IMPLEMENTACIÓN EN HARDWARE DE LA TRANSFORMADA WAVELET USANDO FPGA

Fredy H. Riascos-Campiño, Alexander García-Quinchia, Jaime Velasco-Medina*

Grupo de Bionanoelectrónica, Escuela EIEE, Universidad del Valle, Cali, Colombia.
*GDSPROC, Universidad del Quindío, Armenia, Colombia.

E-mail: gdsproc@uniquindio.edu.co, jvelasco@univalle.edu.co

RESUMEN

La transformada wavelet discreta (DWT) es una función fundamental en el procesamiento digital de señales, debido a su aplicación en sistemas de reducción de ruido, reconocimiento de patrones, encriptación de datos y compresión de voz, imágenes y video. En este trabajo se presenta el diseño de varias arquitecturas hardware para la implementación eficiente de la DWT usando FPGA.

1. INTRODUCCIÓN

El interés por la trasformada wavelet se ha incrementado notablemente en los últimos años, en consecuencia se esta utilizando en aplicaciones tales como: reducción de ruido, reconocimiento de patrones, encriptación de datos y compresión de voz, imágenes y video. El primer artículo que presenta la transformada wavelet fue publicado en 1984 [1]. Desde entonces, se han propuesto numerosas arquitecturas hardware que son presentadas en diferentes artículos [4]. Sin embargo, la gran mayoría de las implementaciones son realizadas considerando diseño VLSI debido a la gran cantidad de recursos hardware que se requieren para implementar esta transformada. Entonces, una excelente alternativa de diseño para implementar la transformada wavelet en hardware es usar los FPGAs.

En este artículo se presentan tres arquitecturas hardware para la implementación de la transformada wavelet usando FPGAs, en primera instancia se utilizan las estructuras polifase y rejilla, pues se caracterizan por un buen desempeño en cuanto a la velocidad de cálculo, y por ultimo se propone una arquitectura basada en el algoritmo piramidal que ocupa poca área y presenta buen desempeño en cuanto a velocidad.

2. LA TRANSFORMADA WAVELET DISCRETA

Para el cálculo de la DWT existe un algoritmo eficiente conocido comúnmente como Algoritmo Piramidal de Mallat [2]. El algoritmo descompone una señal de entrada $a_0(n)$ en dos componentes, generalmente llamadas aproximaciones y detalles, este proceso se repite en forma iterativa para obtener *i-etapas* de descomposición. Por consiguiente, por cada nivel u octava *i* se tiene una señal de aproximaciones $a_{i+1}(n)$ y una señal de detalles $d_{i+1}(n)$ dadas por las ecuaciones (1) y (2):

$$a_{i+1}(n) = \sum_{k=0}^{M-1} h(k) \cdot a_i(2n-k) \quad (1)$$

$$d_{i+1}(n) = \sum_{k=0}^{M-1} g(k) \cdot a_i(2n-k) \quad (2)$$

Las ecuaciones (1) y (2) muestran que la DWT puede ser expresada como la convolución de g*(n) y h(n)*, con la señal a_i submuestrada en un factor de dos [3]. En este sentido, la DWT corresponde a un banco de filtros FIR pasa bajo y pasa alto con respuesta al impulso g*(n)* y *h(n)* donde la salida de estos filtros es submuestrada por un factor de dos para obtener las aproximaciones y los detalles en cada etapa. En Fig. 1 se muestra un diagrama de bloques para el banco de filtros con tres niveles de descomposición.

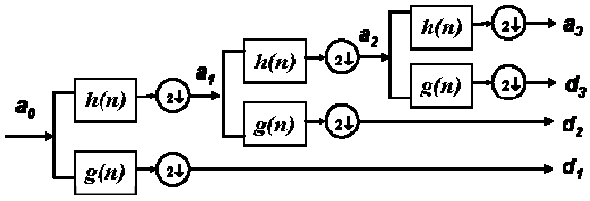

Fig. 1. Banco de filtros con tres niveles de descomposición.

3. ESTRUCTURAS PARA EL CÁLCULO DE LA DWT

Inicialmente para el calculo de la DWT se han seleccionado las estructuras polifase y rejilla, ya que presentan buena velocidad de cálculo.

3.1. Estructura de rejilla

El diagrama básico para un banco de filtros de descomposición basado en la estructura de rejilla (*lattice*) se muestra en Fig. 2. Los parámetros *s*, α[1] y

α[0] se obtienen comparando la función de transferencia de las ecuaciones (3) y (4) con la de los filtros pasa bajo *G(z)* y pasa alto *H(z)*, que para este caso tienen longitudes *M=4* [3].

$$G(z) = ((1+\alpha[0]) + \alpha[0]z^{-1} + \alpha[0]z^{-2} + (1+\alpha[0])z^{-3})s \quad (3)$$

$$H(z) = (-(1+\alpha[0]) + \alpha[0]z^{-1} + \alpha[0]z^{-2} - (1+\alpha[0])z^{-3})s \quad (4)$$

La estructura de rejilla presenta varias ventajas, tales como: modularidad, escalabilidad y baja complejidad. Además, mientras que en una implementación directa del algoritmo piramidal se utilizan *2M* multiplicadores y *2M-2* sumadores, la estructura en rejilla requiere solo *M+1* multiplicadores y M sumadores [5].

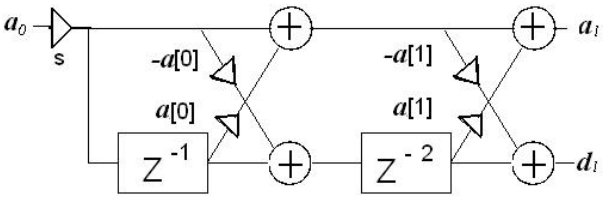

Fig. 2. Estructura en rejilla para una octava.

3.2. Estructura polifase

La estructura para polifase es muy utilizada cuando se realiza diezmado, interpolado o bancos de filtros. Esta consiste en descomponer la señal y el filtro en sus diferentes componentes polifase [3]. En este caso, esta estrategia se puede utilizar para el banco de filtros descrito por el algoritmo de Mallat. La señal de entrada se puede expresar como la suma de sus componentes polifase desplazadas y submuestreadas en un factor de dos y se denota como :

$$a^{(i+1)}(z) = a_0^{(i-1)}(z^{-2}) + z^{-1}a_0^{(i-1)}(z^2) \quad (5)$$

Donde las componentes polifase de entrada, definidas en el dominio z, se pueden expresar a través de las ecuaciones 6 y 7[3]:

$$a_0^{(i+1)}(z) = \sum_{n=-\infty}^{\infty} a^{(i-1)}(2n)Z^{-n} \quad (6)$$

$$a_1^{(i-1)}(z) = \sum_{n=-\infty}^{\infty} a^{(i-1)}(2n+1)Z^{-n} \quad (7)$$

De igual manera, los coeficientes de los filtros se pueden escribir como:

$$G(z) = G_0(z^{-2}) + z^{-1}G_1(z^{-2}) \quad (8)$$

$$H(z) = H_0(z^{-2}) + z^{-1}H_1(z^{-2}) \quad (9)$$

En las ecuaciones (8) y (9), los coeficientes de los filtros polifase, por ejemplo G_0 y G_1, se pueden obtener fácilmente tomando los coeficientes de índice par e impar, respectivamente, de los filtros originales. La Fig. 3 muestra diagrama de bloques básico para la estructura polifase para un filtro FIR. (M=8). Note que para una etapa de descomposición se requiere filtros pasa bajo y pasa alto.

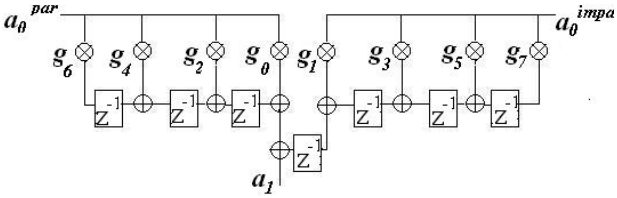

Fig. 3. Estructura polifase para un filtro FIR

4. ARQUITECTURA BASADA EN EL ALGORITMO PIRAMIDAL

La DWT se calcula por medio del algoritmo piramidal de Mallat. En este caso, cada etapa u octava *J* de la DWT se puede calcular a través de un filtro pasa bajo y pasa alto, cuya salida se submuestrea por un factor de dos. El filtrado se puede implementar mediante un único procesador de propósito general en forma iterativa o también en una arquitectura multiprocesador de *J* procesadores, donde cada procesador calcula las salidas del filtro pasa bajo y pasa alto de una octava. Considerando lo anterior se propone un esquema con *2J* unidades de Multiplicación Acumulación MAC, es decir, cada etapa cuenta con dos unidades MAC que trabajan en paralelo. El filtrado se realiza utilizado el esquema de Multiplicación Acumulación y Desplazamiento MACD.

4.1. Esquema MACD.

Un filtro FIR se puede implementar usando la sumatoria de convolución. Esta operación se expresa matemáticamente de la forma:

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) \quad (10)$$

Donde los coeficientes del filtro corresponden al vector *h(n)*, con $0 \le n < M$, siendo *M* el orden del filtro. Lo anterior significa que para calcular la salida actual es necesario conocer la entrada actual, además *M-1* historias de la entrada y los coeficientes *h(n)*. Esto supone que la salida *y(0)* esta dada por:

$$y(0) = \sum_{k=0}^{M-1} h(k)x(-k) =$$

$$h(0)\cdot x(0) + h(1)\cdot x(-1) + h(2)\cdot x(-2) + .. + h(M-1)\cdot x(-(M-1))$$

El elemento *x(0)* es la entrada actual y los demás elementos de índices negativo indican elementos de retardo o entradas pasadas, es decir, la multiplicación debe hacerse con elementos de memoria. Se requiere una memoria RAM para los elementos de retardo y una memoria ROM para los coeficientes del filtro. Por cada salida se realiza un desplazamiento de una posición en la memoria RAM, esto se logra a través del esquema de Multiplicación Acumulación y Desplazamiento.

El procedimiento para realizar el filtrado con el esquema MACD se explica como sigue: Inicialmente se realiza la multiplicación y acumulación de *x(-(M-1))* y *h(M-1),* estos valores se direccionan con el puntero *Ra*, en ese mismo instante se ingresa el dato actual *x(0*), el puntero *Rb* apunta a la posición donde se escribe tal dato como se muestra en Fig.4. El segundo paso es

multiplicar y acumular los elementos *x(-(M-2))* y *h(M-2)*. En este caso no se ingresa un nuevo elemento, sino que se copia *x(-(M-2))* a la siguiente posición de memoria, es decir donde estaba *x(-(M-1))*. Este procedimiento de multiplicación y copia se repite M veces, al final, la celda que contiene a *x(0)* sobrescribe la posición de la entrada anterior *x(-1)*, generando así un desplazamiento de toda la memoria y un espacio para un nuevo dato de entrada. Simultáneamente *y(0)* esta disponible en el acumulador y los punteros listos para que se reinicie el ciclo. En Fig. 4 se ilustra las operaciones realizadas al aplicar M veces la operación MACD. Note que hay un circulo resaltando una posición de memoria, esto indica que esta disponible para escribir un nuevo dato de entrada.

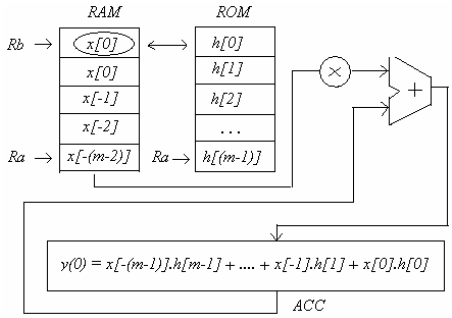

Fig. 4 Esquema de cálculo de filtrado con MACD

4.2. Arquitectura MACD

Para reducir el consumo de recursos hardware sin reducir dramáticamente la velocidad de cálculo, se propone una arquitectura denominada MACD. Esta arquitectura utiliza registros de segmentación para realizar en forma simultánea el cálculo de los dos filtros del algoritmo piramidal. En este caso, el camino de datos usa tres etapas pipeline: lectura/escritura de memoria, multiplicación y suma/acumulación, tal como se muestra en Fig. 5.

Esta arquitectura requiere una memoria RAM de doble puerto de M palabras, utilizada para almacenar los elementos de retardo, dos memorias ROMs para almacenar los M coeficientes de cada filtro, dos multiplicadores y dos unidades de suma y acumulación. Para realizar el cálculo de varias octavas se debe adicionar *J* etapas en cascada. La señal de reloj de la etapa *i* la proporciona la etapa *i-1*, a través de la señal *clk1* proporcionada por la unidad de control que realiza las siguientes acciones:

Inicio:
Ra ←*M y Rb* ← *0.*
Lectura- escritura
Leer *x[-(Ra-1),* Leer *h[(Ra-1)].*
Escribir *x(Rb)*
Actualizar *Rb* ←*Rb-1* y *Ra* ←*Ra-1*.
Si *Rb<0* entonces *Rb* ←*M*.
Si *Ra<0* entonces *Ra* ← *M*.
Multiplicación:
Multiplicar los datos leídos.
Suma-Acumulación:
Acumula el dato multiplicado
Salida:
Dato listo *y(n)* ←*ACC+MULTI*

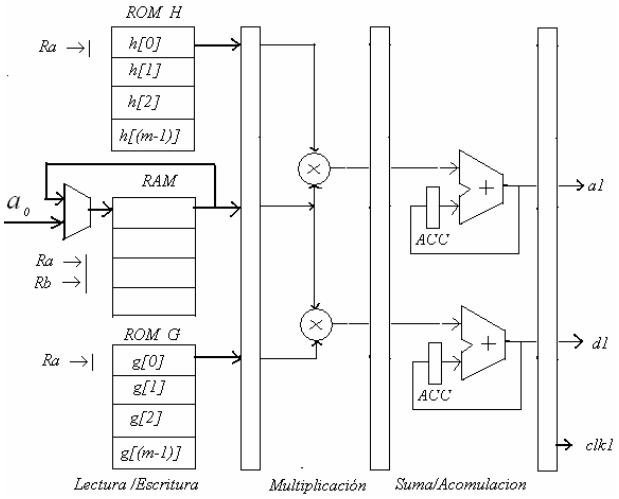

Fig. 5 Arquitectura tipo MACD.

La arquitectura MACD es optimiza el área al hacer uso de la recursividad, además es altamente escalable y versátil, pues los coeficientes de los filtros, necesarios para cada wavelet madre, son fácilmente configurables.

5. RESULTADOS DE SIMULACIÓN

Con el propósito de verificar el funcionamiento de las arquitecturas implementadas se realizaron simulaciones en QuartusII y Matlab. para tal fin se uso una FPGA Ciclone II de Altera (EP2C5T144C6) y una wavelet madre Daubechies 4 (*M*=8). En la tabla I se muestran los datos de verificación de la DWT obtenidos con la función DWT de Matlab, donde la función de entrada son los vectores v*1= 30:60* y *v2 = 0:60*. Para la simulación del hardware se utilizo el primer vector con la arquitectura MACD y el segundo con la arquitectura basada en la estructura polifase.

TABLA I. SIMULACIÓN EN MATLAB PARA DWT DEL VECTOR 30:60 Y 0:30

a3	***d3***	***a3***	***d3***
92.10	1.20	5.93	-2.52
88.13	-0.51	7.25	1.20
99.17	-0.00	3.28	-0.51 -
121.72	-0.00	14.30	0.00
144.35	0.00	36.80	-0.00 -
166.98	0.00	59.50	0.00
189.61	-0.00	82.10	-0.00
212.23	-0.00	104.70	-0.00
234.86	0.00	127.38	-0.00
257.49	0.010	150.01	0.0012

En Fig. 7 se muestra el resultado de la simulación

Para la arquitectura MACD, los valores coinciden con un error muy pequeño, excepto, los dos primeros valores que corresponden al transitorio del filtro. De manera similar en Fig. 8 se muestra la simulación para la arquitectura basada en la estructura polifase.

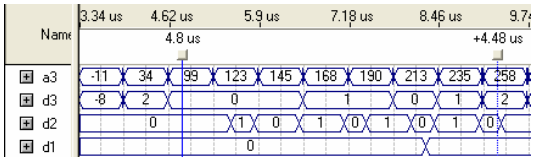

Fig. 7 DWT Arquitectura MACD.

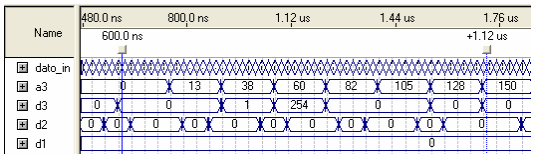

Fig. 8 DWT Arquitectura basada en la estructura polifase.

En la Tabla II se presenta el consumo de recursos para cada una de las arquitecturas, en este caso la información se discrimina por número de niveles de descomposición. Note en la tabla II que para las implementaciones en rejilla y polifase, algunas multiplicaciones se realizaron por medio de desplazamientos y sumas, lo cual causa que herramienta de síntesis no asigne multiplicadores internos.

TABLA II. CONSUMO DE RECURSOS

	MACD	Rejilla	Polifase
Un nivel			
LC	278 (6%)	340(10%)	920(20%)
Registros	251	250	594
Multiplicadores	2*	9* (18**)	16*
Dos niveles			
LC	562 (12%)	1020 (28%)	1212 (31%)
Registros	502	599	1188
Multiplicadores	4 *	18* (26**)	32*
Tres niveles			
LC	842 (18%)	2647 (50%)	2747 (60%)
Registros	753	906	1038
Multiplicadores	8*	27* (26**)	64*

* Numero de multiplicadores de 16 bits, **Multiplicadores embebidos de 9 bits

En la Tabla III se presenta el número de ciclos de reloj que cada arquitectura tarda para el cálculo de la DWT, la latencia por nivel y la cantidad de ciclos de reloj que implica el procesamiento de un bloque de 4096 datos de entrada. Adicionalmente, se incluye una columna con el número de ciclos de reloj utilizados por un DSP de punto fijo TMS320C5416 de *Texas Instruments* para procesar el mismo bloque de datos.

Tabla iii. Desempeño en ciclos de reloj

	MACD	Rejilla	Poli.	DSP
Un nivel				
Latencia	10	7	7	-
C. por muestra	16	2	2	-
Total ciclos*	32778	4103	4103	151709
Dos niveles				
Latencia	20	14	14	-
C, por muestra	32	4	4	-
Total ciclos*	32788	4106	4110	164104
Tres niveles				
Latencia	70	28	28	-
C. por muestra	64	8	8	-
Total ciclos*	32838	4116	4124	170355
Periodo clk (ns)	10	20	20	6.25
Tiempo calculo	328us	82.32us	82.48us	1.060us

* El número total de ciclos corresponde al tiempo requerido para procesar un bloque de 4K

6. CONCLUSIÓN Y TRABAJO FUTURO

En este trabajo se presentan tres arquitecturas hardware para la transformada wavelet: la MACD, la rejilla y polifase, las dos últimas presentan diferencias muy mínimas en cuanto a latencia y velocidad. Sin embargo, la estructura en rejilla es escalable, modular y consume menor cantidad de recursos. La arquitectura propuesta, MACD, consume menor cantidad de recursos y su desempeño en cuanto a velocidad es superior al procesador DSP y bueno comparado con las estructuras rejilla y polifase. Entonces, la estructura MACD puede ser considerada como una buena alternativa hardware usando un dispositivo FPGA de bajo costo.

Finalmente se plantea como trabajo futuro, la implementación de la DWT bidimensional y desarrollar implementaciones de los nuevos estándares basados en wavelet como por ejemplo JPEG2000.

REFERENCIAS

[1] R. Bopardikar, Wavelet Transforms. Addison-Wesley, 1998.

[2] I. Parroqué. Diseño de un ASIC para el Cálculo sin Pérdidas de la Transformada Wavelet Discreta en Aplicaciones Biomédicas. Tesis doctoral Universidad Politécnica de Valencia. 2001

[3] U. Meyer–Baese, Digital Signal Processing With Field Programmable Gate Arrays, Second Ed., Springer-Verlag 2003, pp.109-143.

[4] C. Chakrabarti, M. Vishwanath, R.M. Owens. *"A survey of architectures for the discrete and continuous wavelet transforms"***.** *Proc. of ICASSP-95*, Vol. 5, pp. 2849-2852, May. 1995.

[5] P.I Ramírez, Nuevas Estructuras RNS Para la Síntesis VLSI de Sistemas de Procesamiento Digital de Señales. Tesis Doctoral, Universidad De Granada.