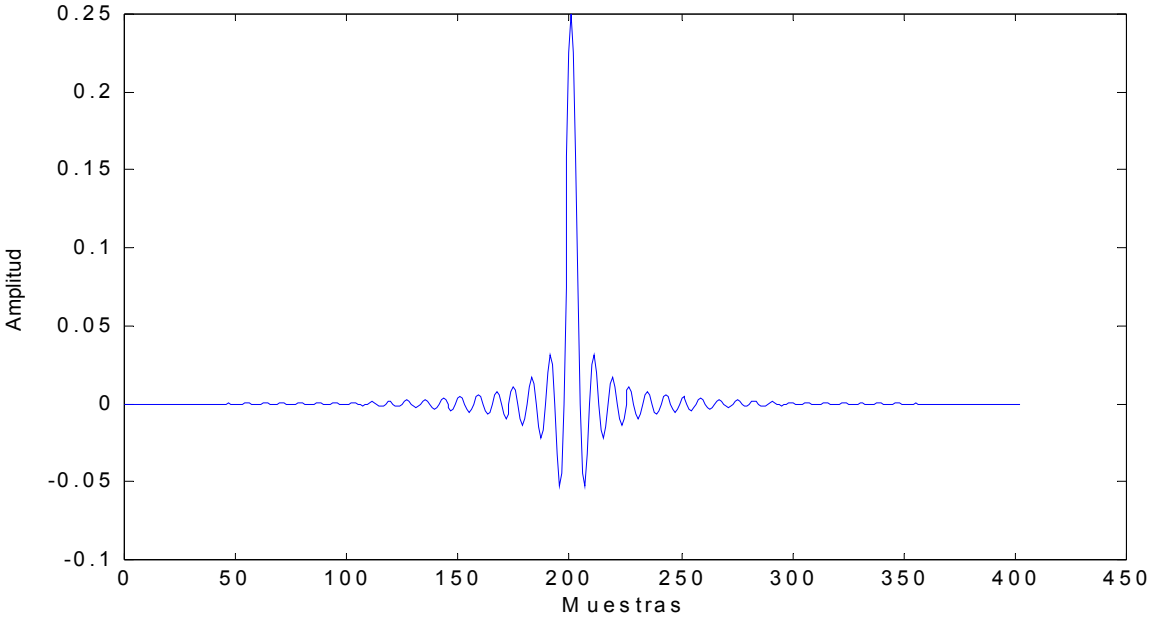
Como se mostró en la ecuación (7) la respuesta al impulso del filtro FIR está dada por el valor de los coeficientes de la ecuación de diferencias que define al filtro. El mismo programa implementado en MATLAB para obtener las respuestas en frecuencia de los filtros diseñados, permite calcular la

respuesta al impulso (Figura 18) y el valor de los coeficientes en formato Q-15 y Q-31.

3.2.4 Diezmado.

EL diezmado es el proceso mediante el cual se eliminan las muestras sobrantes que aparecen por el efecto de sobremuestreo ya que esta tarea ha cumplido su finalidad. El objetivo del diezmado es disminuir la carga computacional del sistema, procesando un número de muestras menor pero apropiado para el correcto procesamiento de la siguiente etapa. El diezmado en programación, es un bloque transparente, que se implementa junto con la etapa de filtrado.

Figura 18. Respuesta al impulso del filtro pasabajas



Fuente: Investigación de los autores.

Al muestrear con factor de sobremuestreo igual a 4, se obtienen 256 muestras por ciclo, pero como el filtro tiene una longitud de 402, es necesario

que mínimo se tengan almacenadas este número de muestras para el cálculo de la primera salida del filtro. Idealmente el filtro debe realizar 256 veces, 402 productos, sumas y desplazamientos, cada 1/15360 seg.; es decir, que en cada periodo de muestreo se debe poder calcular la salida del filtro. En un primer diseño se implementó este tipo de solución, pero resultó que el procesado duraba más que la llegada de la siguiente muestra.

La solución que se presentó ante esta limitante fue, que en lugar de empezar a filtrar en medio de la llegada de cada muestra, se retarde el proceso de filtrado hasta obtener, además de las 402 muestras ya almacenadas, 256 muestras adicionales, correspondientes a los 256 desplazamientos que requiere el filtrado, para obtener un total de 658 muestras, de tal forma que se filtre una vez finalizado el proceso de muestreo. El proceso de multiplicación y adición concerniente al filtrado se realiza cada 4 muestras para obtener un vector de longitud 64 en su salida, realizando de esta forma el proceso de diezmado.

3.3 ENVENTANADO

El enventanado es un proceso de ponderación de la señal a la cual se le van a calcular sus componentes armónicas, con el fin de atenuar los efectos producidos por un posible deslizamiento en la frecuencia de 60 Hz.

El estudio de los efectos y los resultados obtenidos por la utilización de una u otra ventana, se expone claramente en las siguientes referencias [IEC 61000-4-7, 91[Rodríguez & Uribe, 2003]

En el prototipo final se habilitó la posibilidad de que el usuario pueda escoger el tipo de ventana que desea utilizar en su proceso de medición. Las ventanas habilitadas para este fin son la ventana rectangular, *Hamming, Hanning* y *Blackman* [Oppenheim & Schafer, 2000]. Estas se implementaron mediante funciones del *SDK* que permiten las operaciones entre números en formato *Frac16,* tales como el coseno, el producto, y la suma; ya que estas funciones no vienen incluidas en el *CodeWarrior* 5.1.

3.4 TRANSFORMADA RAPIDA DE FOURIER (FFT)

Para la realización de la transformada de Fourier, se utilizó una función de biblioteca incluida en el *SDK* llamada ***rfft*** *(Real Fast Fourier Transform).*
Esta función realiza el algoritmo de *FFT* mediante diezmado en el tiempo y con cómputo en las mismas posiciones de memoria donde se almacena la señal de entrada. El vector de entrada está en formato fraccional de 16 bits y su número de muestras debe ser potencia de dos. Este algoritmo es conocido como *radix-2*. La salida es un vector complejo cuya longitud es la mitad de la longitud del vector de entrada, todo en formato fraccional de 16 bits. La salida equivale a la siguiente ecuación [Motorola, SDK, 2003]:

$$z[k] = escala\sum_{i=0}^{N-1} x[i](\cos(\frac{2\pi ik}{N}) - j\sin(\frac{2\pi ik}{N})) \qquad 0 \le k \le N \qquad (11)$$

Dentro de las especificaciones de entrada de la función se debe tener en cuenta la siguiente restricción en el vector de entrada, para obtener resultados apropiados:

$$(x[i])^2 + (x[i+1])^2 < 1 \qquad (12)$$

siendo *x(i)*, la muestra *i* del vector de entrada, esto se garantiza escalando el vector de entrada por un valor ajustable, que también hace parte de los parámetros de entrada de la función y que para el caso de éste proyecto se ajusto al valor de 64.

La función *rfft* debe ser inicializada con la función ***rfftInit*** *(initilize Real FFT)*. Con esta función se especifican la longitud de la *fft* y una serie de banderas que permiten establecer el escalamiento de los datos de la entrada y la salida y cambiar el orden de los datos de entrada y salida entre otros.

Para poder llamar estas funciones es necesario modificar en el archivo *appconfig.h*, del proyecto creado por *SDK*, la línea
#undef INCLUDE_DSPFUNC; por la línea, *#define INCLUDE_DSPFUNC.* Luego de esto se procede a incluir la cabecera *dfr16.h* al principio del archivo en el cual se van a emplear.

Las funciones se llaman de la siguiente forma:
dfr16RFFTInit (pRFFT , n , options) y *res = dfr16rFFT (pRFFT ,*px* ***,*** *******pz);*
En donde pRFFT es un puntero de un tipo de estructura creado por el *SDK,* especial para el manejo de este tipo de funciones. **px* es un puntero tipo *Frac16 (fraccional)* y apunta a la posición inicial del vector de entrada de la FFT, y **pz* es un puntero tipo *CFRAC16 (Fraccional complejo)*, en donde se guardan los resultados de la FFT.

3.5 PROGRAMACIÓN DEL PROCESAMIENTO

A continuación se explicará cómo se programó en el DSP cada una de las anteriores etapas mencionadas.

3.5.1 Muestreo.

El muestreo del prototipo se realizó mediante la utilización de dos periféricos, el conversor analógico digital y un temporizador interno, QuadTimerC. Este temporizador no tiene conexión a pines. El temporizador atiende una interrupción en el momento que alcanza un conteo equivalente a 1/15630 s. Al atender la interrupción, la tarea que se realiza es leer el ADC y guardar la muestra en un vector ya definido de longitud 658, hasta llenarlo, una vez lleno el vector de muestras se detiene la operación del temporizador.

El conversor analógico digital se configuró de la siguiente forma:

- Cada entrada del *ADC* se usó independiente, habilitando únicamente el canal 0.
- Frecuencia de muestreo del *ADC* de 15 MHz.
- *Offset* del nivel de entrada de 1,5 V.

El rango de tensión aceptado, sin saturación, por el *ADC* en su entrada es de 0 – 3,3 V. El valor obtenido en el registro de resultados del ADC equivale a:

$$\left(\frac{V_{in} - V_{OFFSET}}{3{,}3}\right) * 32768 = R \quad (13)$$

donde V_{in}= Tensión de entrada al *ADC.* Para el prototipo de utilizó un nivel de *offset* de 1,5 V.
El proceso de muestreo realizado, es recomendado por el *SDK* en varios ejemplos arrojando excelentes resultados.

Para mayor información de los registros relacionados con el *ADC* ver la siguiente referencia [Motorola, DSP56f80x User Manuals]. El Temporizador está configurado tal como se expone en el capítulo 5 de la referencia [Motorola, Targeting DSP5680x Platform Manual, 2003]. Las líneas de código del programa se presentan en el Anexo B.

3.5.2 Filtro digital.

El filtrado Digital se realiza implementando la ecuación en diferencias, una vez obtenidos los coeficientes del mismo. Los coeficientes del filtro fueron almacenados en la memoria FLASH de datos en formato Frac32 ocupando 804 direcciones de memoria de 16 bits, de las 2048 disponibles en esta memoria.
Para realizar las operaciones exigidas en esta etapa se utilizaron tres funciones de la biblioteca *DSP* del *SDK,* que son: ***L_mult_ls***(val32,val16), multiplica un valor en Frac32 con otro en Frac16, ***L_add***(val32,val32), suma dos variables en Frac32 y ***round***(val32), que redondea de Frac32 a Frac16. Se debe incluir la cabecera *mfr16. h* y *mfr32.h* en el archivo que contenga estas funciones para su correcto funcionamiento. Las funciones de la biblioteca *DSP* se encuentran en la siguiente referencia [Motorola, SDK, 2003].

3.5.3 Enventanado.

La implementación de las ventanas se realizó mediante la utilización de funciones de biblioteca del SDK [Motorola, SDK, 2003]. que permiten la operación de variables y constantes en Frac16. En el caso de las ventanas se necesita operar con funciones trigonométricas, más específicamente con la función coseno, tfr16CosPix(Frac16 x); para este fin se incluye la cabecera tfr16.h. Es importante saber que la función tfr16CosPix( ), multiplica el valor del argumento de entrada de la función por pi. Los argumentos de entrada de las funciones se almacenaron como constantes en memoria FLASH de datos.

Por ejemplo, la ecuación de la ventana Hanning es:

$$w[n] = \begin{cases} 0.5 - 0.5\cos(\frac{2\pi n}{M}) & 0 \le n \le \mathrm{M} \\ 0, & \text{en el resto} \end{cases}$$

Esta ventana se implementó como se muestra a continuación:

```
for(k=0;k<=63;k++)
        {
        arg = Vect_vent[k];          Vec_vent contiene un vector de 2n/M .
        w = tfr16CosPIx(arg);        Calcula el coseno del argumento.
        a = FRAC16(0.5);             Vuelve a Frac16, el argumento 0,5.
        w1 = mult_r(a,w);            Multiplica la función cos() por la constante.
        w = sub(a,w1);               Resta la constante del producto anterior
        AdcBuffer[k] = mult_r(w,AdcBuffer[k]);   Guarda el resultado en
AdcBuffer
}
```

La implementación de las demás ventanas se describe en el anexo B.

4. VISUALIZACIÓN DE RESULTADOS

Este capítulo aborda la solución que se adoptó para resolver el problema de la visualización de resultados, lo cual se implementó con la pantalla gráfica LCD HYUNDAI HG25601-C. Asimismo se realiza una descripción del manejo de periféricos del *DSP*, el *hardware* que involucra y el desarrollo del código necesario para su implementación, entre otros tópicos. Además se especifica cómo el usuario debe utilizar el prototipo para obtener las gráficas y los valores de los armónicos requeridos, esto teniendo en cuenta que la interfaz del prototipo con el usuario es precisamente esta pantalla.

4.1 INICIALIZACIÓN DE LA PANTALLA

El proceso de inicialización de la pantalla involucra el envío de *bytes* tanto de comandos como de datos, es decir, a través del *DSP* se le envían comandos al circuito integrad controlar las funciones de la pantalla; el integrado recibe las instrucciones dadas por el *DSP*, las ejecuta y las transmite, logrando así que en la pantalla se visualice el resultado del procesamiento hecho previamente.

La descripción exacta del código en lenguaje C se encuentra en el anexo B, cabe mencionar que para programar esta pantalla por medio del *DSP56F801*, se debe estar familiarizado con la programación del mismo, que aunque es en lenguaje C, varía si se trabaja con las funciones del *SDK* ó con el ambiente de trabajo *CODE WARRIOR.*

En este proyecto la programación de la pantalla se basó en la referencia [Amaris & Lopez, 2004]. A partir de esta tesis se reformó y adecuó el código

en lenguaje C para la pantalla; específicamente las funciones de inicialización, configuración de puertos y algunos registros que intervienen en la transferencia de datos, además del control de pulsadores.

Es importante mencionar que el código no se implementó tal cual se había escrito en la referencia, ya que a pesar de hacer uso del mismo procesador, el ensamblador es diferente, pues trabaja con el procesador experto de *METROWERKS*[27]*,* así que fue necesario adaptarlo a las características y funciones de la herramienta de programación de trabajo de este proyecto (*SDK*)

4.2 COMUNICACIÓN ENTRE LA PANTALLA Y EL *DSP*

La comunicación entre la pantalla y el *DSP* se puede llevar a cabo ya sea serial o paralelamente, dependiendo de la disponibilidad de pines que posea el *DSP,* para este proyecto se podía implementar una u otra forma; sin embargo, se optó por la primera, ya que se contaba con la experiencia del proyecto anterior [Amaris & Lopez, 2004]. Aunque esta decisión involucra adicionar *hardware,* pues es necesario agregar un circuito integrado que convierta los datos en serie del *DSP* a paralelo, que es la forma en que opera el SED1330F. El circuito integrado que se utilizó es el 74LS164[28], el cual es simplemente un registro de desplazamiento.

Para poder hacer uso de la comunicación serial con el *DSP* se debe configurar el *SPI* (Interfase Serial de Periféricos), el cual posee 4 pines dedicados o que pueden ser configurados como pines de propósito general,

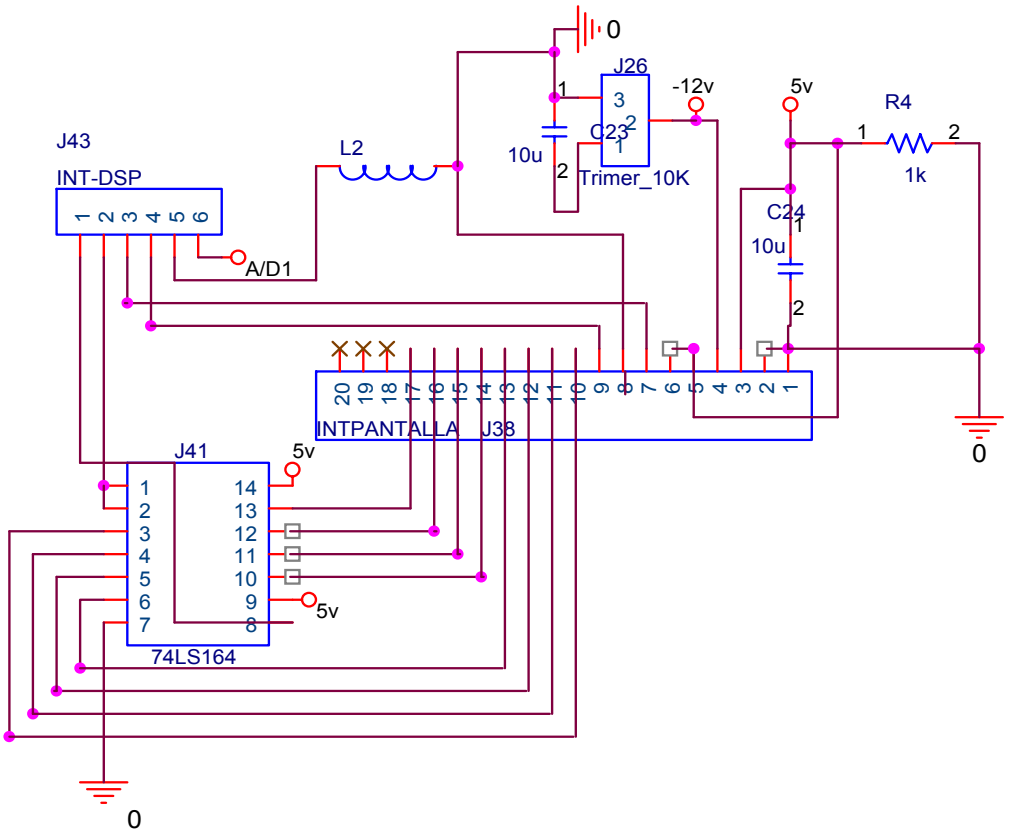
[27] Empresa encargada de producir el *software* para el *CODE WARRIOR*
[28] Fabricado por *TEXAS INSTRUMENTS*

para este caso se utilizaron dos de esos pines dedicados, uno para sincronía con el registro de desplazamiento y el otro para transferencia de datos entre el *DSP* y el registro.

4.3 *HARDWARE* ADICIONAL

Fue necesario implementar *hardware* adicional dado que la pantalla grafica **LCD HYUNDAI HG25601-C,** que requiere una red de condensadores y resistencias, para mejorar su resolución. Esta implementación se hace por recomendación del fabricante, (ver figura 19)

Figura 19. Esquemático del hardware adicional para adecuar el funcionamiento de la pantalla



Por otro lado, para tener acceso a la configuración de la pantalla por parte

del usuario, se dispuso de 3 interruptores que permiten dicha tarea, desplazándose a derecha, izquierda, arriba, abajo, ó actuando como *enter,* dependiendo de la función que realicen en determinado momento, es importante indicar que estos pulsadores no se encuentran en la tarjeta de adquisición, pues se dejan al alcance del usuario, así pues sus conexiones se hacen al aire, un *pin* conectado a tierra y el otro al puerto configurado como entrada del *DSP*.

Como se mencionó previamente, es necesario agregar otro circuito integrado, si se quiere hacer la comunicación en serie, este registro de desplazamiento posee 14 pines, de los cuales 8 son salidas, 1 pin tierra, 1 pin alimentación, 2 de entrada y 2 para el reloj y el *clear*[29]*.* Para mayor información sobre este integrado remitirse a la hoja de datos que se encuentra en el anexo C.

Resulta relevante especificar cuáles puertos del *DSP* se conectan las entradas del registro, el reloj, los controles de la pantalla para escritura (WR^{30}) y el pin de identificación de comando o datos de la pantalla ($A_o{}^{31}$), este tema se expone en la sección siguiente.

4.3.1 Configuración de puertos del *DSP*.

Los pines que se utilizan para la comunicación entre la pantalla y el DSP son básicamente 7, 4 del puerto GPIOB y 3 del puerto GPIOA, siendo todos pines de propósito general en un principio; sin embargo 2 pines del GPIOB se utilizan como pines dedicados, la siguiente tabla hace una descripción de los pines mencionados.

---

[29] Limpia todas las salidas cuando se lleva a cero.
[30] *WRITE*
[31] *Data type select*

Tabla 3 Descripción de puertos del *DSP*

PUERTO	DESCRIPCIÒN
GPIOB4	Pin dedicado (SCLK[32], hace parte del *SPI*), se conecta al reloj del registro de desplazamiento (pin 8).
GPIOB5	Pin dedicado (MOSI[33], hace parte del *SPI*), se conecta a los pines de entrada del registro (Pines 1 y 2 están conectados entre si).
GPIOB6	Pin configurado como propósito general, configurado como salida, se conecta a la pantalla para controlar el pin de escritura WR (pin 7)
GPIOB7	Pin de propósito general, configurado como salida, se conecta al pin A_o de la pantalla, el cual identifica si es un dato o un comando lo que se transmite (pin 9)
GPIOA0	Pin de propósito general, configurado como entrada, es uno de los tres pulsadores que se diseñan para que el usuario controle la pantalla
GPIOA1	Pin de propósito general, configurado como entrada, es uno de los tres pulsadores que se diseñan para que el usuario controle la pantalla
GPIOA2	Pin de propósito general, configurado como entrada, es uno de los tres pulsadores que se diseñan para que el usuario controle la pantalla

En cuanto a la programación se refiere, hay que destacar que se deben configurar los puertos del *DSP* con esta información, la función que realiza esta tarea es llamada Config_GPIOB, y se presenta en el anexo B.

[32] *Serial Clock*
[33] *Master OUT, slave IN*

4.3.2 Pulsadores.

Considerando que la pantalla es la interfaz entre el usuario y el prototipo, es necesario buscar una solución que le permita a éste manipular el equipo, de tal forma que pueda escoger el tipo de ventana para el procesado, iniciar dicho procesado ó moverse dentro de las paginas de presentación de la pantalla, pensando en esto se diseño el prototipo con tres pulsadores normalmente abiertos, a medida que se avanza dentro de las paginas de la pantalla, estos pulsadores adquieren diferentes funciones, que se especifican en la capa de texto de la pantalla, así el usuario tiene total dominio sobre el equipo.

Como se ha mostrado en la tabla anterior, estos pulsadores se conectan al puerto GPIOA del DSP, éste se activa cuando en sus terminales se lee un cero analógico; es decir, que un pin del pulsador va al DSP y el otro a tierra, al cerrarse se lleva un cero a los pines del puerto GPIOA, produciendo así un pulso que le indica al procesador que tarea debe realizar.

4.4 IMPLEMENTACIÓN DEL CÓDIGO

4.4.1 Configuración de periféricos.

Los periféricos que se deben configurar para lograr enviar datos del DSP a la pantalla, son el puerto GPIOB y GPIOA, el SPI y el PLL.

- Puertos GPIOB y GPIOA: Estos pines están conectados a la pantalla y al registro de desplazamiento, por tanto su función es servir de interfaz entre este último y el DSP.
- SPI: Interfaz Serial de Periféricos, se configura para hacer que el procesador envíe datos al registro de desplazamiento de forma serial.

- PLL34: Se configura para disminuir la velocidad del bus de datos a 28MHz para que sea compatible con la velocidad de la pantalla.

4.4.2 Configuración de los pulsadores.

Para configurar los pulsadores de tal forma que realicen las tareas requeridas en cada página de la pantalla, se sigue como base el código presentado en [Amaris & Lopez, 2004] realizando los ajustes respectivos.

4.4.3 Gráficos y texto.

Como se mencionó en el capítulo uno, una de las principales características de esta pantalla es que permite trabajar diferentes capas de gráficos y texto independientes pero simultáneas, en este proyecto se implementó una capa de gráficos y una de texto simultáneas en las páginas de la pantalla. En términos de programación esto se considera en el código principal que está en el anexo B.

Es necesario que los valores que se van a enviar a la pantalla sean descifrables para el SED1330F[35], ya que los resultados que se tienen en el *buffer* de datos del procesador, no lo son (resultados del tratamiento de señal, FFT, filtrado entre otros). Por tal motivo se debe hacer un ajuste que involucra: convertir estos resultados que están en como fija a coma flotante, multiplicarlos por una constante y finalmente llevarlos a un formato que se podría definir *LCD.*

- **Conversión de datos de coma fija a coma flotante.** Para enviar datos a la pantalla y que ésta pueda graficarlos es necesario llevar los datos a coma flotante, en otras palabras, al iniciar el procesamiento se toman las

[34] *Phase Looked loop*
[35] Circuito integrado que se encarga de realizar el proceso de visualización en la pantalla.

muestras que vienen del *ADC* ya normalizadas a 3,3(este es el valor de referencia de todas las muestras tomadas por el conversor analógico-digital) y multiplicadas por 2^{15} (32768), (se debe hacer hincapié en que este ajuste lo realiza el *ADC* directamente), el resultado es un número en formato coma flotante; sin embargo, el procesador interpreta el dato en formato coma fija de 16 *bits* (este cambio entre formatos lo realiza el procesador directamente, es un ajuste llamado *casting*); al finalizar todo el proceso, el valor de las muestras que queda en el *buffer* de datos está de nuevo en coma flotante ó así lo puede interpretar el procesador. Si el lector se interesa por este tipo de conversiones se le recomienda remitirse a la referencia [Stallings, 2000]

- **Multiplicación por una constante.** El resultado que se obtiene después de filtrar, aplicar la *FFT* y el ventaneo a las muestras es el que queda almacenado en el *buffer* de datos antes citado, pero debe ser multiplicado por una constante que ajuste este dato al valor real del armónico. Se debe tener en cuenta al calcular esta variable la normalización que hace el *ADC*, la que hace la pinza al convertir la corriente en tensión y los efectos de *offset* no deseado que presenta la adquisición de datos (básicamente por el *hardware*), al final de estos cálculos si se tiene el verdadero valor de cada armónico a visualizar.

- **Formato *LCD***. En este momento solo resta llevar el resultado final al código *ASCI*[36] para ser interpretado por la pantalla.

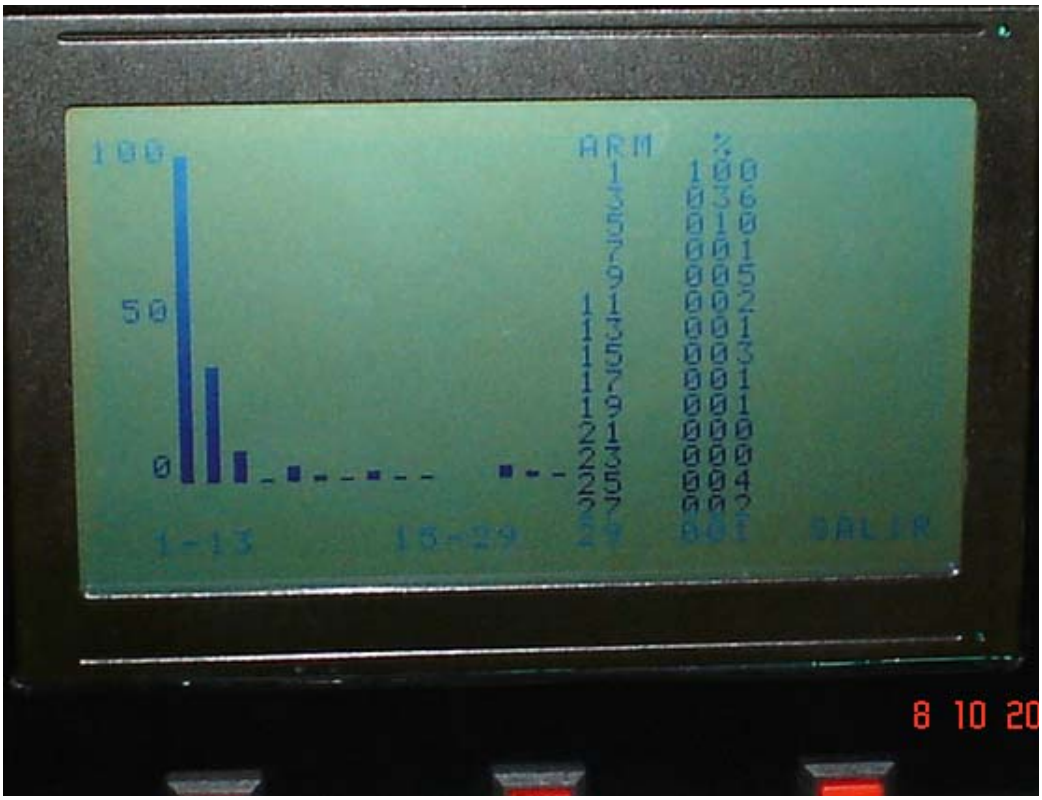
Por otro lado se requiere que la visualización del espectro de frecuencia sea lo más clara posible en cuanto a la amplitud se refiere, para esto se divide el espectro de frecuencia en dos, de tal forma que se obtengan dos imágenes: una desde la componente fundamental hasta el 13^{avo} armónico, y otra desde

[36] Código normalizado de caracteres que interpreta todo procesador

el 15^{avo} hasta el 29^{avo}. Ahora resulta conveniente normalizar los valores de cada gráfico con la magnitud mayor, esto junto con una grafica de barras de todos los armónicos forman una tercera imagen. Finalmente estos valores se deben transmitir sabiendo de ante mano las coordenadas X-Y que debe tener cada uno dentro del gráfico en la pantalla.

Ya se mencionó que se trabaja simultáneamente gráficos y texto, pero no se ha dicho cómo es la distribución; la figura 20 muestra una página de la pantalla para que el lector tenga mayor claridad al respecto.

Figura 20**.** Página 5 de la pantalla de visualización



4.5 Visualización de resultados

La visualización de resultados se presenta mediante una mezcla de gráficos de espectros y datos de texto, que se superponen gracias a las características que posee la pantalla. Se puede observar el espectro en frecuencia, además del valor de la magnitud y fase de todos los armónicos impares hasta el 29vo, la forma de onda en el tiempo no se trabaja en este prototipo.

4.5.1 Uso de la pantalla por el usuario.

En primera instancia al energizar el equipo se observa en la pantalla una página de presentación que contiene el logo de la Universidad Industrial de Santander, el nombre del proyecto, el nombre de los autores, el grupo de investigación al cual pertenece este trabajo y finalmente la sigla E3T, que identifica a la Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. En la figura 21 se muestra esta página de presentación.

Figura 21. Pagina de presentación

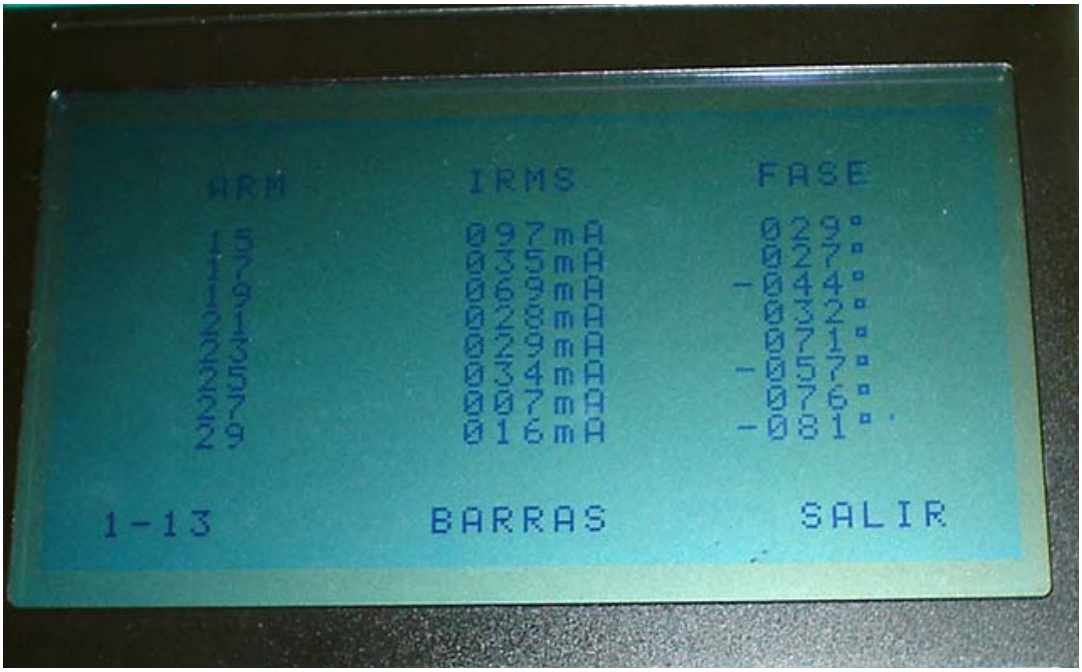
Pulsando cualquier tecla aparece una segunda página, en la cual se presenta un cuadro que permite al usuario configurar a través del pulsador izquierdo el tipo de ventana (Rectangular, *Hamming*, *Hanning,* ó *blackman)* con la que desea realizar el procesado de la señal; una vez se elija la ventana a utilizar, con el pulsador derecho se da la opción de procesar. Las funciones de los pulsadores se muestran en la parte inferior de la pantalla y van cambiando según la función que se desea desempeñar. (Ver figura 22)

Figura 22. Página de configuración

Al elegir procesar desde la página de configuración aparece inmediatamente una tercera página que muestra los resultados del valor RMS de los armónicos 1 al 13 con su respectiva fase. Las funciones de las opciones que aparecen en la parte inferior esta tercera pagina son: Mostar los resultados de los armónicos 15 a 29, con el pulsador izquierdo, mostrar el espectro de los armónicos impares junto con el valor del porcentaje con respecto a la fundamental, con el pulsador del medio y por ultimo volver a la página de configuración, con el pulsador de la derecha.

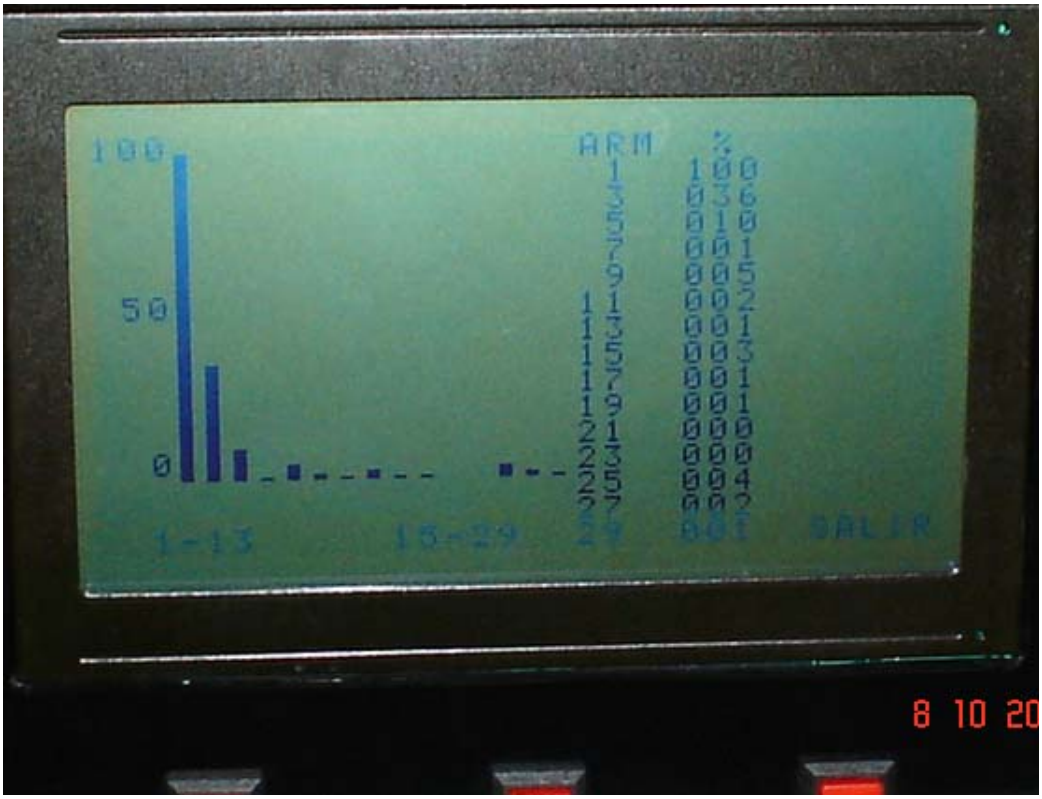
La pagina 4 es idéntica a la pagina tres con la diferencia que los valores mostrados corresponden a los armónicos 13-29. (Ver figura 23)

Figura 23. Páginas de Resultados 3 y 4



La página 5 muestra el espectro y el porcentaje de los armónicos con respecto a la fundamental. En esta página las opciones permiten ir a cualquiera de las páginas de resultados anteriores o volver a la página de configuración. (Ver figura 24)

Figura 24. Página 5 Gráfico del espectro.



5. PRUEBAS Y RESULTADOS OBTENIDOS

En este capítulo se describen los resultados de la estimación de armónicos de corriente, obtenidos con el prototipo en un sistema monofásico; la respuesta de cada una de sus ventanas, así como sus tasas de error con respecto a otro *software* de programación como MATLAB[37] y otro equipo comercial que realiza la misma función. Además del análisis de calibración realizado con señales sintetizadas para armónicos impares.

5.1 ANALISIS DE RESULTADOS

5.1.1 Resultados utilizando señales sintetizadas.

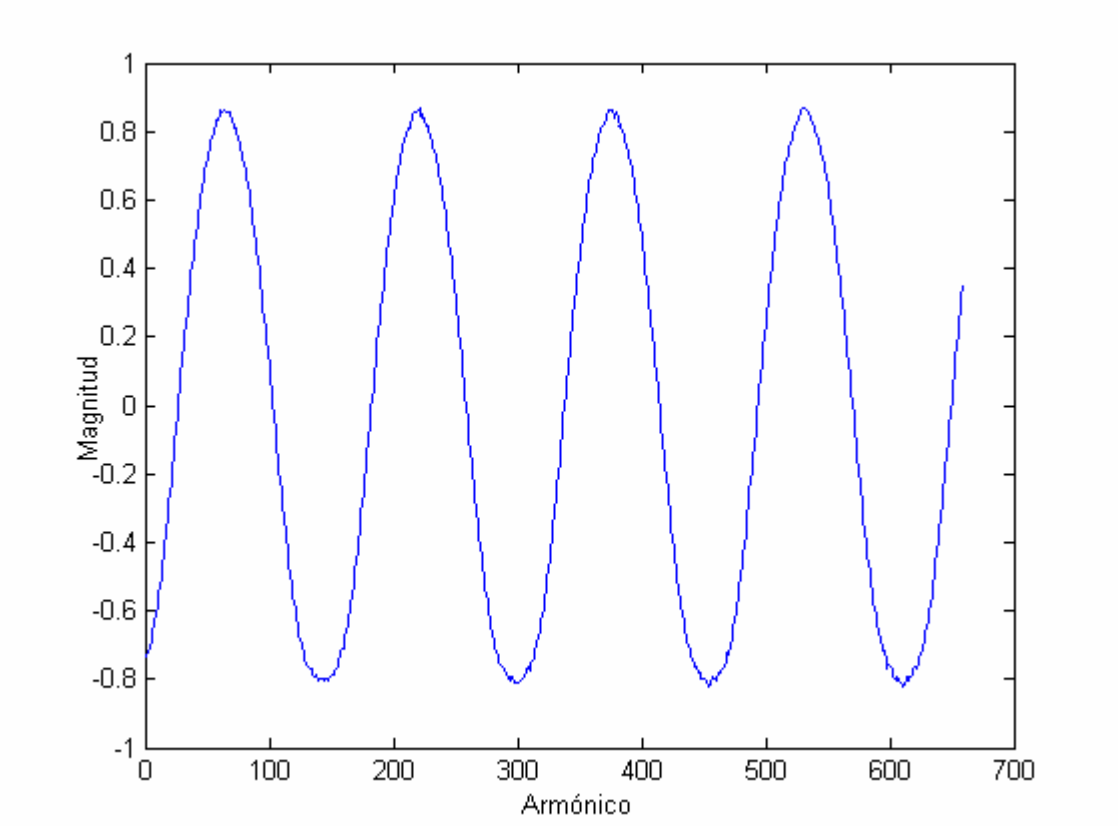
En este tipo de pruebas se evalúa solamente el procesamiento de señal que realiza el DSP; para lo cual se cargó en el buffer del conversor analógico digital (ADC), una señal sinusoidal pura con una amplitud de 2 Vp/p (simulada en MATLAB), esta señal pasa a través del filtro, luego se realiza el enventanado y finalmente la FFT.

Con el fin de realizar un correcto análisis del desempeño del procesador y de cada ventana, se trabajó con dos tipos de señales para cada una; es decir, se simularon señales puras a una frecuencia determinada (fundamental y armónicos impares [Rodríguez & Uribe, 2003]) y señales con un promedio del deslizamiento por encima y por debajo de la frecuencia nominal. (fundamental y armónicos impares [Rodríguez & Uribe, 2003]).

[37] Herramienta de programación y simulación

- **Señales senoidales de frecuencia única**. La señal que se carga en el *buffer* del *ADC* se muestra en la figura 25, cabe mencionar que todas las señales simuladas poseen las mismas características solo varía la frecuencia; esta señal tiene una tensión eficaz (Vrms) de 0.7071 V

Figura 25. Forma de onda señal simulada en MATLAB de 60 Hz, cargada en el *buffer* del *ADC.*



Ahora bien la respuesta del procesador para este tipo de señales arrojó los resultados mostrados en las figuras 26 y 27; que se resumen en la tabla 4 para diferentes frecuencias armónicas.

Figura 26. Error de las ventanas rectangular, *Hanning* y *Hamming*

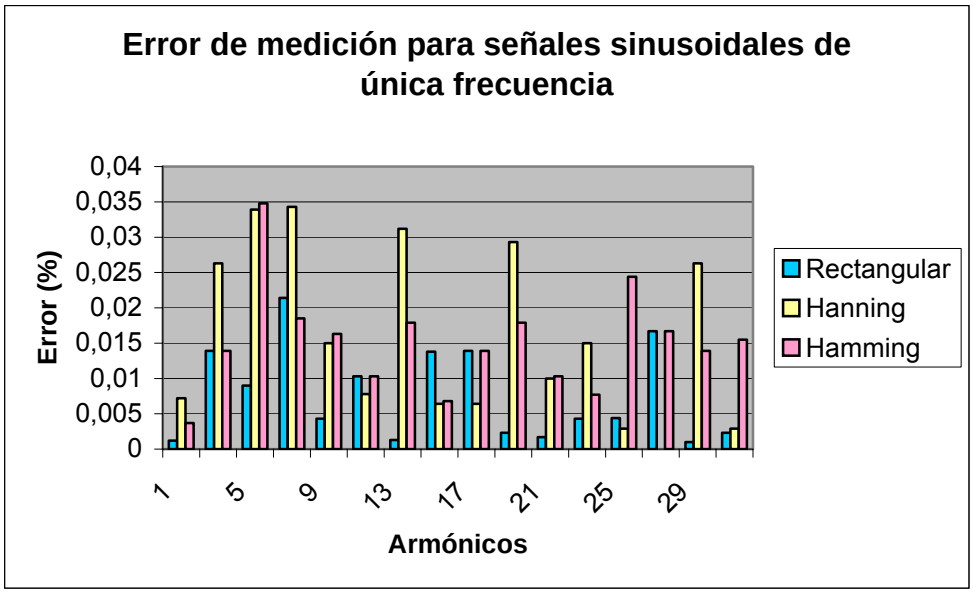


Figura 27. Error de la ventana *Blackman*

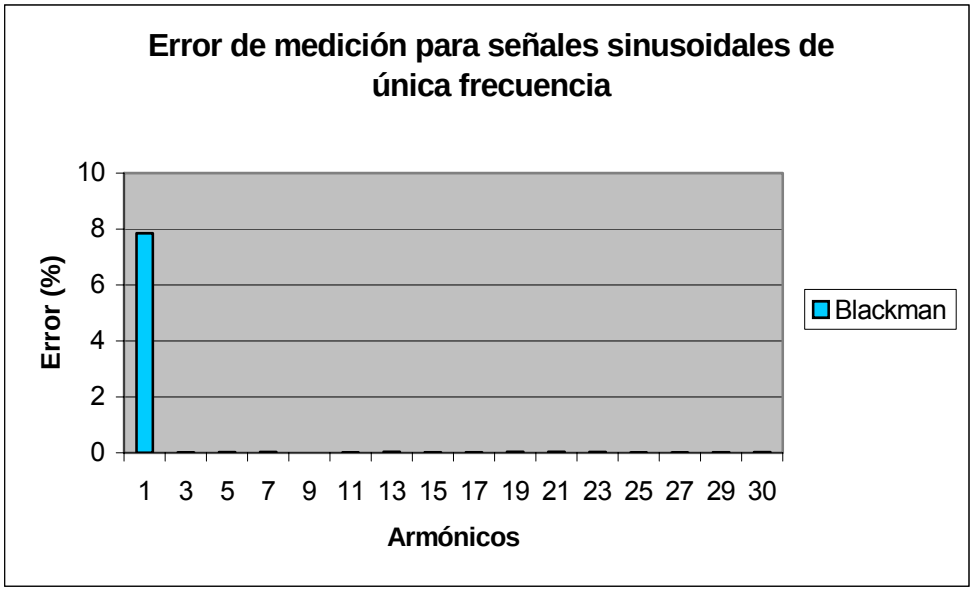


Tabla 4. Error de cada ventana para señales puras, armónicos impares.

VENTANA / ARMONICO	RECTANGULAR		HANNING		HAMMING		BLACKMAN	
	VrmsOut	%Error	VrmsOut	%Error	VrmsOut	%Error	VrmsOut	%Error
60	0,7071	0,0012	0,7072	0,0072	0,7071	**0,0037**	0,7673	7,8478
180(3ro)	0,7070	0,0139	0,7069	0,0263	0,7070	0,0139	0,7070	0,0144
300(5to)	0,7072	0,0090	0,7074	0,0339	0,7074	**0,0348**	0,7074	0,0371
420(7mo)	0,7070	**0,0214**	0,7069	**0,0343**	0,7070	0,0185	0,7069	0,0322
540(9no)	0,7071	0,0043	0,7070	0,0150	0,7070	0,0163	0,7071	0,0010
660(11avo)	0,7070	0,0103	0,7072	0,0078	0,7070	0,0103	0,7072	**0,0070**
780(13avo)	0,7071	0,0013	0,7069	0,0312	0,7070	0,0179	0,7069	0,0362
900(15avo)	0,7072	0,0138	0,7072	0,0064	0,7072	0,0068	0,7072	0,0164
1020(17avo)	0,7070	0,0139	0,7072	0,0064	0,7070	0,0139	0,7072	0,0164
1140(19avo)	0,7071	0,0023	0,7069	0,0293	0,7070	0,0179	0,7069	0,0339
1260(21avo)	0,7071	0,0017	0,7070	0,0100	0,7070	0,0103	0,7069	**0,0363**
1380(23avo)	0,7071	0,0043	0,7070	0,0150	0,7072	0,0077	0,7069	0,0296
1500(25avo)	0,7071	0,0044	0,7071	**0,0029**	0,7069	0,0244	0,7072	0,0082
1620(27avo)	0,7070	0,0167	0,7070	0L0138	0,7070	0,0167	0,7071	0,0081
1740(29avo)	0,7071	**0,0010**	0,7069	0,0263	0,7070	0,0139	0,7072	0,0178
1800(30avo)	0,7071	0,0023	0,7071	**0,0029**	0,7070	0,0155	0,7069	0,0234
1860(31avo)	**0,6688**	**5,7319**	**0,6686**	**5,7616**	**0,6686**	**5,7592**	**0,6131**	**15,341**

Después de observar estos resultados se puede decir que el desempeño del prototipo en cuanto a programación se refiere, es óptimo; aunque, este primer resultado es de señales sin deslizamientos, estos errores no sobrepasan nunca el 0,04%; siendo el error mas alto de 0,0363% en la

ventana *Blackman,* lo cual resulta lógico por las características de esta ventana. Por otro lado, los últimos resultados corresponden al 31^{avo} armónico que se encuentra en al banda de atenuación del filtro digital, por tanto se tienen errores tan grandes en comparación con los demás; sin embargo, este dato no se tiene en cuenta en la estimación pues el prototipo esta diseñado para estimar hasta el 30^{avo} armónico como se puede comprobar.

Es importante mencionar además, que a pesar de tener resultados similares con las 4 ventanas aplicadas, el menor error se consigue con la ventana rectangular, por lo menos en este caso ideal (señales puras sin deslizamiento); este resultado también era de esperarse, pues la ventana rectangular por su respuesta en frecuencia, presenta el lóbulo central mas estrecho, atenuando cualquier posible valor fuera de la frecuencia evaluada, pero por tanto no es conveniente si se presenta deslizamiento.

- **Señales senoidales de frecuencia única con deslizamiento**. En este tipo de prueba se realiza el procesamiento para señales puras con deslizamiento; el cual es de +/-0,0033 seg. De igual forma que con las señales puras, estas son simuladas en MATLAB y luego cargadas en el *buffer* del *ADC,* poseen 2 Vp/p y una tensión RMS de 0,7071 Vrms; la forma de onda en el dominio del tiempo es idéntica a la presentada en la figura 25 y como en el caso anterior solo varía la frecuencia.

La tabla 5 expone los resultados obtenidos con el procesador para éste tipo de señales, de nuevo se evalúan todos los armónicos impares, con respecto al deslizamiento; se visualiza para la componente fundamental y los dos últimos armónicos (son los más críticos), deslizamiento positivo y negativo, con el resto se hace un promedio de los resultados.

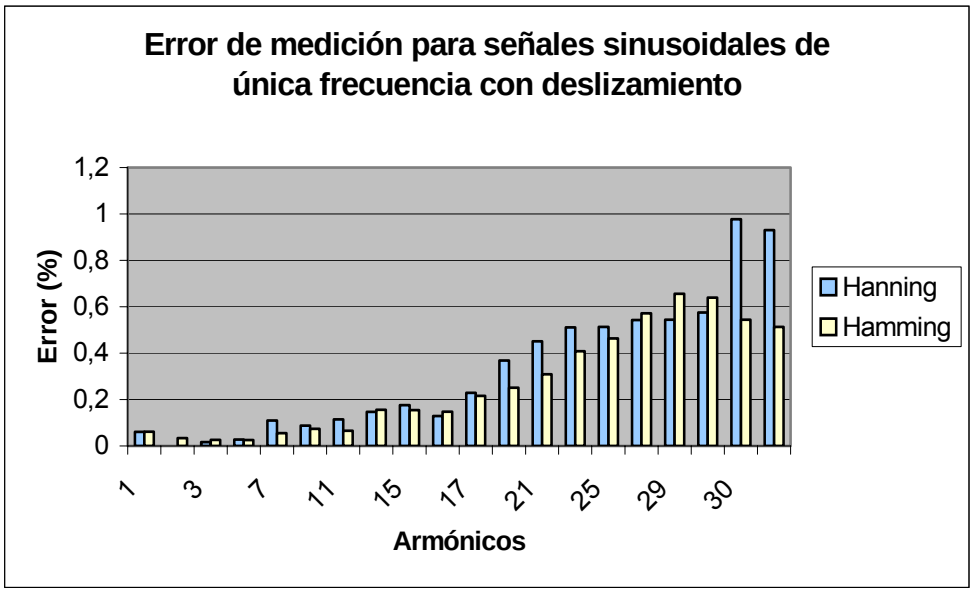
Tabla 5. Error de cada ventana para señales puras desplazadas, armónicos impares.

VENTANA / ARMONICO	RECTANGULAR		HANNING		HAMMING		BLACKMAN	
	VrmsOut	%Error	VrmsOut	%Error	VrmsOut	%Error	VrmsOut	%Error
59.8	0,7061	0,1466	0,7075	0,0608	0,7075	0,0611	0,7695	8,1048
60.2	0,7082	0,1489	0,7071	**0,0002**	0,7069	0,0341	0,7651	7,5754
180,6(3ro)	0,7070	0,0185	0,7070	0,0166	0,7069	**0,0261**	0,7068	0,0507
301(5to)	0,7061	0,1414	0,7069	0,0275	0,7069	0,0255	0,7072	**0,0093**
421,4(7mo)	0,7051	0,2861	0,7063	0,1096	0,7067	0,0552	0,7066	0,0673
541,8(9no)	0,7053	0,2625	0,7065	0,0877	0,7066	0,0735	0,7065	0,0870
662,2(11avo)	0,7059	0,1653	0,7063	0,1148	0,7067	0,0654	0,7066	0,0741
782,6(13avo)	0,7063	0,1132	0,7061	0,1467	0,7060	0,1562	0,7061	0,1363
897(15avo)	0,7036	0,5018	0,7059	0,1762	0,7060	0,1548	0,7073	0,1127
903(15avo)	0,7059	0,1776	0,7062	0,1290	0,7061	0,1475	0,7065	0,0932
1023,4(17avo)	0,7039	0,4534	0,7055	0,2296	0,7056	0,2162	0,7059	0,1663
1143,8(19avo)	0,7011	0,8599	0,7045	0,3681	0,7053	0,2513	0,7056	0,2099
1264,2(21avo)	0,6987	1,2088	0,7039	0,4515	0,7049	0,3092	0,7052	0,2657
1384,6(23avo)	0,6980	1,3090	0,7035	0,5114	0,7042	0,4080	0,7051	0,2865
1505(25avo)	0,6999	1,0346	0,7035	0,5132	0,7038	0,4638	0,7047	0,3473
1625,4(27avo)	0,7034	0,5281	0,7033	0,5427	0,7031	0,5720	0,7041	0,4310
1734,2(29avo)	0,7066	0,0743	0,7033	0,5444	0,7025	**0,6557**	0,7036	**0,5057**
1745,8(29avo)	0,7072	**0,0054**	0,7031	0,5753	0,7026	0,6390	0,7037	0,4811
1794(30avo)	0,6830	**3,5249**	0,7003	**0,9775**	0,7033	0,5438	0,7037	0,4920
1806(30avo)	0,6834	3,4684	0,7006	0,9308	0,7035	0,5137	0,7039	0,4589

Luego de observar el comportamiento del error para estas señales con deslizamiento de frecuencia; se puede determinar que en promedio la ventana que presenta menor error es la *Blackman;* exceptuando los primeros armónicos; sin embargo, las ventanas *Hanning* y *Hamming* están muy cercanas a estos valores; además, entre si, poseen gran similitud en su respuesta. Como era de esperarse la ventana rectangular es la que presenta mayor error, debido a su respuesta en frecuencia.

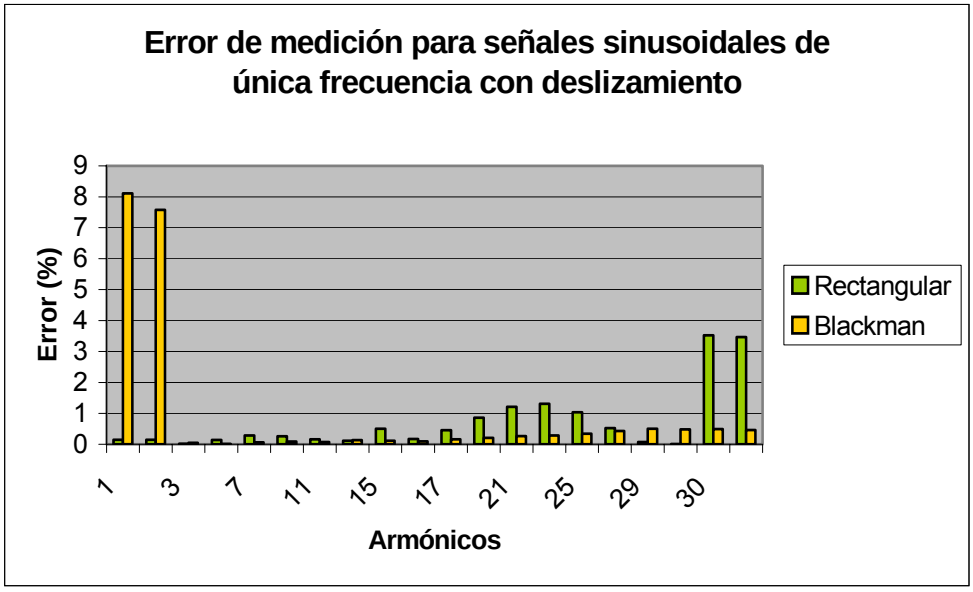
La figura 28 muestra el error de las ventanas *Hanning* y *Hamming,* de forma que el lector pueda hacer una comparación visual del anterior párrafo.

Figura 28. Error de las ventanas *hanning* y *hamming*



Ahora, la figura 29 que sigue hace referencia a los errores de las ventanas restantes.

Figura 29. Error de las ventanas rectangular y *Blackman.*



Otra cuestión a resaltar en este punto, es el uso de ventanas en el procesado de señal; de esta forma se atenúa el efecto de la ventana rectangular de la *FFT* sobre la señal y por supuesto el deslizamiento que necesariamente se ve involucrado en una señal real; aunque, se evidencia que para ciertos armónicos el error de la ventana rectangular es menor que el de las otras, son casos excepcionales no característicos.

Por otro lado también es importante resaltar que el error en general crece con la frecuencia, debido a que el fenómeno de deslizamiento se acentúa a mayores frecuencias.

5.1.2 Resultados utilizando generador de señales.

En esta sección se trabaja de forma similar que en el caso anterior; la diferencia radica principalmente, en que no se carga la señal en el buffer del ADC, sino, que se mide con la tarjeta de adquisición; es decir, se trabaja con el prototipo completo, excepto por la pinza que es reemplazada por el

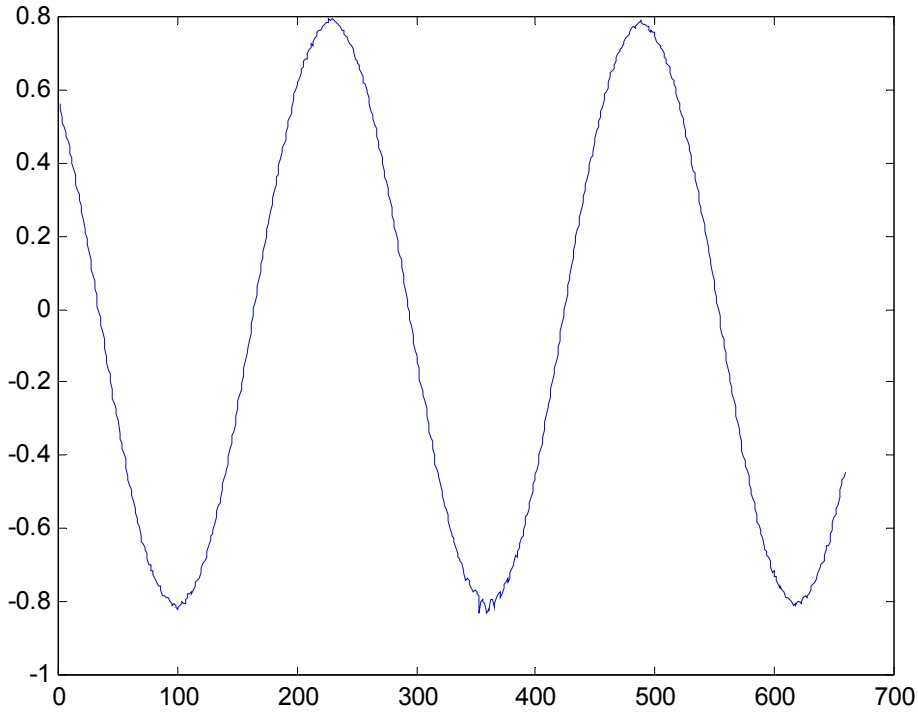
generador de señales, lo que conlleva a que la tensión que entrega el generador pasa a través de la tarjeta de adquisición para llegar a la entrada del ADC. Con este tipo de prueba se busca evaluar el comportamiento del hardware implementado, además del conversor, su respuesta y el error que introduce en el prototipo en comparación con las anteriores pruebas donde se evaluaba únicamente el procesamiento.

De nuevo se plantea la utilización de señales sinusoidales sin armónicos, donde se varía solo la frecuencia de estas señales, para simular cada armónico hasta el 30; se presentan todos los armónicos en este caso. Además se utilizó solo la ventana rectangular, pues el desplazamiento se supone cero con el generador, sin embargo pruebas que se realizaron en laboratorio muestran que hay deslizamiento. Ésta prueba es llamada calibración y se realiza en la evaluación de la mayoría de equipos comerciales.

La figura 30 muestra la forma de onda en el domino del tiempo de la señal producida por el generador, tiene una amplitud en promedio de 0,800 V y una tensión eficaz en promedio de 0,5658 Vrms. Se habla de un promedio en la tensión de entrada porque para cada frecuencia, la amplitud en teoría no debía cambiar; sin embargo, se observó que la inexactitud del generador cambiaba este valor en cada caso.

Los valores de ésta gráfica son tomados del *buffer* del *ADC,* se puede observar que no existe una tensión de *offset,* este punto resultó muy importante en el proceso; pues el error logró reducirse significativamente, al eliminar todo el *offset* que introduce el conversor y la tarjeta de adquisición.

Figura 30. Forma de onda, señal generada.



A continuación en la tabla 6 se muestra el resultado de esta prueba.

Tabla 6. Error utilizando señales generadas.

ARMÓNICO	VENTANA RECTANGULAR		
	VrmsIn	VrmsOut	Error (%)
60	0,5654	0,5647	**0,1340**
180(3ro)	0,5641	0,5589	0,9423
300(5to)	0,5664	0,5655	0,1556
420(7mo)	0,5623	0,5506	2,1368
540(9no)	0,5679	0,5482	3,5910
660(11avo)	0,5682	0,5492	3,4530

780(13^{avo})	0,5698	0,5393	5,6579
900(15^{avo})	0,5631	0,5198	8,3357
1020(17^{avo})	0,5722	0,5141	11,2994
1140(19^{avo})	0,5623	0,5143	9,3465
1260(21^{avo})	0,5658	0,4905	15,3379
1380(23^{avo})	0,5641	0,4823	16,9517
1500(25^{avo})	0,5718	0,4627	23,5751
1620(27^{avo})	0,5670	0,4366	29,8676
1740(29^{avo})	0,5592	0,4139	35,0849
1800(30^{avo})	0,5644	0,3931	**43,569**

Luego de observar estos resultados es posible determinar como se ve afectado el equipo por el *hardware* que se implementó; pues si se compara con los resultados anteriores, donde se realiza la misma prueba, pero simulando las señales, es claro que el error no se encuentra en el procesamiento, sino en la etapa previa para mayor exactitud en el *ADC* y en la fuente de señal, que en este caso es el generador de señales. Con equipos de medición como osciloscopios, se comprobó que la tarjeta de adquisición entregaba al conversor la misma señal del generador, afectada sólo por el *offset* necesario. Ahora bien se deduce que éste error debido al conversor, se presenta por la cuantificación que realiza el mismo*;* para obtener mejores resultados sería conveniente trabajar con un conversor de 16 bits; no obstante este *DSP* posee un conversor de 12 bits, que debería ser óptimo, pero en términos de calidad no es el mejor, inclusive en la siguiente referencia [Motorola, AN1947/D, 2003] se hace alusión a este hecho; el porcentaje de error a causa del generador de señales se debe a que este, no es del todo exacto a frecuencias altas, se puede observar como se incrementa el error a medida que aumenta la frecuencia.

5.1.3 Comparación con los resultados del monitor Reliable Meter Power (RMP)

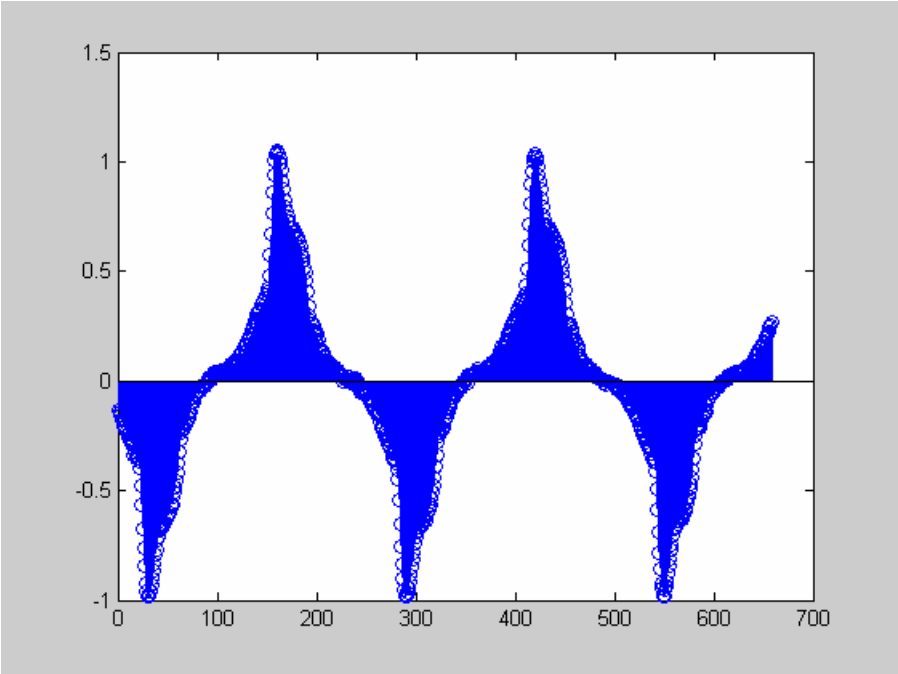
Finalmente se compara la respuesta del prototipo con la respuesta de un equipo comercial que a pesar de tener algunas diferencias en cuanto a la adquisición e inclusive al procesamiento sirvió como referencia para comprobar la eficacia y las limitaciones del equipo diseñado.

El equipo con el que se trabaja es un *Reliable Meter Power* (*RPM*), que pertenece a la empresa Electrificadora de Santander S.A E.S.P; este equipo posee características físicas, de procesamiento y económicas, bastante diferentes al implementado; entre las que están: la dependencia con un computador (para facilidad de estimación preferiblemente portátil), pues es necesario contar con un *software* que realiza el procesamiento mientras se adquiere la señal; por otro lado la adquisición de señal es implementada con un equipo de mayores dimensiones en comparación con el prototipo; no obstante, esto se debe entre otras cosas a los resultados que se obtienen con el *RPM,* pues además de estimar los armónicos de corriente, también estima los armónicos de tensión, estos datos son procesados y almacenados en el computador, siendo esto otra ventaja del *RPM,* cabe mencionar que este equipo realiza estimaciones instantáneas, en un determinado tiempo; es decir, realiza el procesado tomando mas de un ciclo de señal; pero además, permite entregar un promedio de ciertas mediciones realizadas durante un periodo de tiempo que puede variar entre 15 minutos y 2 horas.

Para este caso se tomaron los dos tipos de estimaciones; se realizó una prueba de 15 minutos y 10 mediciones instantáneas para comparar con el resultado del prototipo.

La figura 31 presenta, la señal muestreada, (vista desde el conversor) que estaban midiendo los equipos, para la anterior descripción.

Figura 31. Señal adquirida para las mediciones en volts.(Laboratorio de Redes)



- **Descripción de la prueba**. En el Laboratorio de Redes de la Escuela, edificio de Eléctrica Antigua; se dispuso de 4 computadores con un juego de parlantes para cada medidor; ya que era necesario que los dos medidores estuvieran bajo las mismas condiciones, pues de ello dependía la efectividad de la misma.

Las figuras 32 a 35, muestran los resultados estimados con cada ventana, (Rectangular, *Hanning, Hamming* y *Blackman*) respectivamente, para los armónicos impares y los compara con los obtenidos con el *RPM.*

Figura 32. Resultados medición de armónicos, ventana Rectangular.

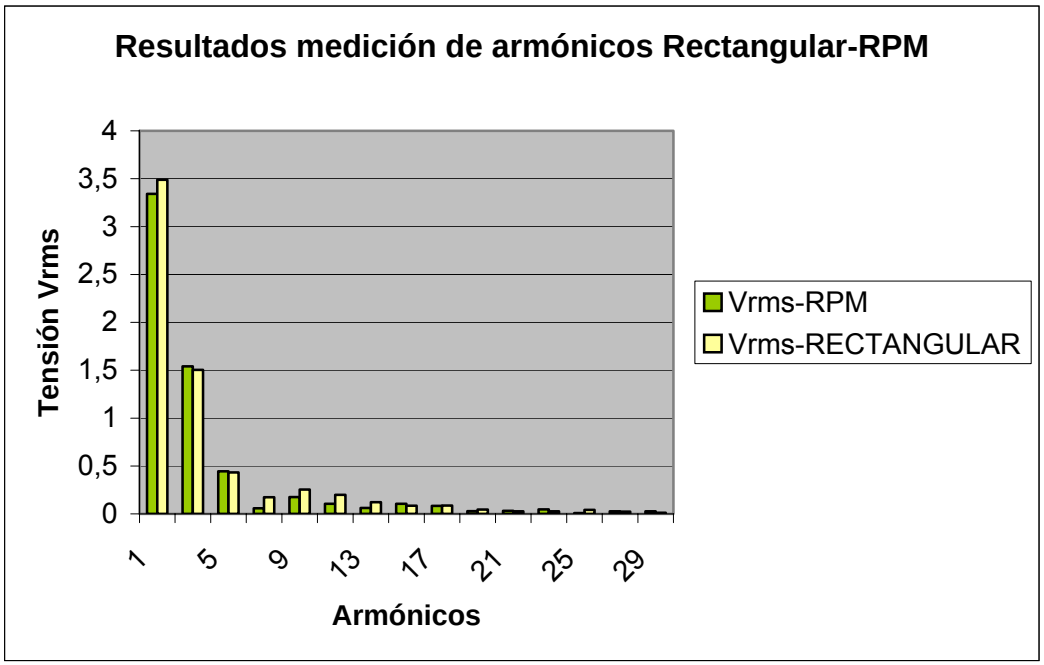


Figura 33. Resultados medición de armónicos, ventana *Hanning.*

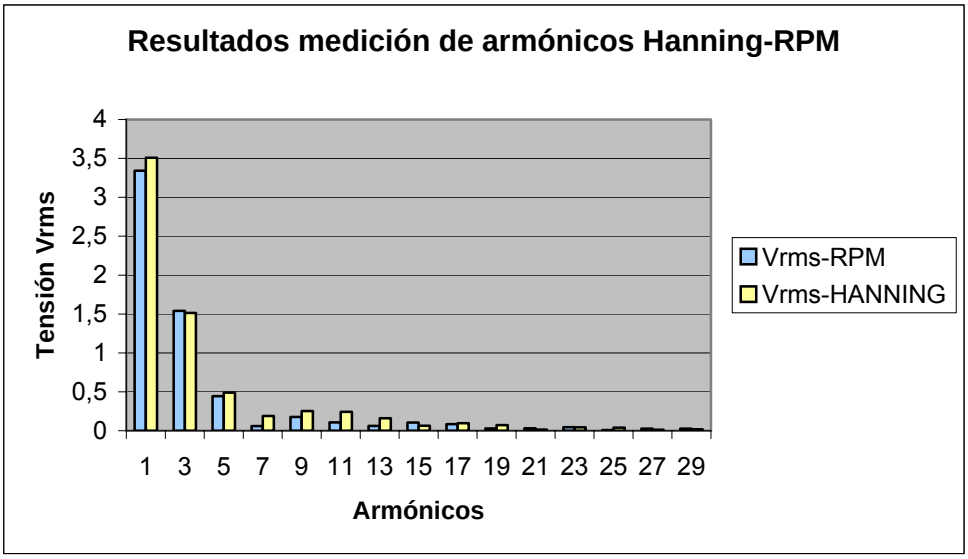


Figura 34. Resultados medición de armónicos, ventana *Hamming.*

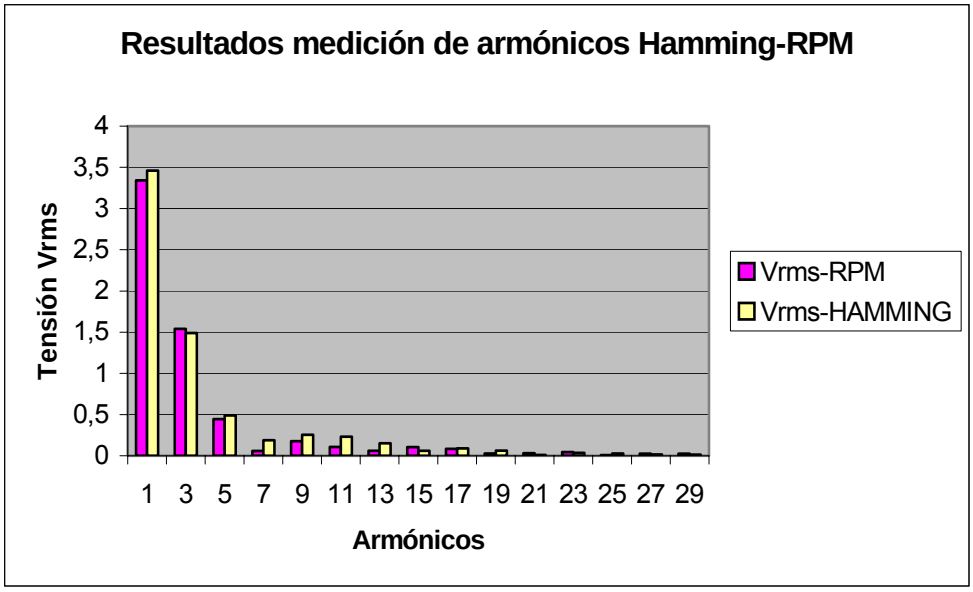
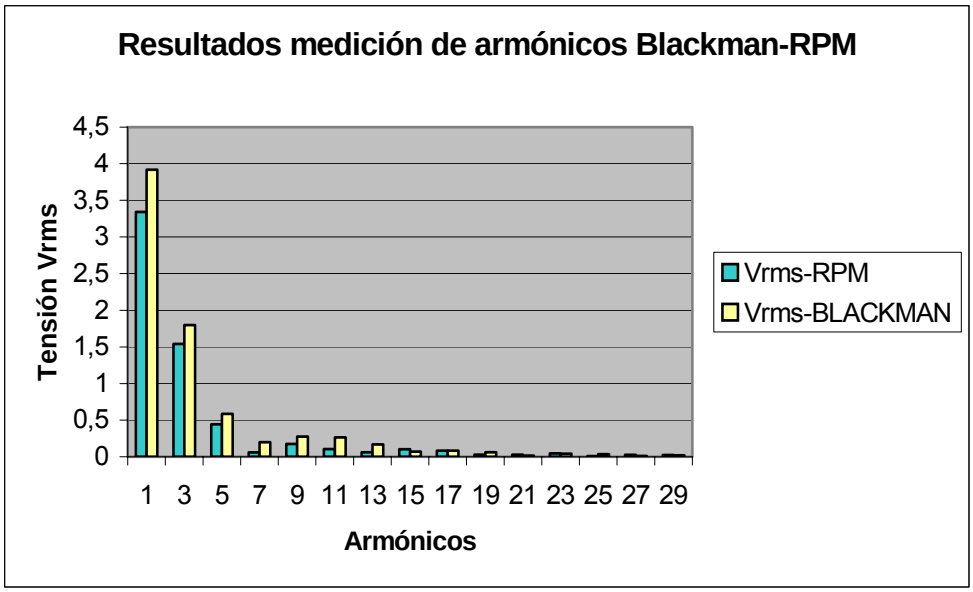


Figura 35. Resultados medición de armónicos, ventana *Blackman.*



La tabla 7 es el resumen de estos resultados; se comparó el valor de cada armónico impar obtenido mediante el prototipo, con los producidos por el *RPM*.

Tabla 7. Error con respecto al *RPM* para cada ventana.

VENTANA / ARMONICO	RESULTADO RPM	RECTANGULAR		HANNING	
		VrmsOut	%Error	VrmsOut	%Error
60	3,3420	3,4891	4,4022	3,5083	4,9756
180(3ro)	1,5400	1,5041	2,3293	1,5129	1,7594
300(5to)	0,4440	0,4335	2,3650	0,4897	10,2899
420(7mo)	0	0	0	0	0
540(9no)	0,1760	0,2547	44,7292	0,2538	44,1946

660(11avo)	0	0	0	0	0
780(13avo)	0,0620	0,1222	97,1271	0,1597	157,571
900(15avo)	0,1050	0,0850	19,0042	0,0630	40,0276
1020(17avo)	0,0840	0,0861	2,4984	0,0964	14,7804
1140(19avo)	0,0290	0,0456	57,1765	0,0726	150,443
1260(21avo)	0,0310	0,0261	15,9036	0,0143	53,778
1380(23avo)	0,0460	0,0268	41,6313	0,0429	6,7191
1500(25avo)	0	0	0	0	0
1620(27avo)	0,0270	0,0240	10,9536	0,0119	55,776
1740(29avo)	0,0260	0,0119	54,1843	0,0188	27,5191

Continua tabla anterior, siguientes dos ventanas.

VENTANA / ARMONICO	RESULTADOS RPM	HAMMING		BLACKMAN	
		VrmsOut	%Error	VrmsOut	%Error
60	3,3420	3,4599	3,5293	3,9202	17,3
180(3ro)	1,5400	1,4897	3,2652	1,7987	16,797
300(5to)	0,4440	0,4898	10,3130	0,5854	31,841
420(7mo)	0	0	0	0	0
540(9no)	0,1760	0,2551	44,9155	0,2763	56.991
660(11avo)	0	0	0	0	0
780(13avo)	0	0	0	0	0
900(15avo)	0,1050	0,0622	40,7962	0,0708	32.5466
1020(17avo)	0,0840	0,0914	8,7775	0,0840	0.0227
1140(19avo)	0,0290	0,0629	116,7247	0,0618	113.123

1260(21avo)	0,0310	0,0112	63,9212	0,0157	49.2725
1380(23avo)	0,0460	0,0358	22,1161	0,0425	7.6718
1500(25avo)	0,0090	0,0291	223,2783	0,0346	284.714
1620(27avo)	0,0270	0,0171	36,5313	0,0107	60.3908
1740(29avo)	0,0260	0,0140	45,9745	0,0216	16.8371

Después de observar las gráficas y las tablas, se puede observar que en algunos armónicos el error se torna significante, esto debido como se dijo anteriormente al problema del conversor; pues en estos casos los amplitudes son insuficientes, (en comparación con otros armónicos), lo cual, los hace muy sensibles al error por cuantificación del *ADC*; sin embargo, no fue posible contrastar con otro equipo comercial, que diera otra referencia, pues el *RPM* también puede presentar errores.

Por otro lado, es poco probable determinar con exactitud que ventana brinda una mejor respuesta, pues es relativo, para ciertos armónicos se tienen óptimos resultados con la *Hanning* pero para otros con la rectangular, así que al establecer cual es la mejor, se puede incurrir en un error; lo que si se puede determinar es cual produce mayor error en general, y para este caso es la ventana *blackman* especialmente en la componente fundamental; este fenómeno se observo a lo largo de todas las pruebas realizadas y según la teoría (comprobado con la práctica) se debe a la componente de continua que adiciona esta ventana en el procesamiento, limitando su exactitud.

6. OBSERVACIONES, CONCLUSIONES Y RECOMENDACIONES

6.1 OBSERVACIONES

Trabajar con integrados de montaje superficial *SOIC* produjo resultados notables en cuanto a exactitud y precisión se refiere, pues la tecnología se ha perfeccionado tanto que su comportamiento se acerca mucho al esperado teóricamente, esto en comparación con los montajes realizados con tecnología *DIP* que no son tan fieles al diseño.

Aunque resulta conveniente trabajar con tecnología *SOIC*, su utilización aumentó los costos de la implementación, pues en Colombia no se consiguieron (casi en un 80%) los implementos utilizados en este proyecto, siendo necesaria su importación, no obstante es importante mencionar que muchas industrias de fabricación de elementos, ofrecen muestras sin costo para este tipo de proyectos.

La respuesta de la tarjeta de adquisición de señal estuvo de acuerdo con los resultados que se esperaban, sin embargo no son exactos a los teóricos; esto se puede entender por el hecho de trabajar con elementos que aunque son de montaje superficial, tienen cierta tolerancia y características no ideales.

Una ventaja notable en la programación de este proyecto fue poder trabajar con un lenguaje de programación de alto nivel como es el C, a pesar de contar con las bibliotecas del *SDK,* se hace necesario poder programar y

compilar en un lenguaje conocido y de fácil adaptación, característica propia del *CodeWarrior.*

La *demoboard* de este *DSP* cumplió una importante tarea en la implementación del prototipo; pues sirvió de fuente de alimentación para la tarjeta de adquisición y la pantalla gráfica, asimismo facilitó el uso de los periféricos del *DSP,* pues cuenta con la implementación física de esos puertos.

Se evidenció claramente la necesidad de realizar sobremuestreo y diezmado digital; sino fuera así, el filtro analógico tendría que ser sumamente exigente y conociendo las limitaciones que se tienen en *hardware*, resulta más conveniente aumentar la complejidad de programación que diseñar este tipo de filtros antisolapamiento.

Por ultimo, una observación importante, es que según las pruebas realizadas con el prototipo, el deslizamiento de frecuencia que se presentó durante las mismas, fue más o menos constante siendo este de 0,003Hz.

6.2 CONCLUSIONES

El uso de una pantalla para visualizar resultados es de gran importancia pues el prototipo queda totalmente independiente del computador, obteniéndose resultados inmediatos en el momento de la medición, sin necesidad de esperar a descargar las imágenes en un computador.

Utilizar la pinza **SONDA AEMC MN375** para sensar corriente fue una buena solución al problema de la adquisición de señal, pues la tensión de salida presentó un comportamiento lineal y en fase con la forma de onda de la señal

de corriente que se sensaba, sin necesidad de introducir más *hardware* que degrade la señal; asimismo la equivalencia en tensión de corriente es alrededor de 1Vrms a 10A; esto es útil si se va a trabajar con un conversor analógico-digital, pues no es necesario atenuar, disminuyendo así el error de implementación.

En cuanto al *Software* de programación que posee el *DSP56F801* se puede concluir que para trabajar con el formato fraccional de 16 *bits,* la respuesta es bastante buena, obteniéndose resultados y tiempos de programación apropiados, no obstante limita en gran medida la programación y compilación a 32 *bits* en el *CodeWarrior,* pues todas las bibliotecas del *SDK* están diseñadas para 16 bits, esto por obvias razones hace que el procesamiento no sea tan rápido como podría llegar a serlo.

Con respecto al ambiente de trabajo, vale la pena resaltar que aunque este ambiente de trabajo permite diferentes tipos de programación; es decir, a bajo y alto nivel, resulta bastante limitado si se pretende analizar los resultados del procesamiento gráficamente, pues no cuenta con este tipo de herramientas, a diferencia de otros ambientes de trabajo que si lo permiten, es necesario recurrir pues a otro tipo de programas como para el caso MATLAB.

Uno de los aspectos más relevantes de este proyecto fue la implementación del filtro *FIR*, aunque estaba en las bibliotecas del *SDK*, su respuesta no era muy favorable, pues presentaba una atenuación de -67db/década en el mejor de los casos, para un valor esperado de -98db/década, esto a causa del formato de 16 *bits;* por esta razón se diseño un filtro *FIR* a partir de la ventana *Kaiser* para procesar a 32 *bits,* con el cual los resultados fueron excelentes pues sus características son idénticas al diseño, conservando una caída de -98db/decáda en 2,04KHz.

El hecho de trabajar con la ventana *Kaiser* se debe a las ventajas que tiene ésta sobre otras como la *Hanning* o la *Hamming* por ejemplo, ya que permite fijar la amplitud y la altura del lóbulo principal, así pues el problema se redujo al cálculo de los coeficientes y a la longitud del filtro. También resulta importante mencionar que aunque los coeficientes del filtro tienen formato de 32 *bits* y que las funciones del *SDK* son para 16*bits*, el error que se presenta por redondeo no es apreciable y además este error se presenta sobre la potencia total y no en cada muestra.

El uso de diferentes tipos de ventana antes de implementar la *FFT,* se debe al desplazamiento en frecuencia que se presenta en cualquier sistema eléctrico, en este proyecto se trabajaron 3 tipos de ventana que atenúan el efecto de la ventana rectangular de la *FFT* sobre la magnitud del armónico a evaluar, concluyendo que el resultado que estas proporcionan es relativo, (así lo demuestran las pruebas realizadas) y por tanto no se puede determinar cual sería la mejor; no obstante si se puede determinar que en promedio la ventana que adiciona mayor componente de continua es la *Blackman*, introduciendo un mayor error en su respuesta.

Se ha hablado de ciertas ventajas y desventajas que posee este equipo, sin embargo hace falta mencionar una desventaja muy importante, relacionada con el desempeño propiamente dicho, pues a pesar de poseer una buena arquitectura este procesador presenta grandes falencias en el conversor, lo cual introduce significativos errores en la medición, especialmente en armónicos de reducida amplitud; en el presente libro se mencionó una referencia bibliográfica que hace énfasis en este hecho, donde se describe que este procesador posee un conversor que fue mejorado notoriamente en las siguientes versiones, éste hecho se vio reflejado en los resultados, pues cuando se evaluó solo el procesamiento se presentaron errores por debajo del 0,04%, al introducir el conversor esta cifra creció considerablemente.

La limitación que se tiene con el prototipo están dadas en primera instancia; al evaluar armónicos pares con cualquiera de las ventanas *hanning, hamming* y *blackman,* pues adicionan componente de continua afectando todos los armónicos, llegando así a resultados erróneos en estos armónicos especialmente; por esta razón no se mencionan en las pruebas, ni se visualizan en los resultados. Asimismo la segunda limitación repercute en la resolución del equipo, que es de 2,5 mA; lo cual quiere decir que el equipo no es lo suficientemente exacto, al estimar cargas de muy poca corriente, ya que en estos valores no se tiene mucha resolución y el error por cuantificación crece; en otras palabras el equipo presenta óptimos resultados si se trabaja por encima de 1 A hasta 10 A, por debajo de este valor los errores se pueden hacer notorios, si se cuenta con una pinza de mayor rango de medición, es posible medir hasta 12 A.

Vale la pena tener presente en cuanto a la capacidad de memoria de almacenamiento, que aunque se tuviese suficiente memoria en este procesador, de tal forma que se pudiese trabajar con 32 *bits* y en coma flotante, la arquitectura que posee no está diseñada para este tipo de formato; es decir, a 32 *bits* la rapidez de procesamiento se vería seriamente afectada, ya que mientras se demora en hacer una operación con 16 *bits* 1 ciclo de máquina por ejemplo, con 32 *bits* necesitaría por lo menos 10 ciclos; en otras palabras existe una relación muy estrecha (pero que debe ser evaluada, si se va a realizar una aplicación específica), entre la velocidad de procesamiento y la precisión y exactitud de los resultados. En este caso se optó más por la velocidad de procesamiento para tiempo real que por la exactitud en los resultados, por demás nada desfavorable con lo cual es posible realizar 12 estimaciones por segundo.

Dadas las ventajas de trabajar con este *DSP* se puede concluir que entre las más relevantes se encuentra la facilidad de independizarlo de su *demoboard,* pues si se compara con otros procesadores del mercado para los cuales esto es dispendioso, con el *DSP56F801* resulta poco complejo, pues cuenta con suficientes puertos y el manejo de periféricos es altamente flexible, además que cargar el programa específico es una tarea sencilla.

En cuanto a costos se refiere para este proyecto se invirtieron solo en *hardware* alrededor de $500.000 lo cual incluye la elaboración de todas las tarjetas necesarias hasta llegar a la tarjeta de adquisición final, además de la tarjeta de alimentación; asimismo, los elementos utilizados en cada caso; incluye también el costo de la *demoboard* y la caja que encierra el prototipo en si. Sin embargo si se piensa en realizar este prototipo en serie su costo solo de *hardware* sería en promedio de $350.000 teniendo en cuenta los costos que conlleva trabajar el *DSP* independiente de su *demoboard;* No obstante en este supuesto no se tiene en cuenta el costo de la pantalla y la pinza, suma que está alrededor de $800.000.

Finalmente para nuestra formación como profesionales realizar el proyecto de grado fue una experiencia gratificante, donde la responsabilidad, la tolerancia, el autoaprendizaje, el trabajo en grupo, el enfrentar y sobrepasar obstáculos fueron la mejor recompensa.

6.3 RECOMENDACIONES

Este proyecto puede mejorarse en pro de futuros trabajos encaminados al uso de procesadores de señales digitales. Como tal, este trabajo pretendió iniciar una investigación en el desarrollo e implementación de equipos basados en este tipo de procesadores, pues es claro el crecimiento que

estos han tenido en los últimos años, así pues la labor apenas comienza y hay mucho por conocer en este campo.

Una de las principales recomendaciones consiste en independizar el *DSP* de su *demoboard,* no porque ésta no tenga utilidad, sino por la presentación del equipo en si; ya no serian necesarias dos tarjetas, pues en una sola se puede tener todo el *hardware* requerido. Ahora bien, si se piensa en independizar el *DSP* se debe hacer con un procesador de mayor capacidad tanto en memoria como en procesamiento, asimismo su conversor debe ser mejor; el *DSP56F801* tiene una capacidad de almacenamiento limitada y aunque el procesamiento con 16 *bits* es satisfactorio, sería más provechoso implementar otro equipo a 32*bits* donde no se sacrifique, ni velocidad ni exactitud. Por otro lado, el implementar el equipo con un procesador de mayor jerarquía permitiría cuatro mejoras más: la primera incorporar al trabajo la estimación de los armónicos de tensión que también son de gran interés para el estudio del sistema eléctrico, claro esta que el solo hecho de adquirir los armónicos de tensión ya es un problema de gran importancia, siendo esta parte del trabajo, tema de un solo proyecto. La segunda, la posibilidad es estimar hasta el armónico 50^{avo} tomando 128 muestras por ciclo y así superar la limitación que se tuvo en este trabajo, pues solo se pudo estimar hasta el armónico 30^{avo} es decir tomando 64 muestras por ciclo, sin posibilidad de realizar un promedio entre varios ciclos. La tercera realizar un promedio de los resultados de varios ciclos. Y la cuarta disminuir el error que se presentó en este prototipo debido al conversor en si. Para estos efectos se recomienda trabajar con el *DSPf56827,* ya que este procesador reúne todas las recomendaciones antes mencionadas.

Si se implementa un nuevo prototipo, independizando el procesador; sería favorable adicionar una memoria externa al diseño, de modo que se pueda salvar en el computador los resultados que arroja el procesamiento con el

DSP, pues en este prototipo eso no fue posible, debido a que la *demoboard* no posee memoria externa.

Implementar un algoritmo cuya función sea disminuir el error que se presenta por deslizamiento, de tal forma que la frecuencia de muestreo cambie según cambie la frecuencia fundamental del sistema.

Realizar un estudio de la inmunidad al ruido que se presenta en la etapa de adquisición, de modo que se garantice la exactitud en próximos desarrollos

Como mejora en la visualización se recomienda trabajar con una pantalla que posea mayor resolución por lo menos en la capa de texto; aunque el trabajo con esta pantalla cumplió a cabalidad las necesidades del proyecto especialmente por el manejo de texto y gráfico simultáneo, no está demás poder graficar la onda en el dominio del tiempo, esto daría otra perspectiva al desarrollo del equipo.

Finalmente si se piensa en la alimentación del equipo seria conveniente hacerlo con una batería de forma tal que el equipo sea totalmente portátil, pues ya es totalmente independiente del computador.

Como una última recomendación sería favorable que en la universidad se continuara trabajando en el diseño e implementación de equipos que utilicen elementos *SOIC* pues en este momento es la tecnología de mayor utilización por lo menos en nuestro medio, sino podemos ir a la mano con la tecnología de países industrializados tampoco sería permisible quedarnos en el pasado y menos en una carrera como Ingeniería Electrónica en la UIS.

Bibliografía

[Amaris & Lopez, 2004] AMARIS Jean Pierre y LOPEZ José Alberto. Elaboración del software para la caracterización de una celda electromagnética utilizando DSP familia 56800 de Motorota. Bucaramanga, 2003 Trabajo de grado (Ingeniero Electrónico). Universidad Industrial de Santander. Facultad Físico Mecánica. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones.

[IEC 61000-4-7, 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 estas. 32 p., Ginebra, Suiza, 1991.

[Dahnoun, 2000] DAHNOUN Naim. *Digital Signal Processing Implementataion using the TMS320C6000™ DSP Platform.* 1 Ed. England: Pearson Education Limited, Prentice Hall 2000

[DSP56800 Hardware Interface Techniques]. DSP56f80x User Manuals Motorola *digital dna, MCU-DSP 56800 Accelerated Development System. Technical Documentation, 56800 Documenation (PDF), Technical Articles*, Application Notes, AN1920/D, 06, 2001. (Este material hace parte del software de soporte que entrega Motorola por la adquisición del *DSP56F801*)]

[Duarte, Ordoñez, 2003] DUARTE Cesar y ORDOÑEZ Gabriel. Algoritmos para la estimación de eventos de estado estacionario en sistemas de energía eléctrica. En: SIMPOSIO DE TRATAMIENTO DE SEÑALES, IMÁGENES Y VISIÓN ARTIFICIAL. Ponencias del VIII Simposio de Tratamiento de Señales, Imágenes y Visión Artificial. Pp. 138-144 Medellín, 2003.

[Duarte, 2004] DUARTE GUALDRÓN Cesar Antonio. Técnicas de procesamiento de señales para la monitorización de la calidad de la energía eléctrica. Bucaramanga de 2004. Trabajo de investigación maestría en Potencia Eléctrica. Universidad Industrial de Santander. Facultad Físico Mecánica. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones.

[GRANADOS, *et al*]. FONSECA Pablo, GRANADOS Javier, PORRAS Claudio. Obtención de curvas características de la maquina de corriente continua en forma autónoma. Bucaramanga, 1999 Trabajo de grado (Ingeniero Electrónico). Universidad Industrial de Santander. Facultad Físico Mecánica. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones.

[Metrowerks, *Targeting_DSP56800,* 2003] *Metrowerks, CodeWarrior, CodeWarrior manuals, pdf, Targeting_DSP56800.* 2003 (Este Pdf se encuentra en la ayuda del software Metrowerks, que entrega Motorola por la adquisición del *DSP56F801*)

[Metrowerks, *IDE_5.1_Users_Guide,* 2003] *Metrowerks, CodeWarrior, CodeWarrior manuals, pdf, IDE_5.1_Users_Guide*. 2003 (Este Pdf se

encuentra en la ayuda del software Metrowerks, que entrega Motorola por la adquisición del *DSP56F801*)

[Motorola, Dsp56f800dbum, 2003] *Motorola digital dank, MCU-DSP 56800 Accelerated Development System. Technical Documentation, Other Documentation, Demo Board user's manuals*. 2003 (Este material hace parte del software de soporte que entrega Motorola por la adquisición del *DSP56F801*)

[Motorola, into 56F8001 Arquitecture, 2003] *Motorola digital dna, MCU-DSP 56800 Accelerated Development System. Training modules, into 56F801 Architecture (multimedia).* 2003 (Este material hace parte del software de soporte que entrega Motorola por la adquisición del *DSP56F801*)

[Motorola, *DSP56F801 User Manuals*, 2003] *Motorola digital dna, MCU-DSP 56800 Accelerated Development System. Technical Documentation, 56800 Documenation (PDF), DSP56800 Processors, Manuals, DSP56f80x User Manuals Addendu (pdf).* 2003 (Este material hace parte del software de soporte que entrega Motorola por la adquisición del *DSP56F801*)

[Motorola, *AN1936/D (pdf),* 2003] Motorola *digital dna, MCU-DSP 56800 Accelerated Development System. Technical Documentation, 56800 Documenation (PDF), Technical Articles, Applications Notes, AN1936/D (pdf).* 2003 (Este material hace parte del software de soporte que entrega Motorola por la adquisición del *DSP56F801*)

[Motorola, *DSP56F801FA60*] Motorola *digital dna, MCU-DSP 56800 Accelerated Development System. Technical Documentation, Data Sheets DSP56F801FA60.* (Este material hace parte del software de soporte que entrega Motorola por la adquisición del *DSP56F801*)

[Motorola, *DSP56f800 Family Manuals,* 2003] Motorola *digital dna, MCU-DSP 56800 Accelerated Development System. Technical Documentation, 56800 Documenation (PDF), DSP56800 Processors, Manuals, DSP56f800 Family Manuals.* 2003 (Este material hace parte del software de soporte que entrega Motorola por la adquisición del *DSP56F801*)

[Motorola, *view pdf,* 2003] Motorola *digital dna, MCU-DSP 56800 Accelerated Development System. Technical Documentation, 56800 Documenation (PDF), SDK Programmer's Guide, view pdf.* 2003 (Este material hace parte del software de soporte que entrega Motorola por la adquisición del *DSP56F801*)

[Motorola, *CodeWarrior (PDF),* 2003] Motorola *digital dna, MCU-DSP 56800 Accelerated Development System. Technical Documentation, 56800 Documenation (PDF) CodeWarrior (PDF)*. 2003 (Este material hace parte del software de soporte que entrega Motorola por la adquisición del *DSP56F801*)

[Motorola, SDK, 2003] Motorola *digital dna, MCU-DSP 56800 Accelerated Development System. Technical Documentation, 56800 Documenation (PDF), SDK software solutions, Signal processing, view PDF. 2003* (Este material hace parte del software de soporte que entrega Motorola por la adquisición del *DSP56F801*)

[Motorola, DSP56f80x User Manuals] Motorola *digital dna, MCU-DSP 56800 Accelerated Development System. Technical Documentation, 56800 Documenation (PDF), DSP556800 Processors, Manuals,* DSP56f80x User Manuals, capitulo 9, 2003. (Este material hace parte del software de soporte que entrega Motorola por la adquisición del *DSP56F801*)

[Motorola, Targeting DSP5680x Platform Manual, 2003] Motorola *digital dna, MCU-DSP 56800 Accelerated Development System. Technical Documentation, 56800 Documenation (PDF), Software Solutions,* Targeting DSP5680x Platform Manual, Capítulo 5, 2003. (Este material hace parte del software de soporte que entrega Motorola por la adquisición del *DSP56F801*)

[Motorola, AN1947/D, 2003] Motorola *digital dna, MCU-DSP 56800 Accelerated Development System. Technical Documentation, 56800 Documenation (PDF), technical articles. Apliacation notes. AN1947/D.* (Este material hace parte del software de soporte que entrega Motorola por la adquisición del *DSP56F801*)

[Oppenheim, Schafer, 2000] Alan V. OPPENEHIM y Ronald W. SCHAFER. Tratamiento de señales en tiempo discreto. 2 Ed. Madrid: Prentice Hall, 2000

[Ordoñez, 2002] ORDOÑEZ Gabriel. Monitorización de la calidad del suministro y consumo de la energía eléctrica. EN: SIMPOSIO USO EFICIENTE Y CALIDAD DE LA ENERGÍA ELÉCTRICA. Conferencia, Universidad Autónoma del Occidente. 2002

[Proakis, Manolakis 98] PROAKIS, J. G. y MANOLAKIS, D. G. Tratamiento digital de señales. 3 ed. Madrid: *Prentice Hall*, 1998.

[Rodríguez & Uribe, 2003] RODRIGUEZ Juan Carlos y URIBE Olga. Propuesta de norma tecnica colombiana de medición de armónicos. Bucaramanga, 2003 Trabajo de grado (Ingeniero Electricista). Universidad Industrial de Santander. Facultad Físico Mecánica. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones.

[Sedra, Smith, 2002] Adel S. SEDRA y Kenneth C. SMITH. Circuitos Microelectrónicas. 4 ed. México: Litografía Eros S.A de CV, Oxford University Press Inc, 2002

[Stallings, 2000] STALLINGS William. Organización y arquitectura de computadores. 4 Ed. Prentice Hall, 2000

ANEXO A
COMPONENTES DE UN PROYECTO CREADO EN SDK

En éste anexo se muestra como se conforma el área de trabajo en la cual se realizó la programación del *DSP*. Para empezar, la Demoboard 56F800 de 60MHz viene con dos instaladores, el de *CodeWarrior™ Development Studio for Motorola 56F800 Demo Board, versión 5.1* y el del *Software Development Kit (SDK), versión3.0.*

El *CodeWarrior* es un ambiente de desarrollo que integra transparentemente el manejo de proyectos mediante: un sistema de construcción de proyectos, un editor, un compilador, un enlazador y un depurador. El *CodeWarrior* no incluye funciones matemáticas para datos en coma flotante como por ejemplo el seno, coseno y la raíz cuadrada (sqrt); funciones necesarias en la mayoría de aplicaciones de tratamiento digital de señales.

A.1 Descripción del *SDK*

El *Software Development Kit (SDK),* sirve de complemento al *CodeWarrior* ya que provee una infraestructura de librerías que permite desarrollos más eficientes, usando un lenguaje de alto nivel, además de hacerlos compatibles entre distintas plataformas.

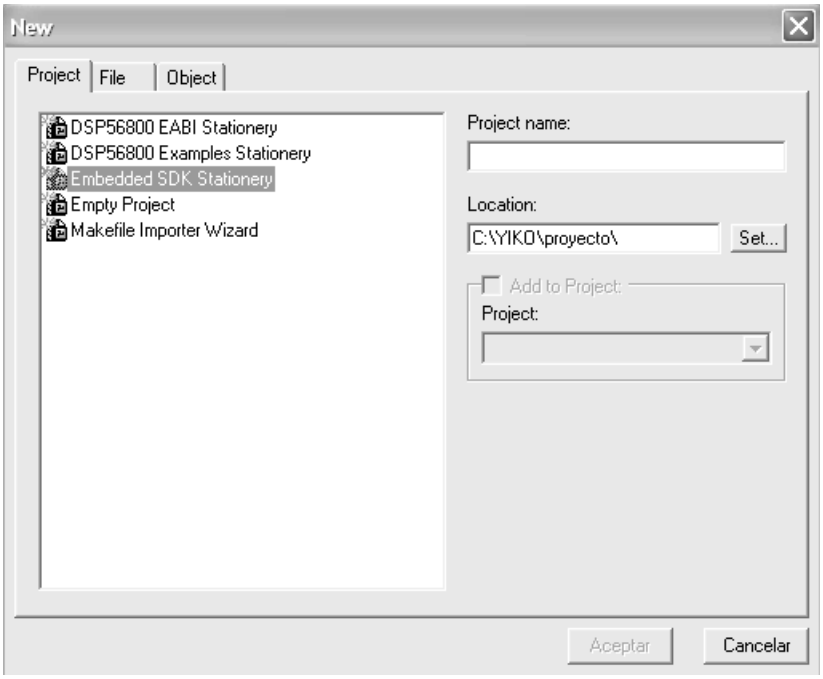
Las librerías que incluye el *SDK* permiten:

- Uso de periféricos por medio de drivers.
- Manejo de interrupciones.
- Manejo de memoria.

- Funciones con rutinas de tratamiento de señales digitales.

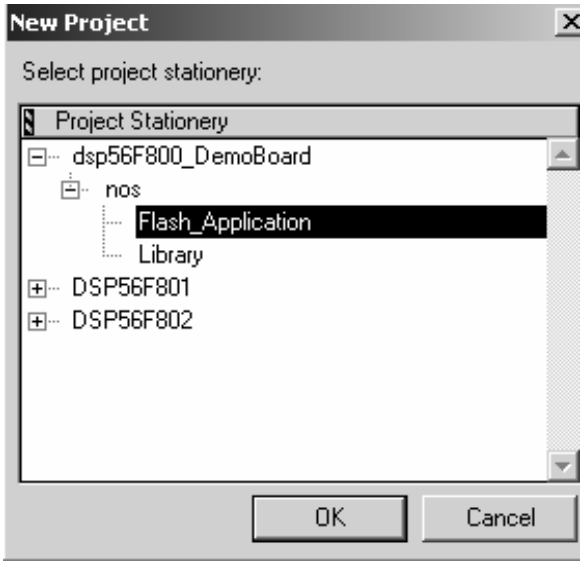
Al instalar el *SDK* se habilita la opción de crear un nuevo tipo de proyecto (*Embedded SDK Stationary*) tal como se aprecia en la Figura A.1.

Figura A.1 Ventana proyecto nuevo con *SDK*.



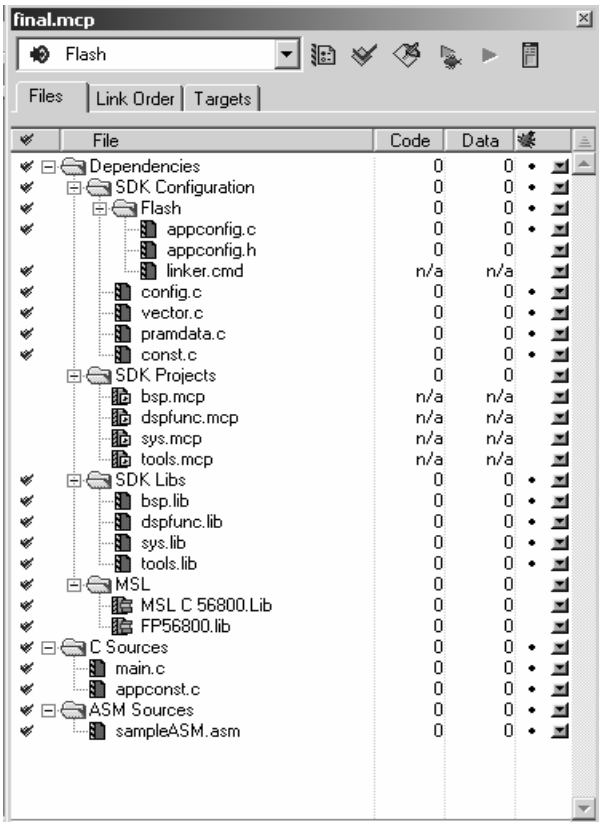
Después de escribir el nombre que se desee dar al proyecto (*final.mcp* en este caso), se elige el tipo de procesador con el cual se está trabajando, para el caso de este proyecto se eligió la opción *Demoboard* tal como se ve en la Figura A.2.

Figura A.2 Selección del tipo de proyecto



Una vez seleccionado el tipo de proyecto a realizar, el SDK construye el proyecto vinculando un conjunto de archivos que tienen una finalidad específica dentro de la aplicación. El objetivo de cada archivo se explicará a continuación y su disposición dentro del proyecto se muestra en la Figura A.3.

Figura A.3. Conformación de un proyecto con *SDK*



El *SDK* organiza el la estructura del proyecto de *CodeWarrior* (final.mcp) de la siguiente forma:

- La carpeta *Dependencies* que contiene todas las dependencias que tiene el proyecto con *SDK* y *Code Warrior*. La carpeta *Dependencies* está dividida de la siguiente forma:
    - *SDK configuration*, contiene archivos que permiten al usuario acceder a la configuración del proyecto.
    - *SDK Projects* contiene los proyectos elaborados en el *SDK* que permiten al usuario modificar y reconstruir librerías.
    - *SDK Libs* contiene el módulo de librerías de SDK.

- *MSL* contiene las librerías estándar de *Metrowerks (**L**ibraries **S**tandard **M**etrowerks)*.

- *C Sources* contiene los archivos fuente de la aplicación, creados por el usuario, escritos en lenguaje de alto nivel.
- *ASM Sources* al igual que *C sources* contiene archivos creados por el usuario pero escritos en lenguaje *Assembler*.

A.2 *SDK configuration*

SDK configuration está compuesto de los siguientes archivos: config.h, config.c, configdefines.h, const.h, const.c, pramdata.h, pramdata.c y vector.c.

A.2.1 config.h y config.c.

La inicialización y configuración de los componentes del SDK residen en el archivo *config.c* el cual inicializará solamente las componentes que han sido incluidas en y definidas en *config.h*. Las componentes definidas en *congig.h* dependen del procesador DSP568xx usado en la aplicación.

A.2.2 configdefines.h.

En muchos casos, un componente de *SDK* depende de otro componente para su correcta operación. El *configdefines.h* incluye automáticamente estas dependencias.

A.2.3 const.h y const.c.

Los archivos const.h y const.c contienen las constantes que necesita el SDK para inicializar y que pueda distinguir entre los datos que pueda tener en memoria flash y los que deberian ser copiaados en memoria RAM durante el arranque.

A.2.4 pramdata.h y pramdata.c.

A veces es útil en aplicaciones, poner parte de las variables del programa o todas en memoria RAM de programas en lugar de la memoria RAM de datos, los archivos pramdata.h y pramdata.c se encargan entonces de aislar datos del SDK y destinarlos a la memoria Ram de programas.

A.2.5 vector.c.

vector.c contiene la tabla del vector de interrupciones del procesador que este usando.

A.2.6 appconfig.c y appconfig.h.

appconfig.h permite al usuario reconfigurar las opciones que vienen por defecto en el archivo config.h, mientras que el archivo appconfig.c, provee un mecanismo que permite al usuario obtener el control sobre la ejecución de la aplicación antes y después de la ejecución del main.

A.2.7 Linker.cmd .

El archivo linker.cmd configura la memoria del procesador. Este archivo utiliza directivas propias de codewarrior que se explican de forma clara en el [Metrowerks, *Targeting_DSP56800,* 2003]

A.3 *SDK projects*

El SDK projects contiene archivos tipo proyecto que permiten al programador modificar y reconstruir las librerias. Dichos proyectos son:

- bsp.mcp, Board Support Peripheral, prove el mismo software de aplicacion API para el mismo Hardware en distintas plataformas garantizando la portabilidad entre aplicaciones.
- sys.mcp, System Support project. incluye tipos básicos de datos, estructuras que se ajustan a la arquitectura del dispositivo, manipulación

de bits, Temporizadores, manejo de memoria y servicios de entrada y salida.

- Tools.mcp Tools provee manejo de Buffers circulares First In First Out, Chequeo de la pila y servicios de conteo de ciclos.

ANEXO B

CÓDIGO DE PROGRAMACIÓN

B.1 ARCHIVO MAIN.C

```
#include "port.h"
#include "dspfunc.h"
#include "arch.h"
#include "fcntl.h"
#include "periph.h"
#include "types.h"
#include "qtimerdrv.h"
#include "quadraturetimer.h"
#include "stackcheck.h"
#include "mfr16.h"
#include "mfr32.h"
#include "vfr16.h"
#include "appconst.h"  /*RECONOCE EL VECTOR DE COEF*/
extern void CallbackOnCompare1(qt_eCallbackType CallbackType, void* pParam);
void filtrado();
void Resultados(UWord16 ventana, UWord16 mostrar);
void Convermod(float Mod, UWord16 linea);
void Converfase(float Fase,UWord16 linea);
void ConverDatos2(float Porcentaje,UWord16 linea);
int  comparar(float res,UWord16 direccion);
void Enventanado(UWord16 ventana);
void EnviarDato(UWord16 dato);
void EnviarComando(UWord16 comando);
void PosCursor(UWord16 pos);
void MoverCursor(UWord16 Mov);
void ImpValor(UWord16 val,UWord16 pos, UWord16 tamano, UWord16 direc);
void ImpVectorTex(const UWord16 texto[],UWord16 pos,UWord16 tamano,
UWord16 direc);
void LimpiaTexto(void);
void LimpiaGrafico(void);
void Config_GPIO();
void Config_Pll(UWord16 t);
void Config_SPI();
void Inic_LCD(void);
void Portada();
void Config_ADC();
bool Izda(void);
bool Medio(void);
```

```
bool Dere(void);
void MicroSeg(UWord32 t);
float sqrt(float x);
Frac16 AdcBuffer[659];                  /* 659 muestras */
CFrac16 FFTInplaceBuf[32];  /* Resultados de la FFT */
Word16 cont_m;
const qt_sState quadParam1 = {

   /* Mode = */                  qtCount,
   /* InputSource = */           qtPrescalerDiv1,
   /* InputPolarity = */         qtNormal,
   /* SecondaryInputSource = */  0,

   /* CountFrequency = */        qtRepeatedly,
   /* CountLength = */           qtUntilCompare,
   /* CountDirection = */        qtUp,

   /* OutputMode = */            qtAssertWhileActive,
   /* OutputPolarity = */        qtNormal,
   /* OutputDisabled = */        1,

   /* Master = */                0,
   /* OutputOnMaster = */        0,
   /* CoChannelInitialize = */   0,
   /* AssertWhenForced = */      0,

   /* CaptureMode = */           qtDisabled,

   /* CompareValue1 for 15360Hz = */  1953,
   /* CompareValue2 = */         0,
   /* InitialLoadValue = */      0x0000,

   /* CallbackOnCompare = */     { CallbackOnCompare1, 0 },
   /* CallbackOnOverflow = */    { 0, 0 },
   /* CallbackOnInputEdge = */   { 0, 0 }
};


/*****************************************************************************
*******/
main()
{
        UWord16 i;
        Result res;
        UWord16 Timer1;
```

```
        UWord16 ventana = 0;
        UWord16 mostrar = 0;
        UInt16 options = FFT_SCALE_RESULTS_BY_N;
        dfr16_tRFFTStruct RFFT;
   dfr16_tRFFTStruct *pRFFT = &RFFT;
        dfr16RFFTInit (pRFFT, 64, options);
        cont_m = 0;
        Config_Pll(14);
        Config_SPI();
        Config_GPIO();
        Inic_LCD();
        Config_ADC();
PORTADA:
        Portada();

MENU:
        for(;;)
        {
                if(!Izda()){while(!Izda()){} goto CONTI;}

                if(!Medio()) {while(!Medio()){} goto CONTI;}

                if(!Dere()){while(!Dere()){} goto CONTI;}
        }

CONTI:
        LimpiaGrafico();
        Portada();
        LimpiaTexto();
RECTANGULAR:
        ImpVectorTex(VEN,0x1C5,7,DERECHA);
        ImpVectorTex(REC,0x1E3,11,DERECHA);
        ImpVectorTex(PROCES,0x1F2,11,DERECHA);
        ventana = 0;
        for(;;)
        {
                if(!Izda()){while(!Izda()){} goto HAMMING;}
                if(!Dere()){while(!Dere()){} goto PROCESAR;}
        }

HAMMING:
        ImpVectorTex(HAM,0x1E3,11,DERECHA);
        ventana = 1;
        for(;;)
        {
```

```
            if(!Izda()){while(!Izda()){} goto HANNING;}
            if(!Dere()){while(!Dere()){} goto PROCESAR;}
        }

HANNING:
        ventana = 2;
        ImpVectorTex(HAN,0x1E3,11,DERECHA);
        for(;;)
        {
            if(!Izda()){while(!Izda()){} goto BLACKMAN;}
            if(!Dere()){while(!Dere()){} goto PROCESAR;}
        }

BLACKMAN:
        ventana = 3;
        ImpVectorTex(BLA,0x1E3,11,DERECHA);
        for(;;)
        {
            if(!Izda()){while(!Izda()){} goto RECTANGULAR;}
            if(!Dere()){while(!Dere()){} goto PROCESAR;}
        }

PROCESAR:
        Timer1 = open(BSP_DEVICE_NAME_QUAD_TIMER_C_0, 0,
&quadParam1 );
        cont_m = 0;
        do
        {
            asm(nop);
        }
        while(cont_m < 659);
        close(Timer1);                  //DETIENE EL TEMPORIZADOR
        filtrado();
        Enventanado(ventana);
        //      TRANSFORMADA DE FOURIER
        res = dfr16RFFT (pRFFT, (Frac16 *) &AdcBuffer[0], (dfr16_sInplaceCRFFT
*) &FFTInplaceBuf[0]);

RESULTADOS1:
mostrar = 0;
//LimpiaTexto();
//LimpiaGrafico();
//ImpVectorTex(QUINCE,0x1C2,5,DERECHA);
//ImpVectorTex(BARRAS,0x1CD,6,DERECHA);
//ImpVectorTex(SALIR,0x1D9,5,DERECHA);
```

```
Resultados(ventana,mostrar);
for(;;)
	{
		if(!Izda()){while(!Izda()){} goto RESULTADOS15;}

		if(!Medio()) {while(!Medio()){} goto RESULTADOSBARRAS;}

		if(!Dere()){while(!Dere()){} goto CONTI;}
	}

RESULTADOS15:
mostrar = 1;
//LimpiaTexto();
//LimpiaGrafico();
//ImpVectorTex(UNO,0x1C2,4,DERECHA);
//ImpVectorTex(BARRAS,0x1CD,6,DERECHA);
//ImpVectorTex(SALIR,0x1D9,5,DERECHA);
Resultados(ventana,mostrar);
for(;;)
	{
		if(!Izda()){while(!Izda()){} goto RESULTADOS1;}

		if(!Medio()) {while(!Medio()){} goto RESULTADOSBARRAS;}

		if(!Dere()){while(!Dere()){} goto CONTI;}
	}

RESULTADOSBARRAS:
//LimpiaTexto();
//LimpiaGrafico();
//ImpVectorTex(UNO,0x1E2,4,DERECHA);
//ImpVectorTex(QUINCE,0x1ED,5,DERECHA);
//ImpVectorTex(SALIR,0x1F9,5,DERECHA);
mostrar = 2;
Resultados(ventana,mostrar);
for(;;)
	{
		if(!Izda()){while(!Izda()){} goto RESULTADOS1;}

		if(!Medio()) {while(!Medio()){} goto RESULTADOS15;}

		if(!Dere()){while(!Dere()){} goto CONTI;}
	}

}
```

```
/*************************************************************
/***************CONFIGURACION DEL GPIO*********************/

void Config_GPIO()
{
        /*Configuracion puerto GPIOA's como PULSADORES*/
        periphBitSet(0x0007, &ArchIO.PortA.PullUpReg);
        periphMemWrite(0x0000, &ArchIO.PortA.DataDirectionReg);
   periphMemWrite(0x0000, &ArchIO.PortA.PeripheralReg);
   /*GPIOB6--->WR GPIOB7--->A0*/
   periphBitSet(0x00c0, &ArchIO.PortB.DataDirectionReg);
   periphBitClear(0x00c0, &ArchIO.PortB.PeripheralReg);
   /*SPI pines dedicados*/
   periphBitSet(0x0030,&ArchIO.PortB.PeripheralReg);
   periphBitSet(0x0030,&ArchIO.PortB.DataDirectionReg);
}

void Config_Pll(UWord16 t)
{
        periphBitClear(0x0004,&ArchIO.Pll.ControlReg);
        while(periphBitTest(0x0004,&ArchIO.Pll.StatusReg)){}
        periphBitSet(0x0010,&ArchIO.Pll.ControlReg);
        periphMemWrite(t,&ArchIO.Pll.DivideReg);
        periphBitSet(0x0080,&ArchIO.Pll.ControlReg);
        periphBitClear(0x0010,&ArchIO.Pll.ControlReg);
        while(!periphBitTest(0x0020,&ArchIO.Pll.StatusReg)){}
        periphBitClear(0x0001,&ArchIO.Pll.ControlReg);
        periphBitSet(0x0002,&ArchIO.Pll.ControlReg);
}

void Config_SPI()
{
        periphMemWrite(0x00D8,&ArchIO.Spi.ControlReg);
        periphMemWrite(0x000F,&ArchIO.Spi.DataSizeReg);
        periphBitSet(0x0002,&ArchIO.Spi.ControlReg);

}

void Config_ADC()
{

                                /* CONFIGURE THE ADC TO SAMPLE
CONTINUOUSLY */
   periphMemWrite(0x5002, &ArchIO.AdcA.Control1Reg);     /* SNYC MODE,
ANA0 IS SINGLE INPUT */
```

```
   periphMemWrite(0x0000, &ArchIO.AdcA.Control2Reg);     /* DIVIDER IS 0 */
   periphMemWrite(0x1111, &ArchIO.AdcA.ChannelList1Reg); /* SAMPLE 0 IS
ANA1 */
   periphMemWrite(0x1111, &ArchIO.AdcA.ChannelList2Reg); /* ALL SAMPLES
ARE AN0 */
   periphMemWrite(0x00FE, &ArchIO.AdcA.DisableReg);      /* ENABLE
SAMPLE 0 */
   periphMemWrite(0x0000, &ArchIO.AdcA.OffsetReg);       /* OFFSET IS 1.5 ((1.5
/3.3)*32768) *  */
   periphBitClear(0x4000, &ArchIO.AdcA.Control1Reg);     /* CLEAR STOP */
   periphBitSet(0x2000, &ArchIO.AdcA.Control1Reg);       /* START */
}

/*****************************************************************************
*******/
void  CallbackOnCompare1(qt_eCallbackType CallbackType, void* pParam)
{

        AdcBuffer[cont_m] = periphMemRead(&ArchIO.AdcA.ResultReg[0])-
0x35a8;
        cont_m++;

}

/*****************************************************************************
******/
/*****************    FUNCIONES DE PROCESAMIENTO
************************/
void filtrado(void)
{
        Frac32 prod32,acc32;
        UWord16  i,k;

        for(k=0;k<64;k++)
        {
                acc32=0;
                for (i=0;i<403;i++)
                        {
                                prod32 = L_mult_ls(coef[402-i],AdcBuffer[i+(4*k)]);
                                acc32 = L_add(acc32,prod32);
                        }
                AdcBuffer[k] = round(acc32);
        }
}
```

```
void Enventanado(UWord16 ventana)
{
	UWord16 k;
	UInt16 c=2;
	switch(ventana)
	{
		case 0:
					/*Rectangular*/
		break;
		case 1:		/*Hamming*/
			for(k=0;k<=63;k++)
			{
				AdcBuffer[k] = mult_r(wham[k],AdcBuffer[k]);
			}
		break;
		case 2:		/*Hanning*/
			for(k=0;k<=63;k++)
			{
				AdcBuffer[k] = mult_r(whan[k],AdcBuffer[k]);
			}
		break;
		case 3:		/*Blackman*/
			for(k=0;k<=63;k++)
			{
				AdcBuffer[k] = mult_r(wbla[k],AdcBuffer[k]);
			}
		break;
	}
}

void Resultados(UWord16 ventana, UWord16 mostrar)
{
	float Mod,Fase,Fundamental;
	UWord16 i,inic,fin,linea=0;
	switch(mostrar)
	{
		case 0:  //Ver 15 Primeros
			inic = 1;
			fin = 13;
			LimpiaTexto();
			LimpiaGrafico();
			ImpVectorTex(QUINCE,0x1C2,5,DERECHA);
			ImpVectorTex(BARRAS,0x1CD,6,DERECHA);
			ImpVectorTex(SALIR,0x1D9,5,DERECHA);
```

```
        ImpVectorTex(ARM,0x45,3,DERECHA);
        ImpVectorTex(IRMS,0x4E,4,DERECHA);
        ImpVectorTex(FASE,0x58,4,DERECHA);
        ImpVectorTex(uno,0x86,1,DERECHA);
        ImpVectorTex(tres,0xA6,1,DERECHA);
        ImpVectorTex(cinco,0xC6,1,DERECHA);
        ImpVectorTex(siete,0xE6,1,DERECHA);
        ImpVectorTex(nueve,0x106,1,DERECHA);
        ImpVectorTex(uno,0x125,1,DERECHA);
        ImpVectorTex(uno,0x126,1,DERECHA);
        ImpVectorTex(uno,0x145,1,DERECHA);
        ImpVectorTex(tres,0x146,1,DERECHA);
break;
case 1:  //Ver 15 Ultimos
        inic = 15;
        fin = 29;
        LimpiaTexto();
        LimpiaGrafico();
        ImpVectorTex(UNO,0x1C2,4,DERECHA);
        ImpVectorTex(BARRAS,0x1CD,6,DERECHA);
        ImpVectorTex(SALIR,0x1D9,5,DERECHA);
        ImpVectorTex(ARM,0x45,3,DERECHA);
        ImpVectorTex(IRMS,0x4E,4,DERECHA);
        ImpVectorTex(FASE,0x58,4,DERECHA);
        ImpVectorTex(uno,0x85,1,DERECHA);
        ImpVectorTex(cinco,0x86,1,DERECHA);
        ImpVectorTex(uno,0xA5,1,DERECHA);
        ImpVectorTex(siete,0xA6,1,DERECHA);
        ImpVectorTex(uno,0xC5,1,DERECHA);
        ImpVectorTex(nueve,0xC6,1,DERECHA);
        ImpVectorTex(dos,0xE5,1,DERECHA);
        ImpVectorTex(uno,0xE6,1,DERECHA);
        ImpVectorTex(dos,0x105,1,DERECHA);
        ImpVectorTex(tres,0x106,1,DERECHA);
        ImpVectorTex(dos,0x125,1,DERECHA);
        ImpVectorTex(cinco,0x126,1,DERECHA);
        ImpVectorTex(dos,0x145,1,DERECHA);
        ImpVectorTex(siete,0x146,1,DERECHA);
        ImpVectorTex(dos,0x165,1,DERECHA);
        ImpVectorTex(nueve,0x166,1,DERECHA);
break;
case 2:  //Porcentajes y Barras
        inic = 1;
        fin = 29;
        LimpiaTexto();
```

```
                LimpiaGrafico();
                ImpVectorTex(UNO,0x1E2,4,DERECHA);
                ImpVectorTex(QUINCE,0x1EB,5,DERECHA);
                ImpVectorTex(SALIR,0x1FB,5,DERECHA);
                ImpVectorTex(ARM,0x12,3,DERECHA);
                ImpVectorTex(PORCENTAJE,0x17,1,DERECHA);
                ImpVectorTex(uno,0x33,1,DERECHA);
                ImpVectorTex(tres,0x53,1,DERECHA);
                ImpVectorTex(cinco,0x73,1,DERECHA);
                ImpVectorTex(siete,0x93,1,DERECHA);
                ImpVectorTex(nueve,0xB3,1,DERECHA);
                ImpVectorTex(uno,0xD2,1,DERECHA);
                ImpVectorTex(uno,0xD3,1,DERECHA);
                ImpVectorTex(uno,0xF2,1,DERECHA);
                ImpVectorTex(tres,0xF3,1,DERECHA);
                ImpVectorTex(uno,0x112,1,DERECHA);
                ImpVectorTex(cinco,0x113,1,DERECHA);
                ImpVectorTex(uno,0x132,1,DERECHA);
                ImpVectorTex(siete,0x133,1,DERECHA);
                ImpVectorTex(uno,0x152,1,DERECHA);
                ImpVectorTex(nueve,0x153,1,DERECHA);
                ImpVectorTex(dos,0x172,1,DERECHA);
                ImpVectorTex(uno,0x173,1,DERECHA);
                ImpVectorTex(dos,0x192,1,DERECHA);
                ImpVectorTex(tres,0x193,1,DERECHA);
                ImpVectorTex(dos,0x1B2,1,DERECHA);
                ImpVectorTex(cinco,0x1B3,1,DERECHA);
                ImpVectorTex(dos,0x1D2,1,DERECHA);
                ImpVectorTex(siete,0x1D3,1,DERECHA);
                ImpVectorTex(dos,0x1F2,1,DERECHA);
                ImpVectorTex(nueve,0x1F3,1,DERECHA);

                ImpVectorTex(uno,0x00,1,DERECHA);
                ImpVectorTex(cero,0x01,1,DERECHA);
                ImpVectorTex(cero,0x02,1,DERECHA);
                ImpVectorTex(cinco,0xC1,1,DERECHA);
                ImpVectorTex(cero,0xC2,1,DERECHA);
                ImpVectorTex(cero,0x182,1,DERECHA);
        break;
}



for (i=inic;i<=fin;i=i+2)
{
```

```
AdcBuffer[500] = abs_s(FFTInplaceBuf[i].real);
AdcBuffer[501] = abs_s(FFTInplaceBuf[i].imag);
Mod = (float) AdcBuffer[500]*3.3/32768;
Fase = (float) AdcBuffer[501]*3.3/32768;
Mod = (Mod*Mod)+(Fase*Fase);
Mod =sqrt(Mod);
switch (ventana)
{
case 0:
break;
case 1:
        Mod = Mod*(64/34.56);
break;
case 2:
        Mod = Mod*2;
break;
case 3:
        Mod = Mod*(64/26.889);
break;
}
Mod = Mod * 10 * 1.41421356;

switch(mostrar)
{
        case 0:
                AdcBuffer[600] =
tfr16Atan2OverPI(FFTInplaceBuf[i].imag,FFTInplaceBuf[i].real);
                Fase = ((
(float)AdcBuffer[600]*3)/32768)*(180/3.1416);
                Convermod(Mod,linea);
                Converfase(Fase,linea);
        break;
        case 1:
                AdcBuffer[600] =
tfr16Atan2OverPI(FFTInplaceBuf[i].imag,FFTInplaceBuf[i].real);
                Fase = (( (float)
AdcBuffer[600]*3)/32768)*(180/3.1416);
                Convermod(Mod,linea);
                Converfase(Fase,linea);
        break;
        case 2:
                if(i==inic)
                {
                        Fundamental=Mod;
                }
```

```
                                if (Fundamental != 0)
                                {
                                        Mod = (Mod/Fundamental)*100;
                                }
                                else
                                {
                                        LimpiaTexto();
                                }
                                ConverDatos2(Mod,linea);
                        break;
                }
                linea++;
        }
}

void Convermod(float valor, UWord16 posilinea)
{
        float res;
        UWord16 k,direccion;
        bool mili;
        res = valor;
        if (valor<1)
                {
                direccion = 0x8D+(32*posilinea);
                res = res * 10;
                for (k=0;k<3;k++)
                        {
                        direccion = direccion+1;
                        res = res - comparar(res,direccion);
                        res = res * 10;
                        }
                ImpVectorTex(mAmpere,0x91+(32*posilinea),2,DERECHA);
                }
        else
                {
                direccion = 0x8C+(32*posilinea);
                res = res - comparar(res,direccion);
                res = res*10;
                ImpVectorTex(coma,0x8D+(32*posilinea),1,DERECHA);
                direccion++;
                for (k=0;k<3;k++)
                        {
                        direccion = direccion+1;
                        res = res - comparar(res,direccion);
                        res = res * 10;
```

```
                    }
                ImpVectorTex(Ampere,0x92+(32*posilinea),2,DERECHA);
                }
}

void Converfase(float Fase,UWord16 posilinea)
{
        float res;
        UWord16 k,direccion;
        res = Fase;
        direccion = 0x97+(32*posilinea);
        if(res<0)
        {
                ImpValor(0x2D,direccion,1,DERECHA);//imprime el menos
                res = res*(-1);
        }
        res = res/100;
        for (k=0;k<3;k++)
        {
                direccion = direccion+1;
                res = res - comparar(res,direccion);
                res = res * 10;
        }
        ImpValor(0xDF,direccion+1,1,DERECHA); //Imprime simbolo de grados
}


void ConverDatos2(float Mod,UWord16 posilinea)
{
        float res;
        UWord16 k,direccion,valor;
        res = Mod;
        direccion = 0x36+(32*posilinea);
        valor = res;
        res = res/100;
        for (k=0;k<3;k++)
        {
                res = res - comparar(res,direccion);
                res = res * 10;
                direccion = direccion+1;
        }
        if(valor != 0)
        {
        ImpValor(0xF0,PTRGRAFICO+0xD03+posilinea,valor,ARRIBA); // Linea
Izquierda de Grafico
```

```
        }
}

int comparar(float res, UWord16 direccion)
{
        float sustrac;
        if (res < 1) {ImpVectorTex(cero,direccion,1,DERECHA); return 0; }
                else
                {
                if (res < 2) {ImpVectorTex(uno,direccion,1,DERECHA);return 1; }

                        else
                        {
                        if (res < 3) {ImpVectorTex(dos,direccion,1,DERECHA); return
2; }
                                else
                                {
                                if (res < 4)
{ImpVectorTex(tres,direccion,1,DERECHA);return 3; }
                                        else
                                        {
                                        if (res < 5)
{ImpVectorTex(cuatro,direccion,1,DERECHA);return 4;}
                                                else
                                                {
                                                if (res < 6)
{ImpVectorTex(cinco,direccion,1,DERECHA); return 5; }
                                                        else
                                                        {
                                                        if (res < 7)
{ImpVectorTex(seis,direccion,1,DERECHA);return 6; }
                                                                else
                                                                {
                                                                if (res < 8
){ImpVectorTex(siete,direccion,1,DERECHA);return 7;}
                                                                        else
                                                                        {
                                                                        if (res < 9)
{ImpVectorTex(ocho,direccion,1,DERECHA);return 8;}
                                                                        else

        {ImpVectorTex(nueve,direccion,1,DERECHA);return 9;}
                                                                        }
                                                                }
                                                        }
```

```
                                        }
                                    }
                                }
                            }
                        }
                    }

/*******************************************************************

/*******************************************************************
*****/
//************************   FUNCIONES OPERACION DE LA PANTALLA
********
void EnviarDato(UWord16 dato)
{
        periphMemWrite(dato, &ArchIO.Spi.DataTxReg);  //escribe el dato en el gpio
        periphBitClear(0x0040,&ArchIO.PortB.DataReg);   //WR=0
        MicroSeg(65);
        periphBitSet(0x0040,&ArchIO.PortB.DataReg);     //WR=1
}

void EnviarComando(UWord16 comando)
{
        periphBitSet(0x0080,&ArchIO.PortB.DataReg);      //A0=1
        periphMemWrite(comando, &ArchIO.Spi.DataTxReg);  //escribe el dato en el
gpio
        periphBitClear(0x0040,&ArchIO.PortB.DataReg);    //WR=0
        MicroSeg(65);
        periphBitSet(0x0040,&ArchIO.PortB.DataReg);      //WR=1
        periphBitClear(0x0080,&ArchIO.PortB.DataReg);    //A0=0
}

void PosCursor(UWord16 pos)
{
        EnviarComando(0x46);        // Posiciona el cursor en la Direccion Indicada
por pos
        EnviarDato(pos);
        EnviarDato(pos>>8);
}

void MoverCursor(UWord16 Mov)
{
        EnviarComando(Mov);      // El parametro mov debe ser las conatantes
DERECHA,
        EnviarComando(0x42);     // IZQUIERDA,ARRIBA,ABAJO
```

```
}

void ImpValor(UWord16 val,UWord16 pos, UWord16 tamano, UWord16 direc)
{
        UWord16 i;                              // Imprime un caracter equivalente
   PosCursor(pos);                      // al valor asociaodo al argumento val

   MoverCursor(direc);
        for(i=0;i<=tamano-1;i++) EnviarDato(val);
}

void ImpVectorTex(const UWord16 texto[],UWord16 pos,UWord16 tamano,
UWord16 direc)
{
        UWord16 i;                              // Impreme un vector de texto
previamente almacenado
        PosCursor(pos); MoverCursor(direc);
        for(i=0;i<=tamano-1;i++) EnviarDato(texto[i]);
}

void LimpiaTexto(void) {ImpValor(0x20,0x00,512,DERECHA);}  //Limpia toda la
capa de texto

void LimpiaGrafico(void) {ImpValor(0x00,PTRGRAFICO,4096,DERECHA);}
//limpia toda la capa grafica

void Inic_LCD(void)
{
        periphBitSet(0x0040,&ArchIO.PortB.DataReg);     // WR=1
        EnviarComando(0x40);        // Inicio de comando
                EnviarDato(0x30);    // M0: CG ROM Interna
                                        // M1: CG RAM es 32 caracteres
maximo
                                    // M2: 8 lineas por caracter
                                        // W/S: = 0 Simple - Panel drive
                                        // 1
                                        // IV = 1; No Screen top_line
correction (no offset)
                                        // T/L = 0  LCD MODE
                                        // DR = 0 Normal Operation
                EnviarDato(0x87);    // FX: Tamaño Horizontal del caacter = 8 pixeles
                EnviarDato(0x07);    // FY : .111: ALTO DE CARACTER
                                            // FY: Tamaño Vertical del
Caracter = 8 pixels
                EnviarDato(0x4F);    // C/R AREA DE DIBUJO 64 por linea
```

```
            EnviarDato(0x52);     // TC/R: Total rango de direcciones por linea =
82
            EnviarDato(0x7F);     // L/F: 128 lineas de display
            EnviarDato(0x20);     // APL:Tamaño de Pantalla Virtual.
            EnviarDato(0x00);     // APH:   32 direcciones
        //COMANDOS DE DESPLAZAMIENTO
        EnviarComando(0x44);        // SCROLL: DESPLAZAMIENTO
        //PARAMETROS PRIMERA PANTALLA
            EnviarDato(0x00);     // Primer bloque inicia direccon 0x0000h
            EnviarDato(0x00);     // PAGINA DE TEXTO
            EnviarDato(0x7F);     // desplega 64 lineas el primer bloque CREO que
son 128
        //PARAMETROS SEGUNDA PANTALLA
            EnviarDato(0x00);     // Segundo inicia 0x1000h direccion
            EnviarDato(0x10);     // PAGINA GRAFICA
            EnviarDato(0x7F);     // 64 lineas el segundo bloque CREO que son
128
        //PARAMETROS TERCERA PANTALLA
            EnviarDato(0x00);
            EnviarDato(0x20);
        //PUNTO DE DESPLAZAMIENTO HORIZONTAL
        EnviarComando(0x5A);
            EnviarDato(0x00);
        //OVERLAY COMMAND
        EnviarComando(0x5B);        // MODO DE LA PANTALLA
TEXTO/GRAFICA
            EnviarDato(0x01);     // MX0 = 1 ^ MX1 = 0  ---> XOR

                                                // DM1 = 0      Capa 1 texto
                                                // DM2 = 0      Capa 3 texto --> no
USED
                                                // OV  = 0      2 Capas

        LimpiaTexto();              // LIMPIA PANTALLA DE TEXTO
        LimpiaGrafico();            // LIMPIA LA PANTALLA GRAFICA
        // FORMA DEL CURSOR
        EnviarComando(0x5D);
            EnviarDato(0x00);     // Tamaño del cursor en x ---> 1 pixel
            EnviarDato(0x81);     // Tamño del cursor en y ----> 1 pixel
                                                // El 8 es para modo gráfico
        // COMANDO ON/OFF-
        EnviarComando(0x59);        // DISPLAY ON
            EnviarDato(0x56);     // No flashing
}
```

```
void Portada()
{
	UWord16 i,j=0,x;
	UWord16 Direccion=0x100D;
	ImpVectorTex(PRO,0x0121,30,DERECHA);
	ImpVectorTex(COR,0x014A,12,DERECHA);
	ImpVectorTex(YIK,0x0184,25,DERECHA);
	ImpVectorTex(SHY,0x01A3,27,DERECHA);
	ImpVectorTex(CPS,0x01CB,9,DERECHA);
	ImpVectorTex(E3T,0x01EE,3,DERECHA);

	for (i=1;i<=63;i++)
	{
		PosCursor(Direccion); EnviarComando(0x42);
		for (x=1;x<=6;x++){EnviarDato(UIS_P[j]);j++;}
		Direccion=Direccion+0x20;
	}

}
/*********************************************************************/
// FUCIONES DE CONTROL PULSADORES

bool Izda(void) { return periphBitTest(0x0001,&ArchIO.PortA.DataReg);}

bool Medio(void) { return periphBitTest(0x0002,&ArchIO.PortA.DataReg);}

bool Dere(void) { return periphBitTest(0x0004,&ArchIO.PortA.DataReg);}

/*****************************************************************/
//FUNCIONES DE RETARDO

void MicroSeg(UWord32 t)
{
	UWord32 i;
	for (i=0;i<=t;i++)
	{
		asm(nop);asm(nop);asm(nop);
	}
}
///FUNCION DE RAIZA CUADRADA
float sqrt(float x)
{
 if(x<=0) return 0;
  else
   {
```

```
   if(x==1) return 1;
    else
     {
      float i=x/2,j=2*0.0001;
      while(j>= 0.0001)
       {
       i=(i+x/i)/2;
       if((i*i-x)*100/x>=0)
         {j=(i*i-x)*100/x;}
       else
         {j=(x-i*i)*100/x;}
       }
       return i;
     }
  }
}
```

B.2 ARCHIVO APPCONST.C

```
#include "port.h"
#include "appconst.h" /*CABECERA QUE USA EL SDK EN DONDE SE
DEFINEN VARIABLES QUE QUEDAN EN FLASH */



const UWord16 DERECHA   = 0x4C;
const UWord16 IZQUIERDA = 0x4D;
const UWord16 ARRIBA    = 0x4E;
const UWord16 ABAJO     = 0x4F;
const UWord16 PTRGRAFICO = 0x1000;
const UWord16 CON[]={"CNTRL"};
          //5
const UWord16 CPS[]={"Grupo CPS"};                                          //9
const UWord16 PRO[]={"PROTOTIPO MEDIDOR DE ARMONICOS"};       //30
const UWord16 COR[]={"DE CORRIENTE"};//12
const UWord16 SHY[]={"SHIRLEY  HERRERA  HERNANDEZ"};     //27
const UWord16 YIK[]={"JAIRO IVAN FLOREZ BARRERA"};          //25
const UWord16 E3T[]={"E3T"};                          // 3
const UWord16 REC[]={"RECTANGULAR"};     //11
const UWord16 HAN[]={"  HANNING  "};          // 7
const UWord16 HAM[]={"  HAMMING  "};        // 7
```

```
const UWord16 BLA[]={"  BLACKMAN "};        // 8
const UWord16 PROCES[]={"  PROCESAR "}; //11
const UWord16 VEN[] = {"VENTANA"};           // 7
const UWord16 UNO[] = {"1-13"};                  // 4
const UWord16 QUINCE[] = {"15-29"};    // 5
const UWord16 BARRAS[] = {"BARRAS"};       // 6
const UWord16 SALIR[] = {"SALIR"};              // 5
const UWord16 ARM[] = {"ARM"};                 // 5
const UWord16 IRMS[] = {"IRMS"};            // 6
const UWord16 FASE[] = {"FASE"};                 // 5
const UWord16 cero[] = {"0"};
const UWord16 uno[] = {"1"};
const UWord16 dos[] = {"2"};
const UWord16 tres[] = {"3"};
const UWord16 cuatro[] = {"4"};
const UWord16 cinco[] = {"5"};
const UWord16 seis[] = {"6"};
const UWord16 siete[] = {"7"};
const UWord16 ocho[] = {"8"};
const UWord16 nueve[] = {"9"};
const UWord16 PORCENTAJE[] = {"%"};
const UWord16 mAmpere[] = {"mA"};
const UWord16 Ampere[] = {"A"};
const UWord16 coma[] = {","};



const UWord16 UIS_P[]=
      { 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x80,0x00,0x00,0x00,0x00,0x01
       ,0x80,0x01,0xCF,0xF3,0x00,0x01,0x80,0x07,0xCF,0xF3,0xE0,0x01
       ,0x80,0x1F,0xCF,0xF3,0xF8,0x01,0x80,0x1F,0xCF,0xF3,0xFE,0x01
       ,0x81,0x9F,0xCF,0xF3,0xFF,0x81,0x83,0x9F,0xCF,0xF3,0xFF,0xC1
       ,0x87,0x9F,0xCF,0xF3,0xFF,0xE1,0x8F,0x9F,0xCF,0xF3,0xFF,0xF1
       ,0x9F,0x9F,0xCF,0xF3,0xFF,0xF9,0xBF,0x9F,0xCF,0xF3,0xFF,0xFD
       ,0xBF,0x9F,0xCF,0xF3,0xFF,0xFD,0xBF,0x9F,0xCF,0xF3,0xFF,0xFD
       ,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD
       ,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD
       ,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD
       ,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD
       ,0xBF,0x9F,0xCF,0xF3,0xF8,0x01,0xBF,0x9F,0xCF,0xF3,0xF8,0x01
       ,0xBF,0x9F,0xCF,0xF3,0xFF,0xFD,0xBF,0x9F,0xCF,0xF3,0xFF,0xFD
       ,0xBF,0x9F,0xCF,0xF3,0xFF,0xFD,0xBF,0x9F,0xCF,0xF3,0xFF,0xFD
       ,0xBF,0x9F,0xCF,0xF3,0xFF,0xFD,0xBF,0x9F,0xCF,0xF3,0xFF,0xFD
       ,0xBF,0x9F,0xCF,0xF3,0xFF,0xFD,0xBF,0x9F,0xCF,0xF3,0xFF,0xFD
```

```
,0xBF,0x9F,0xCF,0xF0,0x01,0xFD,0xBF,0x9F,0xCF,0xF0,0x01,0xFD
,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD
,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD
,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD
,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD
,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD
,0xBF,0x9F,0xCF,0xF3,0xF9,0xFD,0xBF,0x9F,0xCF,0xF3,0xFF,0xFD
,0xBF,0xFF,0xCF,0xF3,0xFF,0xFD,0xBF,0xFF,0xCF,0xF3,0xFF,0xFD
,0xBF,0xFF,0xCF,0xF3,0xFF,0xFD,0xBF,0xFF,0xCF,0xF3,0xFF,0xFD
,0x9F,0xFF,0xCF,0xF3,0xFF,0xF9,0x8F,0xFF,0xCF,0xF3,0xFF,0xF1
,0x87,0xFF,0xCF,0xF3,0xFF,0xE1,0x83,0xFF,0xCF,0xF3,0xFF,0xC1
,0x81,0xFF,0xCF,0xF3,0xFF,0x81,0x80,0xFF,0xCF,0xF3,0xFF,0x01
,0x80,0x3F,0xCF,0xF3,0xFC,0x01,0x80,0x1F,0xCF,0xF3,0xF8,0x01
,0x80,0x07,0xCF,0xF3,0xE0,0x01,0x80,0x00,0xCF,0xF3,0x00,0x01
,0x80,0x00,0x07,0xE0,0x00,0x01,0x80,0x00,0x00,0x00,0x00,0x01
,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};


/*  403 COEFICIENTES DEL FILTRO FIR  */
const Frac32 coef[] =
{         993,0,-1545,-2647,-2237,0,3090,5069,4129,0,-5379,-8616,-
6870,0,8631,13609,
          10696,0,-13103,-20429,-15889,0,19096,29518,22771,0,-26961,-41388,-
31718,0,37098,56623,
          43158,0,-49961,-75889,-57573,0,66064,99932,75509,0,-85984,-129590,-
97572,0,110360,165791,
          124438,0,-139902,-209564,-156850,0,175388,262036,195625,0,-217675,-
324444,-241656,0,267693,398134,
          295916,0,-326457,-484568,-359459,0,395065,585332,433428,0,-474705,-
702139,-519060,0,566662,836841,
          617689,0,-672323,-991439,-730757,0,793189,1168095,859825,0,-930883,-
1369157,-1006587,0,1087173,1597175,
          1172887,0,-1263988,-1854942,-1360746,0,1463447,2145530,1572391,0,-
1687897,-2472349,-1810301,0,1939962,2839220,
          2077261,0,-2222606,-3250474,-2376436,0,2539221,3711079,2711470,0,-
2893737,-4226819,-3086621,0,3290781,4804523,
          3506936,0,-3735881,-5452389,-3978496,0,4235757,6180418,4508754,0,-
4798712,-7001030,-5107009,0,5435205,7929934,
          5785075,0,-6158651,-8987391,-6558269,0,6986628,10200083,7446871,0,-
7942675,-11603959,-8478376,0,9059120,13248721,
          9691062,0,-10381628,-15205125,-
11139862,0,11976886,17577455,12906530,0,-13946217,-20525959,-
15118191,0,16451316,24309999,
```

```
        17983718,0,-19766817,-29378265,-
21871662,0,24399261,36579266,27498208,0,-31396407,-47732643,-
36462088,0,43331374,67571600,
        53205943,0,-68660973,-113455024,-
96392133,0,160950585,341624869,483297853,536870912,483297853,341624869,1
60950585,0,-96392133,-113455024,
        -68660973,0,53205943,67571600,43331374,0,-36462088,-47732643,-
31396407,0,27498208,36579266,24399261,0,-21871662,-29378265,
        -19766817,0,17983718,24309999,16451316,0,-15118191,-20525959,-
13946217,0,12906530,17577455,11976886,0,-11139862,-15205125,
        -10381628,0,9691062,13248721,9059120,0,-8478376,-11603959,-
7942675,0,7446871,10200083,6986628,0,-6558269,-8987391,
        -6158651,0,5785075,7929934,5435205,0,-5107009,-7001030,-
4798712,0,4508754,6180418,4235757,0,-3978496,-5452389,
        -3735881,0,3506936,4804523,3290781,0,-3086621,-4226819,-
2893737,0,2711470,3711079,2539221,0,-2376436,-3250474,
        -2222606,0,2077261,2839220,1939962,0,-1810301,-2472349,-
1687897,0,1572391,2145530,1463447,0,-1360746,-1854942,
        -1263988,0,1172887,1597175,1087173,0,-1006587,-1369157,-
930883,0,859825,1168095,793189,0,-730757,-991439,
        -672323,0,617689,836841,566662,0,-519060,-702139,-
474705,0,433428,585332,395065,0,-359459,-484568,
        -326457,0,295916,398134,267693,0,-241656,-324444,-
217675,0,195625,262036,175388,0,-156850,-209564,
        -139902,0,124438,165791,110360,0,-97572,-129590,-
85984,0,75509,99932,66064,0,-57573,-75889,
        -49961,0,43158,56623,37098,0,-31718,-41388,-
26961,0,22771,29518,19096,0,-15889,-20429,
        -13103,0,10696,13609,8631,0,-6870,-8616,-5379,0,4129,5069,3090,0,-2237,-
2647,
        -1545,0,993,};

const Frac16 wham[] =
{ 2621,  2694,  2911,  3270,  3769,  4401,  5162,  6043,
  7036,  8132,  9320, 10589, 11926, 13319, 14754, 16217,
 17695, 19172, 20635, 22070, 23463, 24800, 26069, 27257,
 28353, 29347, 30228, 30988, 31621, 32119, 32478, 32695,
 32767, 32695, 32478, 32119, 31621, 30988, 30228, 29347,
 28353, 27257, 26069, 24800, 23463, 22070, 20635, 19172,
 17695, 16217, 14754, 13319, 11926, 10589,  9320,  8132,
  7036,  6043,  5162,  4401,  3769,  3270,  2911,  2694,

        };

const Frac16 whan[] =
```

```
{          0,    79,   315,   705,  1247,  1935,  2761,  3719,
  4799,  5990,  7282,  8661, 10114, 11628, 13188, 14778,
 16384, 17990, 19580, 21140, 22654, 24107, 25486, 26778,
 27969, 29049, 30007, 30833, 31521, 32063, 32453, 32689,
 32767, 32689, 32453, 32063, 31521, 30833, 30007, 29049,
 27969, 26778, 25486, 24107, 22654, 21140, 19580, 17990,
 16384, 14778, 13188, 11628, 10114,  8661,  7282,  5990,
  4799,  3719,  2761,  1935,  1247,   705,   315,    79,


 };

const Frac16 wbla[] =
{    0,    29,   115,   264,   479,   770,  1143,  1609,
  2177,  2857,  3657,  4583,  5639,  6827,  8144,  9586,
 11141, 12797, 14537, 16339, 18179, 20030, 21862, 23645,
 25348, 26939, 28389, 29668, 30753, 31621, 32254, 32639,
 32767, 32639, 32254, 31621, 30753, 29668, 28389, 26939,
 25348, 23645, 21862, 20030, 18179, 16339, 14537, 12797,
 11141,  9586,  8144,  6827,  5639,  4583,  3657,  2857,
  2177,  1609,  1143,   770,   479,   264,   115,    29,
};
```

ANEXO C

HOJA D DATOS, CIRCUITOS INTEGRADOS

A continuación se presenta un listado de las hojas de datos, de los circuitos integrados y del *hardware* relacionado con este trabajo.

1. Pinza sonda MN 375 www.aemc.com
2. LTC1563-2 (Filtro Butterworth de 4to orden) www.linear-tech.com
3. AD822 (Circuito operacional, sumador no inversor, *offset*) www.analogdevices.com
4. Condensadores SOIC www.nichicon.com
5. Resistencias SOIC www.phicomp.com
6. LD1117 (Regulador de 3,3 V, alimentación tarjeta de adquisición) www.st.com
7. PT5061 (Regulador de +/- 12 V, alimentado con 5 V) www.ti.com
8. TLV431 (Regulador ajustable a 1,5 V para el *offset*) www.ti.com
9. DSP56F801F60 www.frescale.com
10. Demoboard (dsp56f800dbum) www.frescale.com
11. 74LS164 (Registro de desplazamiento) www.national.com
12. SED1330F (CI Pantalla LCD gráfica) www.smos-systems.com

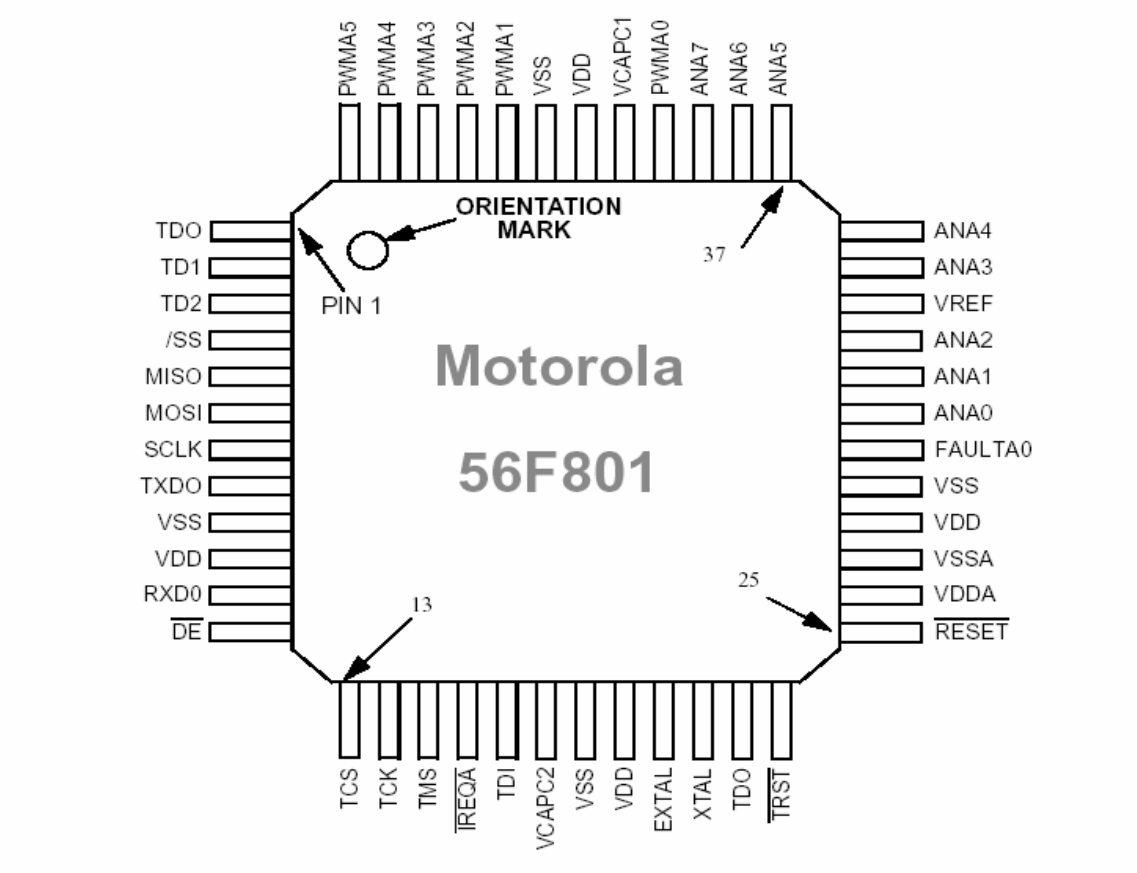
ANEXO D

COMO INDEPENDIZAR EL DSP DE LA DEMOBOARD?

Dentro de las recomendaciones de éste libro se advirtió que uno de los procesos a realizar para la mejora del prototipo es el de independizar el *DSP* de la *Demoboard* 56F800 de 60MHz; para ello hay es necesario saber que la referencia del *DSP* es DSP56F801FA60.

El DSP56F801FA60 tiene 48 pines que se distribuyen tal como se ve observa en la Figura D.1.

Figura D.1 DSP56F801FA60



A partir del *DSP*, el procedimiento a seguir para la correcta utilización del mismo es:

1) Diseñar y conectar adecuadamente la fuente de alimentación.
2) Desarrollar la interfaz de comunicación entre el PC y el puerto *Joint Test Action Group* (*JTAG)* del *DSP*

D.1 Fuente de alimentación:

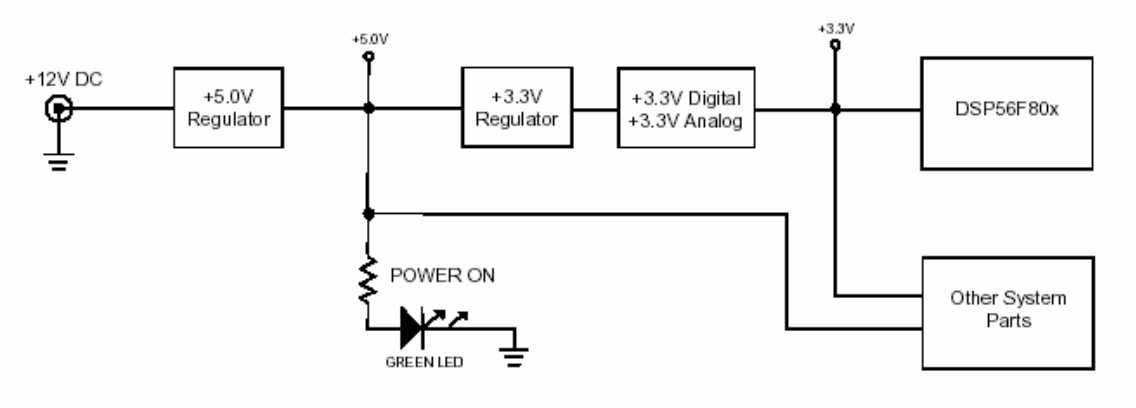
La especificación de tensión de alimentación D.C. (*suply voltage*) para la familia DSP56F801 varía de 3.0V (mínimo) a 3.6V(máximo). Ésta fuente de tensión debe ser aplicada a todas los pines V_{DD}. Internamente, la tensión es regulada a 2.5V para la circuiteria digital del núcleo (*core*). La fuente análoga y el tensión de referencia V_{DDA} y V_{REF} respectivamente deben cumplir la siguiente restricción:

$$V_{REF} =\leq V_{DDA} =\leq V_{DD}$$

El consumo de corriente del *DSP* depende del estado de operación en que se encuentre. El mayor consumo se presenta cuando el *DSP* esta en modo *RUN* y equivale a 102 mA. Para conocer más acerca del consumo de corriente en los diferentes estados de operación del *DSP* dirigirse a la hoja de datos del *DSP* que se encuentra en el anexo C.

Un ejemplo de fuente de Tensión seria el que se muestra en la Figura D.2

Figura D.2 Fuente de Alimentación

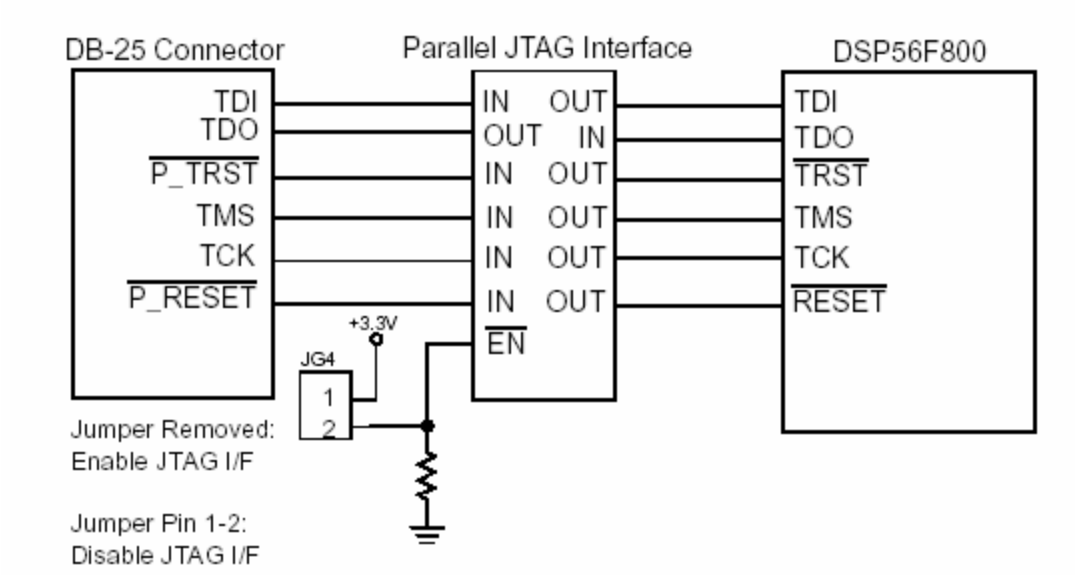


Para mayor información acerca de cómo implemetar ésta fuente de Tensión dirigirse al siguiente referencia [DSP56800 Hardware Interface Techniques].

D.2 Interfaz de Comunicación entre el PC y el JTAG del *DSP*.

La Figura D.3 muestra como es la interfaz entre el puerto paralelo del PC y el *JTAG* del *DSP.*

Figura D.3 Diagrama de bloques de la la interfaz Puerto paralelo *JTAG*



Este tipo de interfaz esta detallado en el estándar IEEE 1149.1a *Standard Test Access Port and Boundary Scan Architecture*.