Versión extraída de la original
Versión original impresa y publicada en 2008 – ISSN 1666-6917
Número 15 – Año 8 – año 2008
https://rtyc.utn.edu.ar/index.php/rtyc/article/view/1201

Desarrollo de filtros fir en fpgas

Juan Manuel LUZURIAGA, Pablo CAYUELA, Sergio OLMEDO, Guillermo GUTIERREZ
CUDAR-Centro Universitario de Automación y Robótica, Universidad Tecnológica Nacional,
Facultad Regional Córdoba, Córdoba, Argentina
IUA – Instituto Universitario Aeronáutico
jluzuriaga@scdt.frc.utn.edu.ar

Resumen - *Se describe el desarrollo de filtros FIR buscando maximizar la velocidad y minimizar la cantidad de celdas lógicas ocupadas en una FPGA. A tal efecto se investigan diversas técnicas de desarrollo en VHDL, implementándose los resultados para dos modelos de FPGAs de fabricantes diferentes: Spartan2s200 de Xilinx y FLEX10K70 de Altera; comparándose los resultados.*

Palabras clave: *Filtros FIR, wavelets, VHDL, dispositivos FPGA*

1. Introducción

En el marco del proyecto "Compresión de video con wavelet en lógica programable", se requiere el uso de bancos de filtros FIR para la implementación de las wavelet en una o más FPGA. Debido a los recursos disponibles al momento de realizar la investigación, es imperiosa la necesidad de un algoritmo realizado en lenguaje de descripción de hardware (VHDL), que permita la implementación de los filtros utilizando pocos recursos de los dispositivos programables. Con esta premisa y disponiendo de FPGA de dos empresas distintas (Xilinx y Altera), se requiere también que el código se pueda utilizar en cualquiera los distintos fabricantes. Las herramientas utilizadas son las proporcionadas por Xilinx para su programa universitario (ISE 6.3i y ModelSim Xilinx Edition II v5.8c) y por Altera en su versión estudiantil de MaxPlusII v10.22.

$$y[n] = x[n] * f[n] = \sum_{k=0}^{L-1} x[k] f[n-k] \qquad (1)$$

siendo *x[n]* la secuencia de entrada al filtro, *y[n]* su secuencia de salida, mientras que *f[0]* a *f[L-1]* son los *L* coeficientes del filtro, coincidentes con su respuesta impulsiva.

Aplicando transformación Z a la Ec. (1) se obtiene:

$$Y(z) = F(z)X(z). \qquad (2)$$

Aquí F(z) es la función de transferencia del filtro FIR en el dominio z.

$$F(z) = \sum_{k=0}^{L-1} f[k] z^{-k}, \qquad (3)$$

que puede escribirse como la serie finita:

$$F(z) = f_0 + f_1 z^{-1} + f_2 z^{-2} + f_3 z^{-3} + f_{L-1} z^{L-1}. \qquad (4)$$

El siguiente es un diagrama en bloques representativo del filtro en su forma directa, que se obtiene por aplicación de (4):

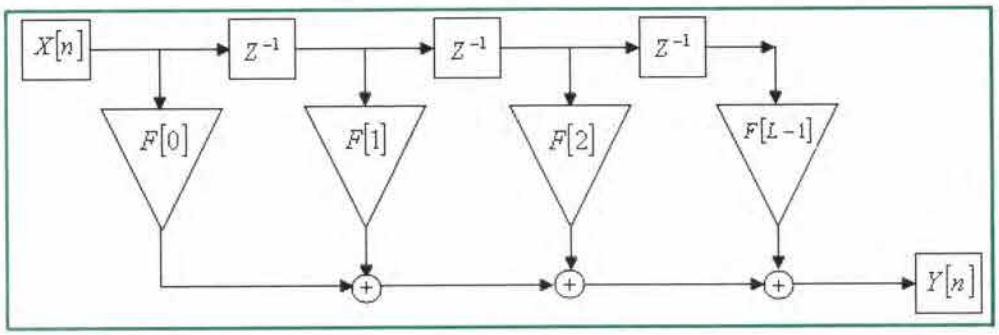

Fig. 1. Filtro FIR en forma directa.

Se va a emplear en este trabajo la *forma transpuesta*, puesto que mejora el rendimiento de la realización práctica.

2. Desarrollo de un filtro fir transpuesto programable

Se desarrolló un filtro FIR transpuesto programable que responde al diagrama de la Fig.2 donde A0... A3 son los registros que materializan los retardos z^{-1} de la Fig.1.

Los coeficientes Cin emplean cada uno 9 bits, por lo que se adoptan también 9 bits para representar los datos de entrada Xin. Si con width(Xin) se denota en forma generalizada la cantidad de bites empleados para los datos de entrada, entonces el multiplicador empleará 2×width(Xin) como ancho del bus de datos y el sumador 2×width(Xin) + 1.

Desde un punto de vista sistémico, la implementación del filtro consiste básicamente en cuatro procesos concurrentes:

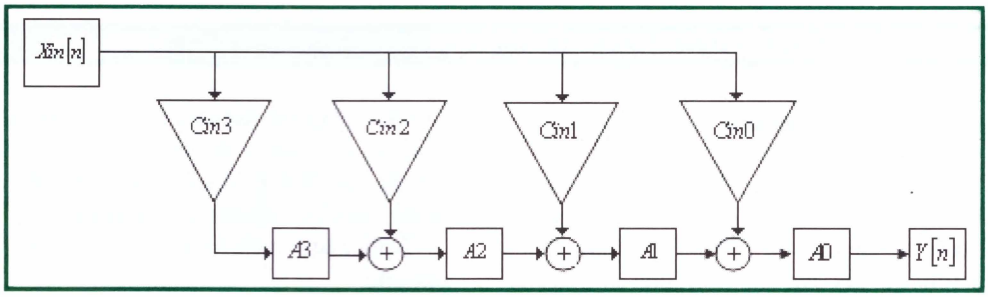

Fig. 2. Filtro FIR en forma transpuesta desarrollado

El filtro a implementar es el siguiente:

$$F(z) = 0.48301 + 0.8365z^{-1} + 0.2241z^{-2} - 0.1294z^{-3} \quad (5)$$

expresión para la que obviamente es $L = 4$. Se ha seleccionado la Ec. (5) por un conjunto de razones: en primer lugar por tratarse de un filtro de orden bajo, resulta realizable en FPGA con un mínimo de programación manteniendo el foco sobre el rendimiento (throughput) comparativo; la segunda razón está dada por la facilidad de implementación de los coeficientes empleando 8 bits –véase la expresión (6)– y, en último término, porque los coeficientes corresponden a un filtro pasabajos de reconstrucción asociado a una wavelet Daubechies DB4.

Si se cuantifican los coeficientes a 8 bits más un bit de signo, la formulación del filtro (5) se expresa como

$$F(z) = (124 + 214z^{-1} + 57z^{-2} - 33z^{-3})/256 \quad (6)$$

Los coeficientes correspondientes a la Fig. 2 toman los valores

Cin3 = 124
Cin2 = 214
Cin1 = 57
Cin0 = - 33

1 - Cargar las Constantes del Filtro y luego cargar los Valores de Entrada

2 - Calcular la Suma de Productos

3 - Calcular Productos

4-Convertir el resultado del filtro a 9 bits y asignarlo a la Salida.

A continuación se describen los procesos enumerados.

2.1. Proceso de carga de constantes de filtro y de datos

Para poder determinar si se están cargando las constantes del Filtro se usará una señal de control, que será mantenida en bajo hasta haber completado la carga de los 4 coeficientes. Estos últimos ingresarán comenzando por Cin3 y se desplazarán hasta la posición que deben ocupar en cada ciclo de reloj. Una vez finalizada la carga, por cada ciclo de reloj se ingresa una muestra al filtro, es decir se ingresa un dato.

La Fig. 3 grafica lo expuesto.

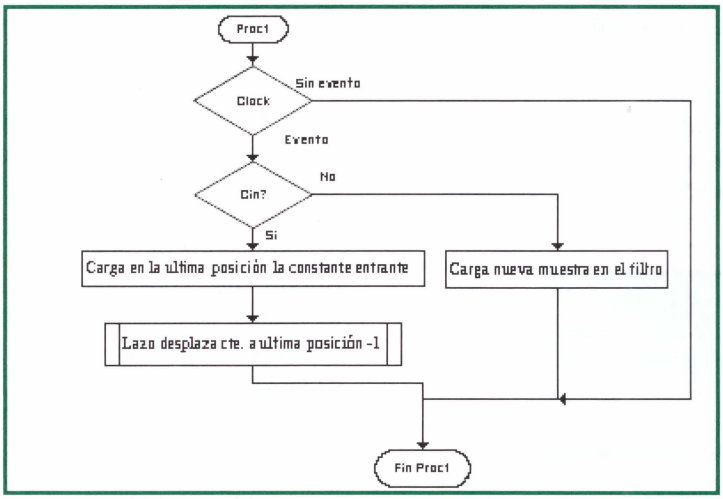

Fig. 3. Diagrama Básico del Proceso de Carga

2.2 Proceso de sumatoria de productos

Este proceso es también controlado por el reloj. Por cada evento del reloj (Fig. 4) se procesa la muestra que ingresó realizando toda la sumatoria de productos del filtro. Se tiene que tener en cuenta el hecho de que el último producto P3=Cin3*Xin no acarrea ninguna suma previa y solamente ha de quedar registrado (véase al respecto la Fig.2).

2.3 Proceso del multiplicador

Este proceso realiza los productos de los datos de entrada Xin con los coeficientes CinN. Los productos son signados y tienen el doble de ancho de bus que los datos entrantes. Para el caso del FPGA fabricado por Altera el multiplicador se implementó mediante la rutina de la librería provista por el software denominada LPM_MULT. Para el producto Xilinx se realizó una rutina de multiplicación de 8 bits signados.

2.4 Proceso para asignar el resultado del filtro a la salida

El proceso final consiste en cargar el resultado producido por el filtro a la salida, previa multiplicación por 256 a los fines de decodificarlo. La realización se ve simplificada considerando solamente los bits más significativos del resultado.

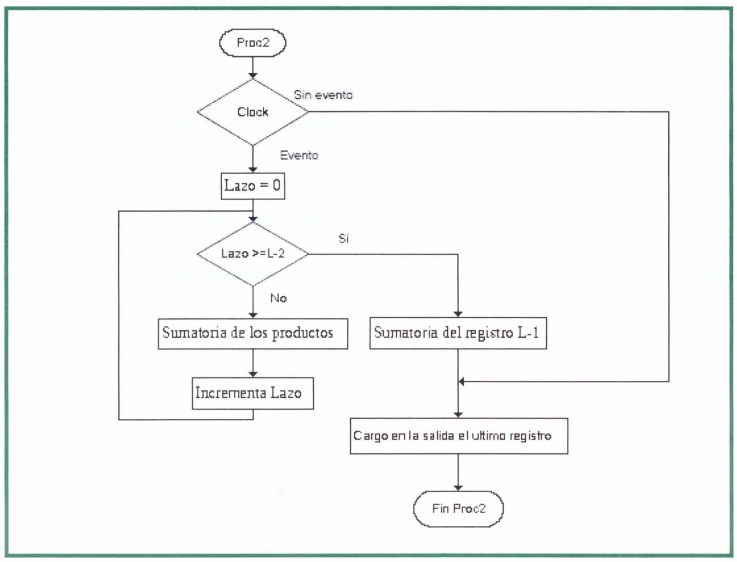

Fig. 4. Diagrama Básico del Proceso Sumatoria de Productos

3. Simulación y resultados

Al objeto de no repetir cada vez las designaciones de las FPGAs simuladas, se hará referencia a las mismas empleando el nombre del fabricante, siendo los modelos los consignados en la Tabla 1.

MARCA	MODELO
ALTERA	EPF10K70RC240-4
XILINX	2s200fg256-5

Tabla 1. FPGAs.

3.1. Requerimientos espaciales

La Tabla 2 muestra la cantidad de celdas lógicas o LCs (logic cells) utilizadas en cada FPGA para implementar el filtro FIR considerado, indicándose el porcentaje de ocupación de la capacidad total resultante.

FPGA	LC	Porcentaje de Ocupación
ALTERA	892	23%
XILINX	393	16%

Tabla 2. LCs utilizadas por el filtro.

3.2. Pruebas del filtro

Se realizó la simulación para obtener la respuesta ante un escalón de 0 a 100 para Xin; como resultado se obtuvo 0 durante los primeros 3 ciclos de reloj y luego el filtro entregó la secuencia {48, 132, 154, 141} y a partir del 7º ciclo de reloj se mantuvo en este último valor. Gráficamente:

Se repitió la simulación pero para una entrada impulsiva, arrojando como resultado la secuencia siguiente {0, 0, 0, 48, 83, 22, -13, 0, 0, 0,...}. La gráfica de la Fig. 6 muestra estos resultados notándose una buena coincidencia con la wavelet discreta Daubechies DB2

Asimismo se ensayo el filtro con otros valores de constantes calculados para una frecuencia de muestreo (*Fs*) de 20Mhz, con una frecuencia de corte (*Fc*) de 1Mhz y ventana rectangular. La respuesta del filtro simulado se mantuvo, en condiciones de régimen, dentro de un 5% de los valores exactos calculados numéricamente, siendo la diferencia debida al redondeo en la implementación de las constantes del filtro.

4. Implementación de un filtro fir con coeficientes fijos

En la realización de este tipo de filtros lo más complicado lo constituye la implementación de las constantes del filtro para la rutina de producto-suma, ya que los coeficientes han de desdoblarse en productos y divisiones en base 2.

Sean por ejemplo los coeficientes de un filtro FIR simétrico $f[k] = \{-1.0, 3.75, 3.75, -1.0\}$, cuya función de transferencia discreta es

$$F(z) = -1 + 3.75z^{-1} + 3.75z^{-2} - z^{-3}$$

Siendo el filtro de orden 4, se implementó un arreglo de 4 elementos para almacenar los valores de las muestras (datos de entrada).

Para la implementación de las constantes –1 puede utilizarse directamente una operación de resta. Para obtener 3.75*Xin se pueden formar

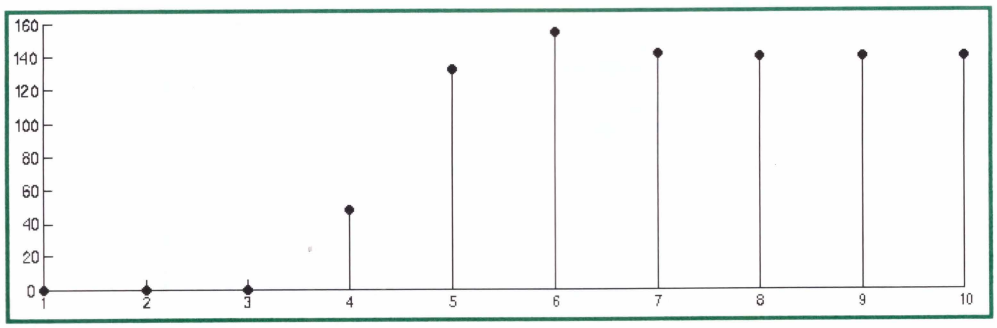

Fig. 5. Respuesta del filtro a un escalón

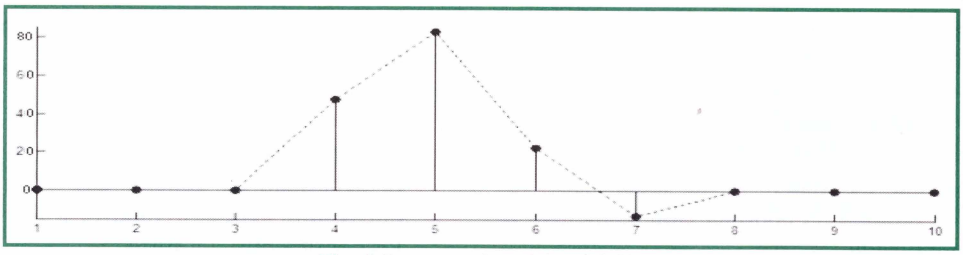

Fig. 6. Respuesta impulsiva del filtro

las operaciones en potencias de 2: (2*Xin+Xin)+(Xin/2)+(Xin/4) = [3 + 0.5 + +.025]*Xin = 3.75*Xin.

Por lo que la ecuación queda, empleando notación de acumuladores:

$$F(A) = 2*A(1) + A(1) + \frac{A(1)}{2} + \frac{A(1)}{4} + 2*A(2) + \quad (7)$$
$$+ A(2) + \frac{A(2)}{2} + \frac{A(2)}{4} - A(3) - A(0).$$

Recordando que para ingresar una nueva muestra, es necesario desplazar los datos del array.

Otro enfoque que puede ser empleado para codificar los coeficientes, es formar para 3,75 = $2^{2} - 2^{-2}$.

Por la simetría del filtro puede aumentarse la velocidad de multiplicación empleando técnicas de pipeline, como asimismo se puede conformar un árbol de sumadores de los multiplicadores, lo que aumentaría aún más el rendimiento.

EJEMPLO 1:

T1 <= A(1) + A(2);	Aplicación de la simetría
T2 <= A(0) + A(3);	
Y <= 4 *T1 – T1/4 – T2;	Uso de la codificación

EJEMPLO 2:

T1 <= A(1) + A(2);	Aplicación de la simetría
T2 <= A(0) + A(3);	
T3 <= 4 *T1 – T1/4;	Uso de multiplicadores Pipeline
T4 <= -T2;	Construcción de un Árbol binario
Y <= T3 + T4;	

De acuerdo a las consideraciones anteriores se procedió a realizar las implementaciones del filtro propuesto en los dos modelos de FPGA, obteniéndose los resultados que se sintetizan en las tablas 3 y 4.

FABRICANTE	ALTERA					
CSD	No	No	Si	No	Si	Si
Árbol	No	No	No	Si	No	Si
Velocidad [Mhz]	18	38	38	100	57	104
LCs	104	80	70	120	64	72
Simetría	No	Si	No	No	Si	Si

Tabla 3. Comparativa de Pruebas realizadas para Altera

Fabricante	XILINX					
CSD	No	No	Si	No	Si	Si
Árbol	No	No	No	Si	No	Si
Velocidad[Mhz]	38	73	56.3	78	103	113
LCs	81	54	43	85	40	51
Simetría	No	Si	No	No	Si	Si

Tabla 4. Comparativa de Pruebas realizadas para Xilinx

De las tablas se ve que el dispositivo de Xilinx ocupa un área menor en todas las combinaciones y en algunos casos puede llegar a superar en más del doble la velocidad de trabajo del de la firma Altera.

5. Implementación de filtro fir usando aritmética distribuida (DA).

La distribución aritmética se diferencia de la suma de productos en que siempre se calcula la suma de productos de un bit específico sobre todos los coeficientes en un solo paso. Esto se realiza mediante una pequeña tabla y un acumulador con desplazamiento. La ventaja de este método esta dado para la realización de Filtros FIR de bajo orden (L <= 4), en que la tabla

puede ser ajustada a un LUT[2].

Además como los FIR son lineales se pueden sumar la salida de varios filtros de bajo orden para generar uno de orden superior.

Se implementó el filtro con DA utilizando una pequeña máquina de estado y la tabla para los coeficientes de orden 4. A continuación se incluye la codificación empleada.

```
PROCESS
   BEGIN
      WAIT UNTIL clk = '1';
      FOR k IN 0 TO 1 LOOP -- Desplazar los cuatro bits
         x0(k) <= x0(k+1);
         x1(k) <= x1(k+1);
         x2(k) <= x2(k+1);
         x3(k) <= x3(k+1);
      END LOOP;
      x0(2) <= x_in(0); -- Cargar x_in en el
      x1(2) <= x_in(1); -- MSBs de register 2
      x2(2) <= x_in(2);
      x3(2) <= x_in(3);
   t0 <= y0; t1 <= y1; t2 <= y2; t3 <= y3;
   s0 <= t0 + 2 * t1; s1 <= t2 - 2 * t3;
   y <= s0 + 4 * s1;
   END PROCESS;
   LC_Table0: case3s
            PORT MAP(table_in => x0, table_out => y0);
    LC_Table1: case3s
            PORT MAP(table_in => x1, table_out => y1);
    LC_Table2: case3s
            PORT MAP(table_in => x2, table_out => y2);
    LC_Table3: case3s
            PORT MAP(table_in => x3, table_out => y3);.
```

En la Tabla 5 se muestran los resultados comparativos obtenidos en Altera y Xilinx utilizando en la realización las técnicas de Distribución Aritmética y Suma de Productos.

	Distribución Aritmética		Suma de Productos	
Fabricante	Altera	Xilinx	Altera	Xilinx
Velocidad[Mhz]	116	181	104	113
LCs	49	25	72	51

Tabla 5. Comparativa de técnicas DA y SDP

Analizando los resultados, se puede dictaminar que la solución con DA es mucho mejor, tanto en velocidad como en economía en el uso de celdas lógicas. El inconveniente que presenta este tipo de implementación es la necesidad de generar una tabla toda vez que resulte necesario modificar los valores de los coeficientes del filtro.

6. Implementación de filtro fir con coeficientes fijos usando Matlab®

Esta prueba consistió en calcular el filtro con la herramienta FDATool de Matlab®, generándose el código VHDL y el TestBench para punto fijo con la misma herramienta. La implementación en Altera y Xilinx resultó decepcionante ya que después de compilado el código ocupa mayor número de LCs y resultó más lento en su ejecución que el filtro FIR Programable descrito en el punto 2. La simulación de funcionamiento se llevó a cabo con la herramienta ModelSim también provista por Matlab®. De la experiencia rea-

[2] LUT (Look-Up Table) definición: En un FPGA es conjunto de circuitos combinacionales que permiten almacenar tablas de búsqueda. También se denominan generadores de funciones.

lizada merece ser destacada la completez de las pruebas incluidas en el TestBench generado por FDA-Tool.

7. Conclusiones y futuros desarrollos.

De todos los métodos descriptos para la implementación de filtros FIR en FPGA, los mejores resultados se obtuvieron con la técnica de Distribución Aritmética. Después de un análisis de la metodología para la carga de los coeficientes con esta técnica se concluyo que no tendría un buen resultado debido a que los filtros deben calibrase para su funcionamiento en el banco de filtros de la wavelet. Lo que implica modificar y recalcular las constantes de la tabla de coeficientes de cada filtro. Una técnica cuya experimentación y análisis queda pendiente, consiste en implementar en el FPGA un pequeño procesador programable en C, determinando asimismo una metodología para evaluar su rendimiento frente al VHDL. Esta técnica representaría una considerable simplificación en la implementación y modificación de los coeficientes del filtro.

8. Glosario

FPGA: Redes de puertas lógicas programables
VHDL: Lenguaje de descripción y modelado de hardware
VHDL'93: Indica el uso del estándar IEEE Std 1076 – 1993
LUT: Lookup table
LC: Celdas lógicas

9. Agradecimientos.

Se agradece especialmente el asesoramiento y motivaciones aportadas por los Ings. Rodolfo A. Cavallero y Walter J. D. Cova del Instituto Universitario Aeronáutico (IUA) y la Universidad Tecnológica Nacional Fac. Regional Córdoba (CUDAR).

10. REFERENCIAS

Proakis, John G.; Manolakis, Dimitris: *Tratamiento Digital de Señales*.
Oppenheim, Alan V.; Schafer, Ronald W.: *Discrete-Time Signal Processing*.
Ashenden, Peter J.: *The Designer Guide to VHDL*.
Xilinx Incorporated. *The Programmable Logic Data Book. 1996*.
ALTERA *Data Book. 1998*