Implementación de la transformada *Cepstrum* en hardware

Juliana Alejandra Arévalo Herrera[1]
Augusto Pedraza Bonilla[2]

Resumen

El tratamiento digital de señales es una de las áreas de la electrónica y de la informática de mayor auge de los últimos años. El auge reciente de plataformas de desarrollo más robustas permite implementar algoritmos de tratamiento digital más exigentes y más eficientes, con menor costo y menos espacio. Esta investigación muestra la forma en que se implementa la transformada cepstrum en hardware usando la transformada rápida de Fourier base cuatro.

Palabras clave

Transformada cepstrum, tratamiento digital, transformada Fourier, plataforma, desarrollo procesamiento de voz.

Abstract

The digital signal processing is one of the most important areas of electronics and system engineering. The recent peak of robust development platforms, permits develop more efficient algorithms, with low cost and

1 Estudiante de la Facultad de Ingeniería de Telecomunicaciones. Universidad Santo Tomás.

2 Ingeniero Electrónico, Universidad Santo Tomás. Magíster en Ingeniería Electrónica y de Computadores, Universidad de Los Andes. Docente de Ingeniería Electrónica e Ingeniería de Telecomunicaciones de la Universidad Santo Tomás.

space. This investigation shows a way to implement the cepstrum transform in hardware using the four radix fast Fourier transform.

Key words

Cepstrum transform, digital processing, Fourier transform, platform, development, voice processing.

Introducción

El avance tecnológico de las últimas décadas ha permitido crear plataformas robustas que facilitan la implementación de diferentes algoritmos para el procesamiento y análisis en diversas aplicaciones. Entre los algoritmos de mayor aplicación en ingeniería se encuentra la transformada rápida de Fourier (FFT), con la que se representan señales en el dominio de la frecuencia de forma unívoca. A continuación se presentan los principales conceptos relacionados con el cálculo de una FFT base 4 de 1024 puntos así como su implementación en hardware por medio de un FPGA de la familia Spartan 3 de XILINX, escogido por su alta capacidad de procesamiento y posibilidad de optimización del sistema.

Objetivos

General

- Diseñar un hardware sobre una plataforma FPGA para el desarrollo de la transformada rápida de Fourier de 1024 puntos.

Específico

- Usar los componentes internos especializados de los FPGA para el tratamiento digital de señales, tales como multiplicadores, DLLs y memorias.

- Entender los conceptos básicos de tratamiento digital de señales en FPGA.

Metodología

Pruebas del algoritmo con voces en ato nivel

En primera instancia se verifica el funcionamiento del algoritmo en un lenguaje de alto nivel.

Aunque es posible calcular la transformada por medio de coeficientes de predicción lineal, fue escogido el método planteado originalmente, es decir, calculada por medio de transformada de Fourier; debido al previo conocimiento para desarrollar herramientas como la transformada rápida de Fourier y su implementación en hardware.

Diseño del hardware para la implementación de la FFT y el algoritmo

Posteriormente, se realizan pruebas y simulaciones con señales de voz, todas en un lenguaje de alto nivel. Así mismo, se realiza un diseño preliminar del hardware y se simula en alto nivel, para verificar su funcionalidad. Estos algoritmos que se prueban tienen una estructura similar a la de los diseños VHDL para probar su efectividad. Una vez realizadas las principales simulaciones, se inicia el diseño de los bloques de funcionales de la transformada en el lenguaje VHDL, que han entregado buenos resultados durante las simulaciones, comparadas con los algoritmos desarrollados en alto nivel.

Implementación y depuración del diseño de hardware

Los diseños de FFT y logaritmo realizados para un FUGA se depuran mediante un recolector de datos de hardware y una interfaz serial al PC, donde mediante una aplicación desarrollada en JAVA se verifica la eficiencia y funcionamiento del sistema. Aquí se ha escogido JAVA debido a su compatibilidad entre los sistemas operativos Linux y Windows.

La figura 1 muestra un flujo de diseño propuesto para dicha implementación basada en el flujo general de diseño de hardware.

Finalmente, la transformada completa será implementada sobre el FUGA para realizar las pruebas finales y ajustes necesarios.

Marco teórico

Transformada discreta de fourier

Gran cantidad de aplicaciones requieren el uso de las técnicas de Fourier para el tratamiento de señales, como los sistemas de radares en comunicaciones, aplicaciones médicas, como una alternativa a la convolución en el dominio del tiempo y principalmente análisis espectral, pues representa señales en el dominio de la frecuencia. Para fines computacionales, es conveniente el uso de una representación discreta de la transformada de Fourier, dando lugar a la DFT (Discrete Fourier Transform), expresada como:

$$X(k) = \sum_{n=0}^{N} x(n) e^{-jkn} \qquad (1)$$

Que es la transformada de Fourier de la secuencia, muestreada en frecuencias determinadas por *k* (múltiplo de), limitando el número de muestras calculadas desde 0 hasta un valor N, así la DTF no es más que la secuencia de los coeficientes de Fourier de la señal original. Se observa claramente que el cálculo de la DFT requiere un total de N^2 adiciones y multiplicaciones complejas, que la hacen computacionalmente ineficiente.

El algoritmo FFT (Fast Fourier Transform) fue introducido por Cooley y Turkey en 1965, reduciendo considerablemente el número de operaciones requeridas para el cálculo de la DFT, por medio de la técnica "divide y vencerás", en la que se agrupa las muestras y se calculan transformadas con pocos puntos de forma iterativa. El proceso se repite según el número de puntos y el algoritmo específico utilizado.

Algoritmo FFT Cooley Turkey

La transformada discreta de Fourier puede ser vista como una transformación lineal de la siguiente forma:

(2)

Donde $W_N = e^{-j2\Pi/N}$ y son usualmente llamados factores de giro.

Asumiendo que $N = N_1 \cdot N_2$, es posible reorganizar las muestras en una matriz de estas dimensiones, como lo representa la figura 1.

x_0	x_2	⋯	x_{N_1-1}
x_{N_1}	x_{N_1+1}	⋯	x_{2N_1-1}
⋮	⋮	⋱	⋮
$x_{N_1(N_2-1)}$	$x_{N_1(N_2-1)+1}$	⋯	$x_{N_2N_1-1}$

Figura 1. Arreglo de muestras en matriz de dimensiones $N_1 \times N_2$

De esta forma, la DTF puede ser calculada por filas, es decir:

$$X(k)=\sum_{n=0}^{N_1-1}x(n)W_N^{nk}+\sum_{n=N_1}^{2N_1-1}x(n)W_N^{nk}+\mathrm{L}+\sum_{n=(N_2-1)N_1}^{N_2N_1-1}x(n)W_N^{nk} \quad (3)$$

Cada una de estas DTF se calcula al dividir la secuencia en secuencias más pequeñas hasta que pueda aplicarse una DTF más sencilla, que deberá estar acompañada de factor determinado por el número de muestras y la muestra que se está operando.

FFT Base 4

El algoritmo puede manejar un tamaño arbitrario de las DFT, aunque valores adecuados de los factores de N para su implementación en hardware son las potencias de 2, pues permiten un manejo computacional más sencillo en relación a otros valores.

Si se tiene un número de muestras $N=4^{\nu}$ con $\nu\in\mathbb{N}$, es posible combinarlas por DFTs de 4 puntos para calcular la FFT. Si estas muestras se toman de forma consecutiva, se está utilizando el algoritmo de diezmado en frecuencia (DIF):

$$X(k)=\sum_{n=0}^{(N/4)-1}x(n)W_N^{nk}+\sum_{n=N/4}^{(N/2)-1}x(n)W_N^{nk}+\sum_{n=N/2}^{(3N/2)-1}x(n)W_N^{nk}+\sum_{n=3N/2}^{N}x(n)W_N^{nk} \quad (4a)$$

$$X(k)=\sum_{n=0}^{(N/4)-1}x(n)W_N^{nk}+W_N^{Nk/4}\sum_{n=0}^{(N/4)-1}x\left(n+N/4\right)W_N^{nk}+W_N^{Nk/2}\sum_{n=0}^{(N/4)-1}x\left(n+N/2\right)W_N^{nk}+W_N^{3Nk/4}\sum_{n=0}^{(N/4)-1}x\left(n+3N/4\right)W_N^{nk} \quad (4b)$$

Sin embargo, tenemos:

$$W_N^{kN/4}=e^{(-j\pi/2)k}=(-j)^k$$
$$W_N^{kN/2}=e^{(-j\pi)k}=(-1)^k \quad (5)$$
$$W_N^{3kN/4}=e^{(-j3\pi/2)k}=(j)^k$$

Reemplazando (5) en (4):

$$X(k)=\sum_{n=0}^{(N/4)-1}x(n)W_N^{nk}+(-j)^k\sum_{n=0}^{(N/4)-1}x\left(n+N/4\right)W_N^{nk}+(-1)^k\sum_{n=0}^{(N/4)-1}x\left(n+N/2\right)W_N^{nk}+(j)^kW_N^{3Nk/4}\sum_{n=0}^{(N/4)-1}x\left(n+3N/4\right)W_N^{nk} \quad (6)$$

Como los factores de giro están en función de N y no de N/4, que es el tamaño de la secuencia que se va a calcular, es necesario dividirla de nuevo en 4 secuencias de N/4 puntos:

$$X(4k),\ X(4k+1),\ X(4k+2)\ \text{y}\ X(4k+3):$$

$$X(4k)=\sum_{n=0}^{(N/4)-1}\left[x(n)+x\left(n+N/4\right)+x\left(n+N/2\right)+x\left(n+3N/4\right)\right]W_N^{0}W_{N/4}^{kn}$$
$$X(4k+1)=\sum_{n=0}^{(N/4)-1}\left[x(n)-j\cdot x\left(n+N/4\right)-x\left(n+N/2\right)+j\cdot x\left(n+3N/4\right)\right]W_N^{1}W_{N/4}^{kn}$$
$$X(4k+2)=\sum_{n=0}^{(N/4)-1}\left[x(n)-x\left(n+N/4\right)+x\left(n+N/2\right)-x\left(n+3N/4\right)\right]W_N^{2}W_{N/4}^{kn} \quad (7)$$
$$X(4k+3)=\sum_{n=0}^{(N/4)-1}\left[x(n)+j\cdot x\left(n+N/4\right)-x\left(n+N/2\right)-j\cdot x\left(n+3N/4\right)\right]W_N^{3}W_{N/4}^{kn}$$

La operación correspondiente en la FFT de base 2 es llamada mariposa, debido a su representación como diagrama de flujo. Como una analogía, llamaremos a la ecuación (6) libélula para diezmado en frecuencia, representadas en la figura 2. Como se observa, la libélula requiere de 12 sumas complejas y 3 multiplicaciones complejas, pues el primer punto no contiene componentes imaginarias. Esta operación se realiza para todos los puntos y de forma iterativa durante ν etapas con $N=4^{\nu}$, $\nu\in\mathbb{N}$. En la figura 3 se muestra el proceso para el cálculo de una DFT de 16 puntos diezmada en frecuencia.

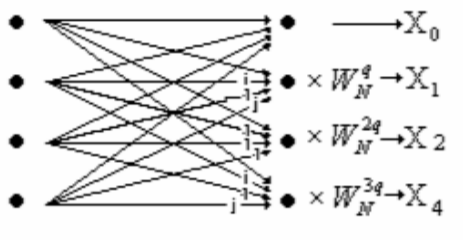

Figura 2a. Diagrama de flujo de la libélula.

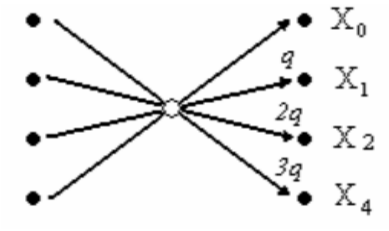

Figura 2b. Diagrama simplificado de la libélula.

El uso de este algoritmo implica la obtención de los puntos resultantes mezclados. Para ordenarlos, es necesario realizar una inversión de base 4, es decir que la posición de la muestra se representa en base 4 y se invierte el orden de los dígitos para encontrar la posición real. La tabla 1 muestra ejemplos de esta inversión de base 4.

Posición original	Representación en base 4	Inversión base 4	Nueva posición
$(26)_{10}$	$(122)_4$	$(221)_4$	$(41)_{10}$
$(52)_{10}$	$(310)_4$	$(013)_4$	$(7)_{10}$
$(95)_{10}$	$(1133)_4$	$(3311)_4$	$(245)_{10}$
$(120)_{10}$	$(1320)_4$	$(0231)_4$	$(45)_{10}$

Tabla 1. Ejemplos de inversión de base 4

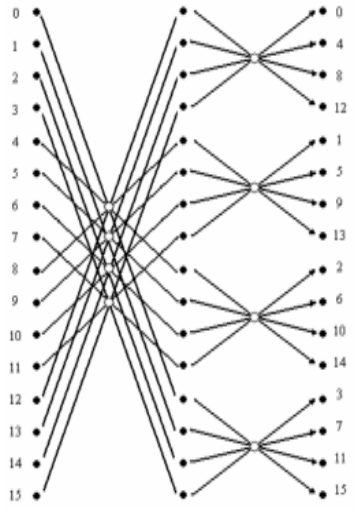

Figura 3. DFT base 4 de 16 puntos, diezmada en frecuencia.

Con base en el algoritmo Cooley-Turkey y el cálculo de la libélula, se propone la implementación de la FFT en un PLD (Programmable Logic Device) de tipo FPGA, que consisten en una matriz bidimensional de bloques configurables, altamente concentrados, que pueden ser interconectados por pistas o multiplexores para implementar sistemas digitales. Los componentes básicos del FPGA son los bloques lógicos, las interconexiones y bloques de entrada/salida; aunque según el FPGA y el fabricante existen módulos que se introducen en el sistema.

La programación se realiza por diferentes métodos, aunque lo más frecuente es el uso de lenguajes de descripción de hardware (HDL), con los que se establecen células de memoria RAM que controlan transistores de paso, puertas de transmisión o multiplexores, según la función que se desee obtener.

Uno de los HDL mas conocidos es el VHDL (Very High Speed Integrated Circuit HDL) definido por el estándar IEEE 1076 de 1993, con el que se documentan, simulan y sintetizan diseños de circuitos digitales, desde tres enfoques diferentes: estructural, en el que describen interconexiones por bloques; funcional, con el que se describe la función que realiza y, el algorítmico, que se describe como un flujo de operaciones. Comúnmente, se utiliza una combinación de los tres métodos.

Diseño

Implementación en hardware de FFT

El cálculo de la FFT base 4 se realiza en un número de etapas equivalente al logaritmo en base 4 del número de puntos N y en cada una de ellas se realizan N/4 operaciones libélula, cuyos resultados son almacenados en memoria para ser tomados como datos de entrada para la siguiente etapa. El proceso seguido por el sistema se muestra en la figura 5.

La FFT base 4 de 1024 puntos será implementada en un FPGA de la familia Spartan 3 de Xilinx, se divide el diseño en diferentes bloques funcionales distribuidos como indica la figura 4, que realizarán el proceso del cálculo sucesivo de DTF de 4 puntos para las muestras almacenadas en la memoria de datos. Las muestras iniciales tendrán un tamaño de 8 bits y los datos procesados ocuparán palabras de 32 bits divididos en parte real e imaginaria, con representación en punto fijo de 15 bits más el bit de signo. La libélula procesa los datos de entrada procedentes de la memoria, con factores de giro determinados por el exponente calculado en el bloque de control, y entrega el resultado de nuevo a la memoria para ser procesado en la siguiente etapa.

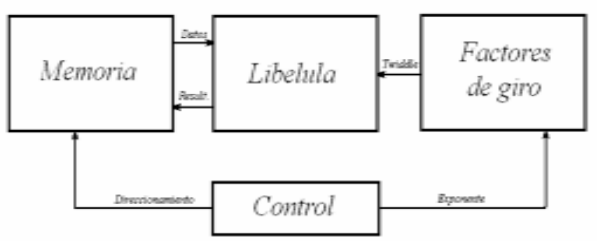

Figura 4. Diagrama de bloques FFT

Bloque de memorias

Los datos ingresados del conversor analógico-digital, se almacenan en una memoria de 1024x8. El almacenamiento de los datos procesados se realiza por medio de 2 bloques de memoria de 1024 x 32, de forma que se almacene la parte real e imaginaria (cada una de 16 bits) en la misma posición. Se requiere la utilización de 2 bloques de memoria, pues no es posible reemplazar los datos almacenados por los procesados por la libélula y es necesario guardarlos en un espacio diferente, de esta forma, cada etapa lee y escribe en memorias diferentes, como se muestra en la tabla 2. Esta configuración es lograda con 5 memorias del FPGA, cuyas características son mostradas en la tabla 3.

Etapa	1	2	3	4	5
Lectura de	Datos	A	B	A	B
Escritura en	A	B	A	B	A

Tabla 2. Escritura y lectura de la memoria según la etapa.

Nombre de la primitiva	Generalidades	Cantidad	Almacena	Utilización
RAMB16_S36	512 x 32 + 4 bits de paridad Sincrónica, puerto sencillo	4	Datos procesados	100%
RAMB16_S9	2k x 8 + 1 bit de paridad Sincrónica, puerto sencillo	1	Datos de entrada	50 %

Tabla 3. Memorias del FPGA utilizadas en la implementación.

Factores de giro

Entre los exponentes para el cálculo de la libélula existe una relación directa, pues

$$W_N^{2q} = 2 * W_N^q$$
$$W_N^{3q} = 3 * W_N^q \quad (8)$$

Y cada uno de los valores es entregado en uno de los 4 flancos de reloj necesarios para operar la libélula.

Los factores de giro son representados por palabras de 32 bits, de los cuales 16 corresponden a parte real y los 16 restantes a parte imaginaria, representados en punto fijo. La representación se logra multiplicando por una constante que permita utilizar 15 bits (2^{15}) y reservar uno para el signo. El resultado de esta operación se redondea y se almacena en

memoria como una constante entera que será entregada a la libélula.

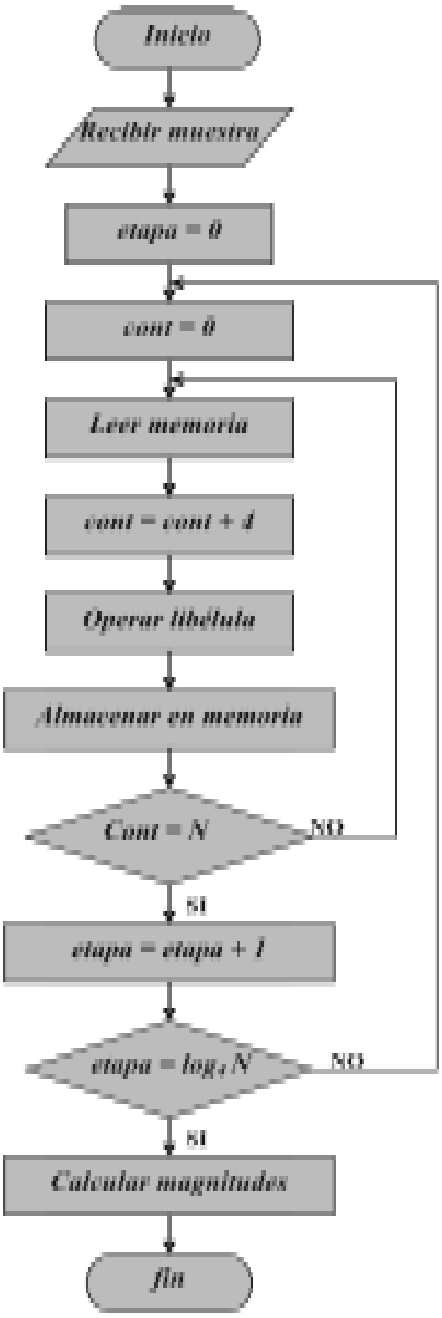

Figura 5. Diagrama de flujo para el cálculo de FFT de base 4

Durante simulaciones realizadas previamente se encontró una relación entre los valores de los factores de giro, que pueden ser representados por 129 configuraciones de bits. esta propiedad se aprovechó para reducir el espacio en memoria requerido para su almacenamiento. Así solo son necesarios 128 posiciones de memoria, cada una de 32 bits, que serán direccionados según el exponente calculado en el bloque de control. El factor de giro restante se presenta cuando el exponente tiene un valor de 0 o de 128 y su valor es unitario, por lo que no es necesario el almacenamiento en memoria.

El factor de giro entregado a la libélula, se debe formar con el valor obtenido de la posición de memoria. El direccionamiento y la conformación del factor de giro se detallan en la tabla 4, con una representación de los números negativos realizada en complemento a 2.

Exponente	Dirección	Tw real	Tw imag
0	N.A	3FFF	0000
0<w<128	w_x	H	-L
128	N.A	0000	8001
128<w<256	$-w_x$	L	-H
256<w<384	w_x	-L	-H
383<w<512	$-w_x$	-H	-L
512<w<642	w_x	-H	L
w>641	$-w_x$	-L	H

Tabla 4. Direccionamiento y organización de factores de giro

Cálculo de libélula

Si representamos los datos de entrada como a, b, c y d, los resultados como A, B, C y D y los factores de giro como W^1, W^2 y W^3 tenemos:

$$
\begin{aligned}
A &= a + b + c + d \\
B &= \left((a - c) + j(d - c)\right) \cdot W^{1} \\
C &= (a - b + c - d) \cdot W^{2} \\
D &= \left((a - c) + j(b - d)\right) \cdot W^{3}
\end{aligned}
\qquad (9)
$$

Como se indicó anteriormente, la libélula se calcula por medio de 3 multiplicaciones complejas y cada una de ellas realiza 4 multiplicaciones para dividir la parte real de la imaginaria, por lo que es necesario el uso de 12 multiplicadores, cuya salida será posteriormente combinada para obtener los resultados de salida. Así, la libélula está formada por 3 bloques combinacionales, dos de suma y uno compuesto de multiplicadores. Puesto que cada salida de la memoria de datos se realiza con un flanco de reloj, son necesarios 4 flancos para obtener todas las muestras que van a ser operadas. Sin embargo, al ser combinacional, la libélula está cambiando sus salidas cada vez que se realiza un cambio en los datos de entrada, por lo que es necesario un registro que almacene los resultados cada 4 flancos para ser posteriormente entregados a la memoria.

Cada uno de los 12 multiplicadores utilizados se encuentran previamente creados como bloques de hardware en el FPGA y cuentan con 2 entradas de 18 bits y una salida de 36 bits, de los cuales se toma del bit 15 al 29 (siendo el bit 35 el más significativo) y se concatenan con el bit 35 que representa el signo. Los bits menos significativos pueden ser despreciados y así dividir por el factor que acompaña los factores de giro.

Bloque de control

El bloque de control es el encargado de generar las señales que manejan los otros bloques, a partir de un contador de 0 a 1029. Este valor es necesario para entregar las 1024 muestras a ser operadas por la libélula y permitir que los datos calculados en la última libélula de cada etapa se almacenen en la memoria. Las señales generadas a partir del contador son:

- Dirección de lectura: reorganizando los bits del contador según la etapa.
- Dirección de escritura: tomando el valor de la dirección de lectura, retardado 6 ciclos de reloj, necesarios para obtener los datos de memoria y realizar el cálculo de la libélula.
- Etapa: aumenta cuando el valor del contador es máximo (1029) y reinicia el contador.
- Exponente del factor de giro: calculado como aumento progresivo hasta 252, según la etapa que se esté calculando.

Los cambios del contador se hacen con el flanco negativo del reloj, para asegurar que el direccionamiento de la memoria se encuentre listo en el flanco positivo, que es cuando se entregan los datos.

Adicionalmente, se tiene un bloque de interfaz entre la libélula y la memoria, que permite el almacenamiento o entrega de los datos cada 4 flancos de reloj.

Reorganización de los datos

Al procesarse como diezmado en frecuencia, la FFT entrega los datos mezclados y es necesario reorganizarlos con una inversión de dígitos en base 4. Sin embargo, existe una relación directa entre el sistema de base 4 y el sistema binario y solo debe cambiarse el direccionamiento de las muestras obtenidas de la memoria reorganizando los bits del contador, como se indica en la tabla 5.

Direccionamiento	$Addr_9$	$Addr_8$	$Addr_7$	$Addr_6$	$Addr_5$	$Addr_4$	$Addr_3$	$Addr_2$	$Addr_1$	$Addr_0$
Contador	$Cont_1$	$Cont_0$	$Cont_9$	$Cont_8$	$Cont_7$	$Cont_6$	$Cont_5$	$Cont_4$	$Cont_3$	$Cont_2$

Tabla 5. Reorganización de los bits del contador para la inversión base 4.

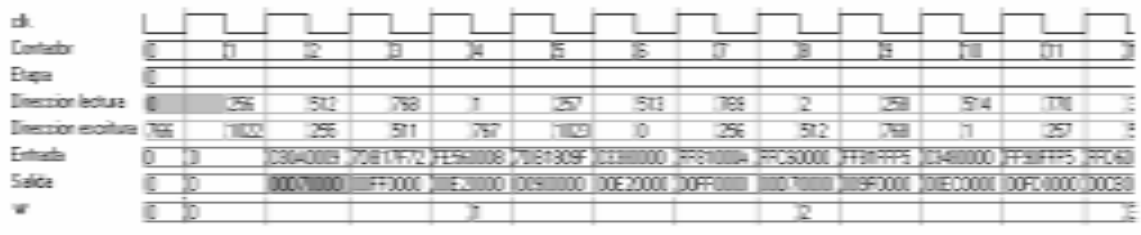

Figura 6. Simulación de las señales de control.

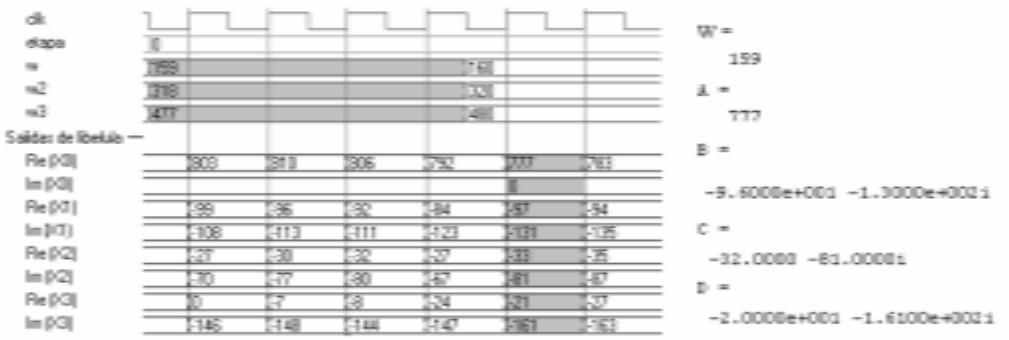

Figura 7. Simulación de los datos entregados por la libélula y datos obtenidos en MATLAB.

Simulaciones

Actualmente se encuentran codificados los primeros 4 bloques de la FFT, que ha sido simulada con una señal de prueba. Las señales de control se muestran en la figura 6.

Las señales **entrada** y **salida** representan los datos de entrada y salida de la memoria, aunque el valor de la posición indicada por la dirección aparece después de dos flancos positivos, como se observa con las áreas sombreadas, por lo que el cálculo de la libélula se realizan 6 flancos después de que se entrega la dirección.

Los valores obtenidos en cada uno de los pasos del proceso se han comparado con resultados obtenidos de un algoritmo creado para el paquete MATLAB en el que se observa el resultado para cada cálculo de la libélula. La figura 7 muestra una comparación entre los datos obtenidos con el simulador y los entregados por el algoritmo. Claramente se ve que las salidas de la libélula cambian cada vez que se presenta un cambio en el contador y, por lo tanto, en sus entradas. Es importante realizar la adquisición de los datos de salida en el momento apropiado cada 4 ciclos de reloj, y 6 ciclos después de iniciada la entrega de datos.

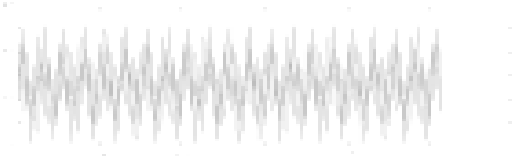

Figura 8. Simulación del algoritmo FFT base 4 de 1024 puntos en MATLAB.

Se observa una pequeña diferencia entre los datos obtenidos del simulador y los entregados por el algoritmo, debido a que en la implementación se desprecian los bits menos significativos. Esta diferencia es despreciable para los valores trabajados, incluso en etapas posteriores.

El algoritmo implementado en MATLAB se creó siguiendo el proceso descrito en la figura 5, de forma que se espera un comportamiento similar al de la implementación presentada. La figura 8 muestra el cálculo de la FFT de 1024 puntos de base 4 con el algoritmo de MATLAB, para una señal compuesta de 3 tonos de 200 Hz, 800 Hz y 2300 Hz. La señal de prueba ha sido limitada a valores entre 0 y 255, pues son los valores que puede almacenar la memoria de datos, proveniente de un conversor analógico-digital de 8 bits.

La cantidad de puntos permite al algoritmo tener una buena precisión, a pesar de estar manejando datos redondeados (parte entera) para asegurar una correcta aproximación a los resultados que se obtendrán de la implementación en hardware.

Resultados

Se propone el uso de un oscilador de 50 MHz, con lo que se obtendría un cálculo de FFT cada 102,9 µs, y por tanto, 9718 FFT por segundo. Esta velocidad puede ser aumentada si se implementan varias etapas en paralelo o aumentando la frecuencia del oscilador.

Esta propuesta se presenta como primer paso para la implementación de la transformada Cepstrum, definida como la transformada de Fourier del logaritmo del espectro de una señal, introducida por Bogert en 1963, con el que puede observarse el efecto de ecos tardíos en una señal.

Bibliografía

CHILDERS, Donald G. SKINNER, David P. y KEMERAIT, Robert C. (1977). *The cepstrum : a guide to processing. Proceedings of the IEEE*, Vol. 65, No 10. Octubre.

GOLDBERG, Randy G. (2000). *Practical handbook of speech codes*. CRC Press LCC.

PROAKIS, John G. y MANOLAKIS, Dimitris G (1997). *Tratamiento digital de señales*. Prentice Hall.

RABINER, Lawrence R. y SHAFER, Ronald W. (1978). *Digital processing of speech signals*. Prentice Hall.