Implementación de un Algoritmo de Filtrado en Microcontroladores dsPICy PIC32

Vazquez Raimundo[1], Mariguetti Jorge[a], Burgos Alejandro[a], Canali Luis[b].

[a] GUDA-Grupo Universitario de Automatización
Universidad Tecnológica Nacional - Facultad Regional Resistencia
French 414. 3500 Resistencia-Chaco. República Argentina
Tel: +54 362 4432928. / Fax: +54 362 4432683/
e-mail: ray_vazquez_2005@hotmail.com/ray_vazquez_2005@yahoo.com.ar

[b] UTN Facultad Regional Córdoba
Maestro M. Lopez esq, Cruz Roja Argentina, Ciudad Universitaria, 5016
Córdoba, Argentina.
e-mail: luis.canali@gmail.com

Resumen: Se implementa un algoritmo para la Trasformada Rápida de Fourier (FFT) en microcontroladores dsPIC y PIC32 mediante la reutilización software, lo cual permite disminuir el esfuerzo y el tiempo en programación. Los conocimientos necesarios para su implementación son mínimos por parte del programador, utilizándose librerías de funciones matemáticas y números complejos. Se busca generalizar el uso de los algoritmos de filtrado con los dispositivos genéricos utilizados. Se observa que la implementación de Filtros Lineales de Respuesta Finita (FIR) requiere menos cálculo computacional que la FFT. También se observa que existe una limitación en la longitud del vector donde se alojan los valores digitales provenientes de la operación de conversión analógica a digital. Para el caso de los PIC 32 esta longitud es de 512 y para los dsPIC es de 128. Es posible aumentar este número cambiando las condiciones de compilación y linkeo del simulador MPLAB. Los resultados obtenidos del hardware se pueden extender a otros desarrollos genéricos que utilicen dsPIC y PIC32.

Palabra clave: Transformada Rápida de Fourier, Filtros Lineales, PIC32 y dsPIC

1. Introducción

La teoría del control automático y la tecnología de los robots, posibilita combinar diversas disciplinas como por ejemplo mecánica, informática y electrónica con la finalidad de diseñar y construir aplicaciones que facilitan la experimentación. La integración de la arquitectura de control con las funciones de un sistema de medida permite evaluar tareas de fusión sensorial [1]. En el siguiente trabajo se implementan procedimientos que permiten mejorar la toma de datos analógicos mediante la implementación de Transformada Rápida de Fourier [2] y filtros respuesta finita [3]. Los mismos se esquematizan en un algoritmo genérico fácilmente codificable en un lenguaje de programación de alto nivel para implementarlos en dispositivos genéricos

denominados microcontroladores [4] del tipo dsPIC y PIC32. Se parte de la premisa que afirma que es posible implementar en forma sencilla un algoritmo de filtrado FFT y FIR utilizando un lenguaje de alto nivel como por ejemplo el C. La mayoría del hardware implementa el cálculo de FFT en un lenguaje de bajo nivel, requiriendo elevados conocimientos de teoría de números complejos y dominio de la programación [5][6][7].

2. Algoritmos utilizados

2.1. Transformada Rápida de Fourier

La transformada rápida de Fourier (FFT) es un algoritmo eficiente que permite calcular la Transformada de Fourier Discreta (DFT) y su Transformada Inversa. La FFT es de gran importancia para una amplia variedad de aplicaciones, especialmente para el procesamiento digital de señales y el filtrado digital como lo desarrolla Di Jasio [8][9]. El algoritmo impone algunas limitaciones en la señal y en el espectro resultante, como por ejemplo las muestras que se transforman deben consistir en un número igual a una potencia de dos. Hay que tener en cuenta que el rango de frecuencias cubierto por el análisis de la señal depende de la cantidad de muestras recogidas.

2.2. Filtros lineales de Respuesta Finita (FIR)

Un filtro lineal es aquel filtro electrónico que aplica un operador lineal a una señal variable en el tiempo. Una de sus aplicaciones más frecuentes es la eliminación de frecuencias no deseadas de una determinada señal de entrada o, al contrario, discriminar una determinada frecuencia de las demás.
Los filtros lineales pueden dividirse en dos clases: filtros de respuesta infinita (IIR, infinite impulse response) y filtros de respuesta finita (FIR, finite impulse response):

- Los filtros FIR (que sólo puede ser implementados en tiempo discreto) pueden ser descritos como una suma ponderada de entradas con un determinado retardo. Para estos filtros, si la entrada en un determinado instante es cero, la salida será cero a partir de un instante posterior a los retardos inducidos por el filtro. De este modo, solo existirá respuesta por un tiempo finito.

- Los filtros IIR, por el contrario, pueden presentar salida aún cuando la entrada sea cero, si las condiciones iniciales son distintas de cero. La energía del filtro decaerá con el tiempo, pero no llegará a ser nula. Por tanto, la respuesta al impulso se extiende infinitamente.

Hay una clase muy importante de filtros FIR que son los que poseen fase lineal, esto es, la respuesta en fase del sistema es una recta en la banda pasante. Este caso de

filtros FIR se da cuando los coeficientes son simétricos o antisimétricos, a su vez, el orden del filtro puede ser par o impar. De esta manera los filtros FIR se pueden clasificar en cuatro tipos según Naguil [10], simétricos par, simétricos impar, antisimétricos par y antisimétricos impar.

3. Algoritmo FFT

El algoritmo FFT desarrollado en esta sección está basado en el método conocido como doblado sucesivo descripto por Gonzales [11]. Se parte de la ecuación (1) de la forma:

$$F_u = \frac{1}{2 \cdot M} \cdot \left[\sum_{x=0}^{N-1} \left(a_x \cdot W(u,x) \right) \right] \tag{1}$$

Donde W(u,x) es

$$W(u,x) = e^{\left(-j \cdot 2 \cdot \frac{u \cdot \pi \cdot x}{2 \cdot M} \right)} \tag{2}$$

El resultado de la transformada se representa en un vector denominado F de subíndice u. El valor N es la cantidad de elemento de la muestra, M es la mitad de N. Los valores de la muestra se cargan en el vector denominado a_x y x representa el subíndice. La fórmula (3) y (4) se obtiene a partir de (1).

$$Fpar_u = \frac{1}{M} \cdot \left[\sum_{x=0}^{M-1} \left(a_{2 \cdot x} \cdot W1(u,x) \right) \right] \tag{3}$$

$$Fimpar_u = \frac{1}{M} \cdot \left[\sum_{x=0}^{M-1} \left(a_{2 \cdot x+1} \cdot W1(u,x) \right) \right] \tag{4}$$

Todos los subíndices pares de x perteneciente a_x se agrupan en el vector Fpar y los impares en el vector Fimpar.

Donde W1(u,x) es:

$$W1(u,x) = e^{\left(-j \cdot 2 \cdot \frac{\pi \cdot u \cdot x}{\frac{N}{2}} \right)} \tag{5}$$

Para implementar el FFT en el microcontrolador es necesario disponer de dos vectores denominados p1 y p2. El algoritmo de la figura 1 separa los subíndices pares e impares de a_x en dos columnas de la matriz p. La primera columna tiene posiciones pares que se agrupan en la primera mitad de p1 y los impares en la segunda mitad. Se repite la misma operación para la segunda columna en p2.

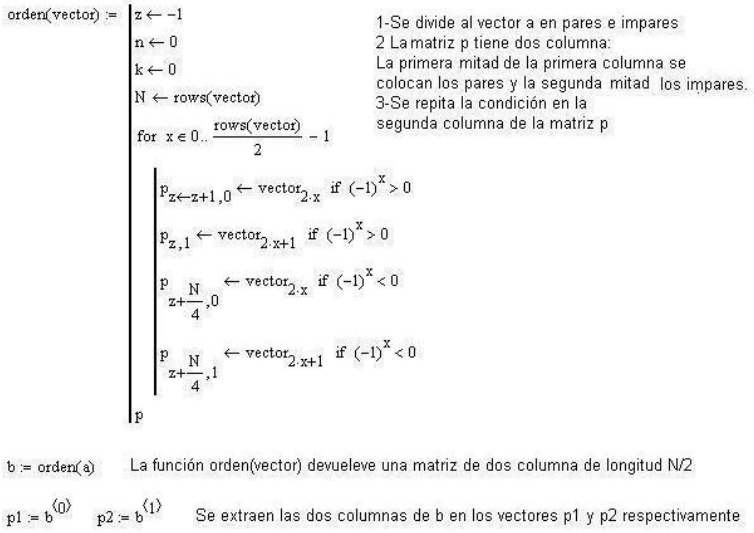

Fig. 1. Algoritmo para ordenar los elementos del vector a.

Luego de obtener los vectores p1 y p2 se aplican las ecuaciones (6) y (7).

$$Fp_u = \left[\sum_{x=0}^{\frac{N}{4}-1} \left[\left(p1_x \cdot W1(u,x) \right) + p1_{x+\frac{N}{4}} \cdot W1(u,x) \cdot W2(u) \right] \right] \tag{6}$$

$$Fi_u = \left[\sum_{x=0}^{\frac{N}{4}-1} \left[\left(p2_x \cdot W1(u,x) \right) + p2_{x+\frac{N}{4}} \cdot W1(u,x) \cdot W2(u) \right] \right] \tag{7}$$

Los vectores denominado Fp_u(par) y Fi_u(impar) contienen los valores de la transformada Rápida de Fourier. Su longitud es la cuarta parte del vector a_x, de esta manera se disminuye tiempo de procesamiento, ya que se recorre la cuarta parte del vector. Las funciones denominadas W1(u,x) y W2(u) quedan expresadas como:

$$W1(u,x) = e^{\left(\frac{-j \cdot 2 \cdot \pi \cdot u \cdot x}{\frac{N}{4}} \right)} \qquad W2(u) = e^{\left(-j \cdot 2 \cdot \pi \cdot \frac{u}{\frac{N}{2}} \right)} \tag{8}$$

El procedimiento desarrollado por González finaliza aplicando las ecuaciones (9) y (10).

$$S_u = \frac{1}{N} \cdot \left(Fp_u + Fi_u \cdot W3(u) \right) \tag{9}$$

$$S_{\frac{N}{2}+u} = \frac{1}{N} \cdot \left(Fp_u - Fi_u \cdot W3(u) \right) \tag{10}$$

Donde W3(u) es

$$W3(u) = e^{\left(-j \cdot \frac{2 \cdot \pi \cdot u}{N} \right)} \tag{11}$$

El vector S contiene los elementos de la transformada de Fourier utilizando la FFT. Su cálculo requiere variar el subíndice u de cero al valor N/2. La segunda mitad se obtiene aplicando la ecuación (10), donde Fp y Fi fueron previamente calculados en (9). De esta manera en una sola corrida se obtienen todos los valores de FFT. La figura 2 muestra una señal digital tipo TTL y su representación espectral.

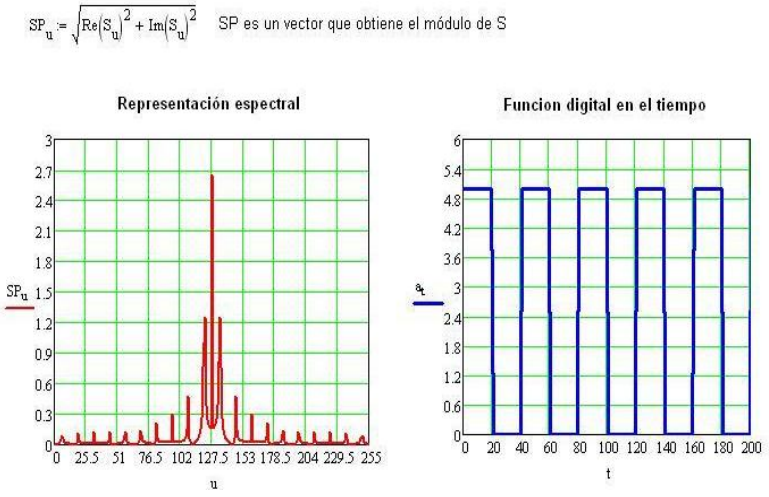

Fig. 2. Representación espectral y en tiempo de una señal digital tipo TTL.

El algoritmo FFT permite obtener los elementos del vector S. Luego se calcula el módulo de sus componentes complejos en otro vector denominado SP.

4. Filtro digital aplicando algoritmo FFT

En la figura 3 se muestra una señal digital tipo escalón TTL afectada por un ruido de alta frecuencia.

```
filtro(F,k1,k2,f1,f2) := | for x ∈ 0..rows(F) − 1
                         |   | T_x ← k1·F_x  if (x ≤ f1) ∨ (x ≥ f2)
                         |   | T_x ← k2·F_x  if (x > f1) ∧ (x < f2)
                         | T
```

La función filtro devuelve el vector F.
1-Necesita las constantes de atenuación k1 y k2
2-Las frecuencias de corte son f1 y f2
3-El vector se calcula a partir de FFT.
4-El código fuente en C para construir la función filtro es directa.
Un si condicional (if) y dos constantes int k1,k2.

$F := filtro(S, 0.1, 1, 500, 1500)$ Vector filtro de ls señal S

$Filtro_u := \sqrt{Re(F_u)^2 + Im(F_u)^2}$ Módulo de la señal filtrada F

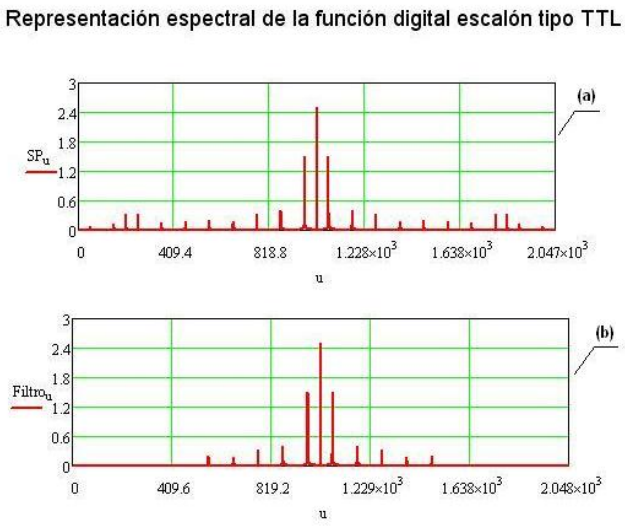

Fig. 3. Representación de la señal digital ruidosa (a) y la filtrada (b).

Al resultado de la señal filtrada se le debe aplicar la transformada rápida inversa de Fourier, que solo consiste en aplicar las ecuaciones (9) y (10) cambiando el signo de las exponentes de las ecuaciones (8) y (11). El algoritmo se muestra en la figura 4.

$$W1(u,x) := e^{\left(\frac{j\cdot 2\cdot\pi\cdot u\cdot x}{\frac{N}{4}}\right)} \quad W2(u) := e^{\left(j\cdot 2\cdot\pi\cdot\frac{u}{\frac{N}{2}}\right)} \quad W3(u) := e^{\left(j\cdot\frac{2\cdot\pi\cdot u}{N}\right)}$$

$$Fp_u := \left[\sum_{x=0}^{\frac{N}{4}-1}\left[\left(p1_x\cdot W1(u,x)\right) + p1_{x+\frac{N}{4}}\cdot W1(u,x)\cdot W2(u)\right]\right] \quad Fi_u := \left[\sum_{x=0}^{\frac{N}{4}-1}\left[\left(p2_x\cdot W1(u,x)\right) + p2_{x+\frac{N}{4}}\cdot W1(u,x)\cdot W2(u)\right]\right]$$

$S_u := \left(Fp_u + Fi_u\cdot W3(u)\right)$ Ecuación (9)

$S_{\frac{N}{2}+u} := \left(Fp_u - Fi_u\cdot W3(u)\right)$ Ecuación (10)

Traslación de la señal digital

$A_u := Re\left[(-1)^u\cdot S_u\right] \quad a_u := (-1)^u\cdot a_u$

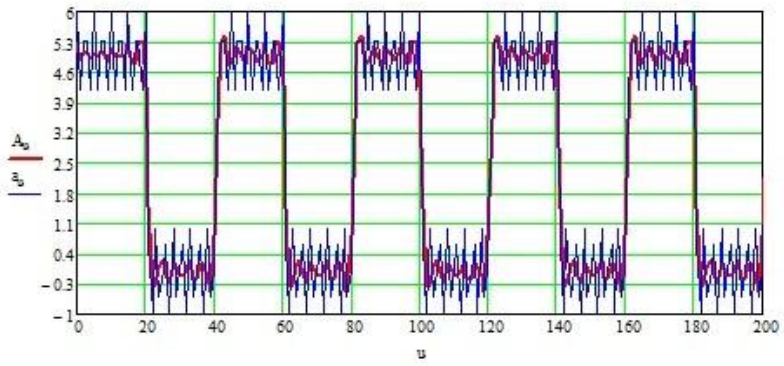

Fig. 4. Atenuación del ruido de alta frecuencia en la señal ruidosa digital.

5. Filtro FIR lineal

El procedimiento utilizado por Naguil [10] permite en forma fácil y sencilla atenuar el efecto de ruido en la señal digital. En la figura 5 se representa la frecuencia ideal según la tipología básica.

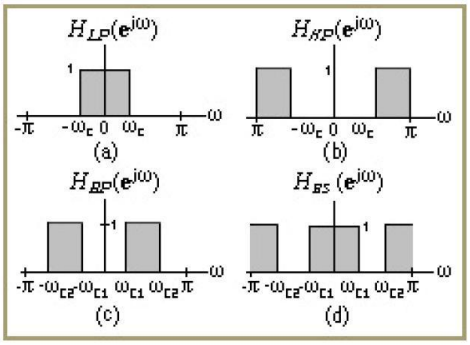

Fig. 5. Representación en frecuencia ideal de la tipología básica [5].

En este caso se elige el ejemplo (a) del gráfico. En la figura 6 se muestra el algoritmo que calcula los coeficientes b_k del FIR lineal para una frecuencia de corte.

$\omega_2 := 0.6 \cdot \pi \qquad M1 := 5$

coeficiente(n, ω, M1) :=
- $z \leftarrow 0$
- for $k \in -n..n$
  - $b_{z,0} \leftarrow \frac{\sin(\omega_2 \cdot k)}{k \cdot \pi}$ if $k \neq 0$ — Coeficientes del filtro pasa bajo Según el método de Naguil Jorge
  - $b_{z,0} \leftarrow \frac{\omega_2}{\pi}$ if $k = 0$
  - $w_z \leftarrow 0.54 - 0.46 \cdot \cos\left[\frac{2 \cdot \pi \cdot (k + n)}{M1}\right]$ <---Ventana Hamming
  - $b_{z,1} \leftarrow w_z$
  - $b_{z,2} \leftarrow w_z \cdot b_{z,0}$
  - $z \leftarrow z + 1$
- b

bc := coeficiente(2, ω, M1)

$$bc = \begin{pmatrix} -0.09355 & 0.08 & -0.00748 \\ 0.30273 & 0.39785 & 0.12044 \\ 0.6 & 0.91215 & 0.54729 \\ 0.30273 & 0.91215 & 0.27614 \\ -0.09355 & 0.39785 & -0.03722 \end{pmatrix}$$

Matriz numérica que contiene:
1-Los coeficientes bk
2-Los coeficientes corregidos según la ventana Hamming

Fig. 6. Algoritmo utilizado para obtener los coeficientes b_k del FIR lineal.

El resultado obtenido utilizando el FIR lineal y comparado con la figura 4 se muestra en la figura 7.

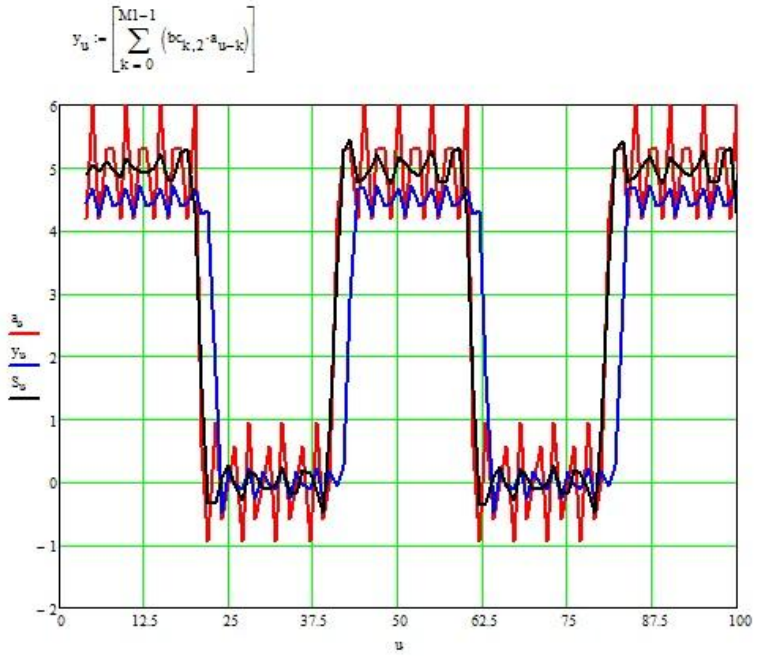

Fig. 7. Comparación entre el filtro FIR Lineal y la figura 4.

Se observan en la figura el comportamiento del ruido representado por el color rojo (vector a), el filtro FIR de color azul (vector y) y la transformada rápida inversa de Fourier por la línea negra (vector S)

6. Herramienta de desarrollo

Los dispositivos digitales en este trabajo fueron desarrollados por la empresa Microchip [12], como se visualiza en la figura 8. El compilador utilizado en los microcontroladores PIC gama 24FJXX y PIC32MXX son respectivamente C30 y C32

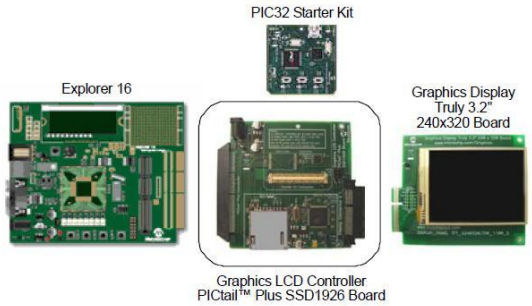

Fig. 8. Dispositivos genéricos utilizados en la experiencia de laboratorio [12].

Se destaca el proyecto denominado Graphics Primitive Layer Demo que contiene el algoritmo base y las librerías encargadas de manejar el controlador del LCD y los conversores analógicos digitales.

7. Implementación del algoritmo de filtrado

Las ecuaciones (6), (7), (9) y (10) junto con el algoritmo de ordenamiento mostrado en la figura 1, se implementan en un código fuente en lenguaje C utilizando el compilador C30 para dsPIC o C32 en los PIC32. En la figura 9 se visualiza el código fuente.

```
/*
The MPLAB v8.91 C30 and C32 compiler supports complex data types as an extension of both integer
and floating-point types. Here is an example declaration for a single precision
floating-point type: __complex__. Ver Lucio Di Jacio pagina 77.
*/
        for(u=0;u<N/2;u++)
        { for(x=0;x<N/4;x++) {
         __real__     W1=cos(-2*M_PI*u*x/(N/4));// __complex__ float W1;
         __imag__     W1=sin(-2*M_PI*u*x/(N/4));//__complex__ float W2;
         __real__     W2=cos(-2*M_PI*u/(N/2));//__complex__ float W3;
         __imag__     W2=sin(-2*M_PI*u/(N/2));//__complex__ float Z1; __complex__ float Z2;
                                              Ecuación (6)
        real_par[u]=real_par[u]+par[x]*__real__ W1+par[x+N/4]*__real__ (W1*W2);
        imag_par[u]=imag_par[u]+par[x]*__imag__ W1+par[x+N/4]*__imag__ (W1*W2);
                                              Ecuación (7)
        real_impar[u]=real_impar[u]+impar[x]*__real__ W1+impar[x+N/4]*__real__ (W1*W2);
        imag_impar[u]=imag_impar[u]+impar[x]*__imag__ W1+impar[x+N/4]*__imag__ (W1*W2);}

         __real__     W3=cos(-2*M_PI*u/N);
         __imag__     W3=sin(-2*M_PI*u/N);
         __real__     Z1=real_par[u];
         __imag__     Z1=imag_par[u];
         __real__     Z2=real_impar[u];
         __imag__     Z2=imag_impar[u];

         S_real[u]=(__real__ Z1+__real(Z2*W3))/N;Ecuación (9)
         S_imag[u]=(__imag__ Z1+__imag(Z2*W3))/N;
         S_real[u+N/2]=(__real__ Z1-__real(Z2*W3))/N;Ecuación (10)
         S_imag[u+N/2]=(__imag__ Z1-__imag(Z2*W3))/N;
         }
```

Fig. 9. El código fuente se compila en un simulador denominado MPLAB v8.91.

El programa C que contiene al algoritmo tiene una librería matemática y el comando denominado __complex__ que facilita cálculos con números complejos. El C30 o C32 no pueden manejar vectores con números complejos. Se debió utilizar otra matriz para contener la parte imaginaria. Por este motivo se debió definir ocho vectores en lugar de cuatro.

El procedimiento para compilar y correr el software fue extraído del trabajo de Usategui [13]. En la figura 10 se muestra el código fuente para visualizar la señal digital y su representación espectral.

```
for(u=0;u<N;u++){//Código fuente necesario para representar S
aux1=S_real[u];//Variable auxiliar
aux2=S_imag[u];//Variable auxiliar
aux3=sqrt(pow(aux2,2)+pow(aux1,2));//Módulo de S mediante la librería math.h
ftoa(aux3,bufer2,5,'f');//El número del módulo S se transforma a string
                        //mediante la libreria ftoa.h
SetColor(BRIGHTRED);//Determina el color de la señal de espectro.
a=(int)(aux3*5);
if(a>=230){a=230;}//Hay veces que el módulo S sale de la escala
Line(u,230,u,230-a);//en pixel del LCD TOUCH
if(conta<10){a=50;}
if(conta>=10){a=0;}
if(conta==20){conta=0;}
SetColor(BRIGHTGREEN);//Determina el color de la señal digital.
Line(u+130,230,u+130,230-a);
conta++;}
sensor=(int)readADC(11); //Se obtiene valores analógicos del puerto 11
PORTA=sensor;//Valor de la señal digital.
SetFont((void *) &Font25);//Código fuente para representar valores en el LCD TOUCH
SetColor(BRIGHTGREEN);
sprintf(bufer2,"%d",sensor);
OutTextXY(10, 10, bufer2);//Código para representar textos o valores en el LCD TOUCH
DelayMs(DEMODELAY*1);//Tiempo de espera hasta el proximo muestreo
conta=0;
SetColor(BLACK); //Se pone en negro el LCD TOUCH
ClearDevice();//Se limpia la pantalla LCD TOUCH
}
```

readADC(11)

$$y_u := \sum_{k=0}^{M1-1} \left(bc_{k,2} \cdot a_{u-k}\right) \cdot K$$

Se puede reemplazar sensor = y_u

Fig. 10. Código fuente que visualiza la señal filtrada: FIR lineal o el espectro de Fourier.

En la izquierda de la figura 11, se visualiza el resultado del código fuente para representar la función tren de pulso en forma espectral. En la derecha se muestra el esquema de trabajo donde la salida de un generador de señales es conectada a la entrada analógica del PIC32.

Fig. 11. Señal digital y su representación espectral junto al esquema de trabajo.

Existe una limitación en la longitud de los vectores. En el caso de los PIC32 es de 512 y en los dsPIC es de 128. Es posible aumentar este número en ambos casos, solo se debe cambiar las condiciones de compilación y linkeo de los microcontroladores para liberar más espacio en la memoria de programa.

8. Implementación a otros dispositivos

El desarrollo del algoritmo permite desarrollar filtros FFT y FIR en microcontroladores del tipo dsPIC y PIC3. Su código fuente es abierto, de alto nivel y posibilita su generalización en otros dispositivos, como por ejemplo los mostrados en la Figura 12, 13 y 14.

Fig. 12. Dispositivo genérico Starter Kit Board diseñado para procesar sonido [7].

El Starter Kit Board es un desarrollo de trabajo en áreas específicas como por ejemplo el sonido. El PIC24F Starter Kit 1 y el Starter Kit Board pueden medir aceleraciones y manejar datos a partir de dispositivos de almacenamiento, como por ejemplo un pendrive. Debido a que la estructura de hardware y software son simples en relación a la toma de información de datos, solo hay que aplicar el código fuente de la figura 9 y el de la figura 10, para el manejo y el procesamiento de la información. Los resultados obtenidos se visualizan directamente en el display gráfico o en un archivo de datos.

Fig 13. Dispositivo PIC24F Starter Kit para procesar sonido y aceleración triaxial [12].

Fig 14. Dispositivo PIC24H Starter Kit para manejar datos, gráficos y teclado [12].

Los desarrollos mencionados son ideales en aquellas necesidades donde se quieran analizar fallas, mantenimiento preventivo o almacenamiento de datos (dataloger). El esfuerzo y el tiempo de desarrollo (del software) son bajos y el conocimiento para su implementación por parte del programador, es mínimo.

9. Conclusión

Se implementó el algoritmo FFT en microcontroladores dsPIC y PIC32 mediante la reutilización de software, permitiendo disminuir el esfuerzo y tiempo en

programación. Los conocimientos necesarios para su implementación son mínimos por parte del programador, permitiendo generalizar el uso de los algoritmos de filtrado con los dispositivos genéricos utilizados. La implementación del algoritmo FFT en microcontroladores se vio facilitada debido a las librerías de uso matemático y números complejos. La implementación de filtros FIR lineal requiere menos cálculo computacional que la FFT. Existe una limitación en la longitud del vector donde se alojan los valores digitales provenientes de la operación de conversión analógica digital. Para el caso de los PIC 32 es de 512 y para los dsPIC es de 128. Es posible aumentar este número cambiando las condiciones de compilación y linkeo de los microcontroladores. Los resultados obtenidos del hardware mostrado en la figura 8 se pueden extender a otros desarrollos genéricos que utilicen dsPIC y PIC32.

10. Referencias

1. Pérez García A, Antón Álvarez C, Rodrígues Campo C, Ferrero Martín F: Instrumentación Electrónica. Editorial Thomson. Cap 1. pp 10. España. Madrid 2011.
2. P. Simonen and H. Olkkonen, “Fast method for computing the Fourier integral transform via Simpson’s numerical integration,” J. Biomed. Eng., vol. 7, pp. 337–340, Oct. 1985
3. T. E. Tuncer,“Causal and stable FIR-IIR Wienerfilters,” inProc. IEEE Workshop Statistical Signal Process. , St. Louis, MO, Sep. 28, 2003
4. Usategui A, Ruiz A, Martínes I, Parra I: dsPIC Diseño práctico de aplicaciones. Editorial Mac Graw Hill. Cap 1. pp 3.Tercera edición. España 2006. Madrid.
5. J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series,” Mathematics of Computation, vol. 19, no. 90, pp. 297–301, April 1965.
6. Y. Ma, “An effective memory addressing scheme for fft processors,” Signal Processing, IEEE Transactions on, vol. 47, no. 3, pp. 907–911, Mar 1999.
7. I. Uzun, A. Amira, and A. Bouridane, “Fpga implementations of fast fourier transforms for real-time signal and image processing,” Vision, Image and Signal Processing, IEE Proceedings -, vol. 152, no. 3, pp. 283–296, June 2005.
8. Di Jasio L: Programming 16-bit Microcontrollers in C. Learning to fly the PIC 24. Editorial Newnes. Cap 4. pp51. Primera Edición. EEUU 2007. Boston.
9. Di Jasio L: Programming 32-bit Microcontrollers in C. Exploring the PIC32. Editorial Newnes. Cap 7. pp 152. Primera Edición. EEUU 2008. Boston.
10. Naguil J, Gabotti D, Brac E, Gutierrez G: Diseño y Simulación de Filtros FIR de Fase Lineal en Matlab y Simuling. Revista de la U.T. N. República Argentina. Año 3. N°4. ISSN 16666933.
11. González C, Wood E: Tratamiento digital de imágenes. Editorial Addison-Wesley/Dias de Santos. Cap 3. pp131. Primera Edición. EEUU 1996. New York.
12. www.microchip.com/
13. Usategui A, Zapirain G, Martínez I, Sáez V: Microcontroladores Avanzados dsPIC. Controladores Digitales de Señales. Arquitectura, programación y aplicación. Editorial Thomson. Cap 14. pp 262. Primera Edición. España 2006. Madrid.