IP core FFT configurable en Runtime

Conference Paper · March 2016

CITATION
1

READS
112

3 authors:

Rodrigo Alejandro Melo
Instituto Nacional de Tecnologia Industrial
34 PUBLICATIONS **15** CITATIONS
SEE PROFILE

Francisco Salomón
Instituto Nacional de Tecnologia Industrial
9 PUBLICATIONS **48** CITATIONS
SEE PROFILE

Bruno Valinoti
Instituto Nacional de Tecnologia Industrial
20 PUBLICATIONS **5** CITATIONS
SEE PROFILE

IP core FFT configurable en Runtime

Rodrigo A. Melo, Francisco Salomón, Bruno Valinoti
Instituto Nacional de Tecnología Industrial
Centro de Micro y Nanoelectrónica
Email: {rmelo, fsalomon, valinoti}@inti.gob.ar

***Resumen*—Debido al requerimiento de una FFT para resolver OFDM sobre FPGAs de Xilinx y Altera, y dado que dicho algoritmo es parte fundamental del procesamiento de señales, se afrontó su desarrollo. Se exploraron varias arquitecturas conocidas basadas en *butterflies Radix-2* con decimación en frecuencia. Se obtuvo una implementación en VHDL que soporta configuraciones en tiempo de ejecución, con relación señal a ruido que ronda los 80 dB para el caso contemplado, pero que no se limita al mismo siendo útil para otras aplicaciones. El desarrollo se validó sobre FPGAs de Xilinx y Altera, de manera aislada en pruebas de laboratorio, y como parte de un demodulador de TV digital, con resultados satisfactorios.**

I. Introducción

El Centro INTI - Electrónica e Informática, en su Laboratorio de Radiocomunicaciones, desarrolló un IP *core* modulador de *Integrated Services Digital Broadcasting Terrestial* (ISDB-T) [1]. El mismo fue implementado sobre una FPGA de la empresa Xilinx, utilizando entre otros el *core Fast Fourier Transform* (FFT) propietario [2] para implementar *Orthogonal Frequency Division Multiplexing* (OFDM).

Por requerimiento de un cliente, el modulador debió ser portado a una FPGA de la empresa Altera. Si bien dicha empresa, al igual que otras, provee *cores* para cálculo de FFT, estos son brindados en modo caja negra y no permiten ajustar o personalizar a bajo nivel, además de solo funcionar en determinados dispositivos, presentar incompatibilidad a nivel interfaz y forma de uso con *cores* de otros fabricantes, e incluso entre diferentes versiones del mismo bloque, y poseen licencias que limitan sus aplicaciones. Se pueden encontrar implementaciones como [3], pero suelen presentar licencias no comerciales y poca flexibilidad en tiempo de ejecución (*runtime*).

En este trabajo se presenta el desarrollo de un *core* FFT, independiente de fabricantes, con múltiples configuraciones en tiempo de síntesis mediante *generics* y en *runtime* mediante señales, permitiendo obtener el balance deseado entre lógica y uso de memoria, entre precisión y recursos consumidos. Esta última es una característica que destacamos, dado que es brindada por algunos fabricantes de FPGA pero no se encontraron *cores* comerciales que lo provean. Los resultados son preliminares, dado que hay aspectos a mejorar y optimizar, pero ha sido utilizado de manera satisfactoria como remplazo en el modulador citado, aunque excede las necesidades del mismo pensando en versatilidad y posibles usos futuros.

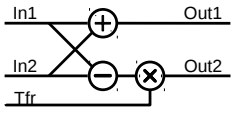


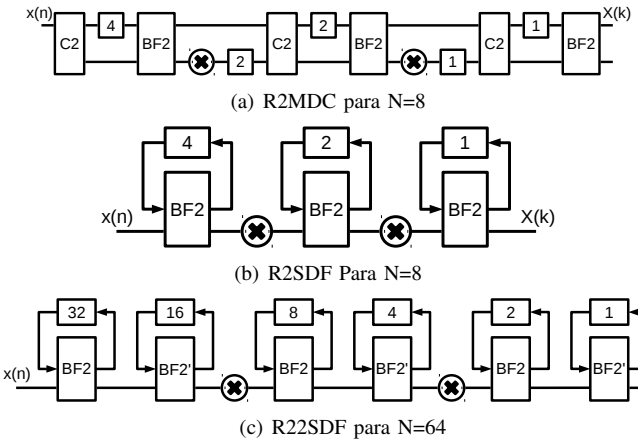
Figura 1. Butterfly Radix-2 para DIF

II. Transformada Rápida de Fourier

FFT es el nombre genérico de algoritmos eficientes para el cálculo de la *Discrete Fourier Transform* (DFT) y su inversa. Una de las versiones más difundida, y la utilizada en este trabajo, es la de Cooley-Tukey [4], la cuál a partir de aprovechar características de simetría y periodicidad sobre los llamados *twiddle factors*, descompone de manera recursiva la transformada en otras más pequeñas, hasta llegar al computo mínimo al cuál denomina *butterfly*. Básicamente, el mismo puede involucrar dos o cuatro datos de entrada complejos recibiendo respectivamente los nombres *Radix-2* y *Radix-4*, siendo el segundo más eficiente en cuanto a operaciones realizadas, pero solo es válido para largos de transformada potencia de cuatro. Aunque menos comunes, existen también implementaciones de orden superior [5] [6] e híbridas [7] [8]. La estrategia de descomposición puede ser en el dominio temporal o en el de la frecuencia denominándose respectivamente algoritmos tipo *Decimation In Time* (DIT) y *Decimation In Frequency* (DIF) [9]. En este trabajo utilizamos *Radix-2* por simplicidad y versatilidad, dado que sólo tiene la condición de aplicarse a cantidad de muestras potencia de dos, y DIF por parecer la forma más popular de resolverlo, aunque la complejidad computacional sea idéntica en ambos casos. En la Fig. 1 se puede observar un esquema genérico del *butterfly* seleccionado, donde las entradas, las salidas y los *twiddle factors* son números complejos.

Siendo N el número de muestras o largo de la transformada el computo de la DFT involucra N^2 operaciones complejas. Utilizando el método descripto logra disminuirse a un valor proporcional a $N * log_2 N$.

Para lograr un flujo de datos de salida continuos, hace falta pasar a arquitecturas del tipo *pipeline*, las cuales han sido objeto de múltiples estudios desde principios de los setenta. Logran disminución en multiplicadores y sumadores, a expensas de utilizar más memoria. En [8] se presenta un buen resumen y comparación de las más difundidas y en la Fig. 2 se muestran esquemas de las basadas en el uso de *butterfly Radix-2* que exploramos. Cabe destacar que los multiplicadores se representan fuera del *butterfly* para explicitar su uso y que, en



(a) R2MDC para N=8

(b) R2SDF Para N=8

(c) R22SDF para N=64

Figura 2. Arquitecturas Pipeline con Radix-2

el caso de R22SDF, el bloque BF2' es una variante donde sólo se multiplica por $-j$.

- *Radix-2 Multi-path Delay Commutator* (R2MDC, Fig. 2(a)). La entrada tiene dos caminos paralelos que se van conmutando y retardando. Reduce a la mitad la cantidad de *butterflies* utilizados.
- *Radix-2 Single-path Delay Feedback* (R2SDF, Fig. 2(b)). Similar al anterior pero utiliza un único camino de datos y realimentación usando registros. Misma cantidad de *butterflies* que R2MDC pero reduce la cantidad de registros.
- *Radix-*2^2 *Single-path Delay Feedback* (R22SDF, Fig. 2(c)). Realiza nueva decimación y utiliza dos *butterflies* diferentes, para lograr eficiencias de *Radix-4* utilizando estructura de *Radix-2*. Respecto a R2SDF disminuye la cantidad de multiplicadores.

III. Implementación

Los requerimientos básicos para OFDM que se tuvieron en cuenta, los cuales demandan configuración en *runtime*, pueden resumirse en:

- Largo de transformada variable. En el caso de ISDB-T se deben poder procesar longitudes de 2K, 4K y 8K muestras.
- Transformada inversa para sistemas de modulación y directa para demodulación.
- Salida de datos continua, con inserción de prefijo cíclico entre símbolos, para evitar interferencia entre los mismos.

Teniendo en cuenta dichos requerimientos, y considerando usos futuros, el *core* desarrollado soporta:

- Modos por ráfaga (*Burst*) y continuo (*Streaming*).
- Largo de transformada desde 8 a 64K muestras complejas (valores potencia de 2).
- Transformada directa e inversa.
- Datos de entrada y *twiddle factors* de 8 a 32 *bits*.
- Aritmética de punto fijo, modo escalado y sin escalar, con redondeo convergente o por truncamiento.
- Salida de datos en modo Natural o *Bit-reversed*.
- Agregado de prefijo cíclico.
- Escalado por etapas.
- Configuración en *runtime*, con habilitación mediante *generic*, de cantidad de muestras, transformada directa o inversa, cantidad de datos del prefijo cíclico y modo de escalado.
- Selección de cantidad de etapas en modo continuo que usaran bloques de memoria de la FPGA. El resto de etapas utilizan memoria distribuida.

La diferencia principal entre los modos por ráfagas y continuo, es que el primero reutiliza memoria para los cálculos intermedios mientras que en el segundo cada etapa posee su propia memoria, insumiendo así grandes cantidades de la misma. La primer versión continua se basó en la arquitectura R2MDC, que fue mejorada en todos los aspectos al pasar en una segunda versión a R2SDF. Se realizó una tercera versión con arquitectura R22SDF, cuya única mejora fue disminuir notablemente la cantidad de multiplicadores utilizados, desmejorando otros aspectos. Actualmente está resuelto con R2SDF y posee soporte experimental de R22SDF.

Los *twiddle factors* se toman de una memoria. La misma se genera a partir de un *script* desarrollado, que los calcula en función del ancho de *bits* seleccionado y la arquitectura utilizada.

En el caso no escalado, la cantidad de *bits* de los datos de salida se incrementa en uno por cada etapa de *pipeline* más un *bit* adicional considerado por redondeos, mientras que en el modo escalado son iguales, pudiendo configurarse para evitar *overflow* que la salida de cada *butterfly* sea dividida por 1, 2, 4 u 8. Por otra parte, luego de la multiplicación por los *twiddle factors*, los datos se ajustan aplicando truncamiento o redondeo convergente.

Al utilizar un algoritmo tipo DIF, la entrada de datos se da en orden Natural y la salida *Bit-reversed* (cabe señalar que en DIT se da la situación opuesta). Para obtener salida en modo natural se utiliza un *Ping-Pong buffer*: dos memorias iguales, donde mientras una se llena en el orden correcto la otra se lee como salida de datos.

La inserción de prefijo cíclico consiste en anteponer a la salida del símbolo, una cantidad de datos configurable correspondientes al final del mismo. Se implementó como un *offset* en el direccionamiento que inicia en valor acorde a lo configurado, llega hasta el final del *buffer* y se vuelve a iniciar, esta vez en cero para barrerlo entero.

Para la selección de cantidad de bloques de memoria en *hardware* y distribuida se utilizaron atributos VHDL propios de cada herramienta de síntesis.

A las variaciones de índices necesarias para implementar las distintas arquitecturas de *pipeline* y tamaños variables de datos y *twiddle factors*, se suman las utilizadas por configuraciones en *runtime* para poder insertar el flujo de datos de entrada en distintas etapas del procesamiento. Además, junto al escalado, provocan la aparición de múltiples modificaciones de tamaño (*resize*) de señales, esparcidos por el código. Estos fueron los puntos más complejos, propensos a errores, que probablemente en algunos casos se puedan optimizar en mayor medida.

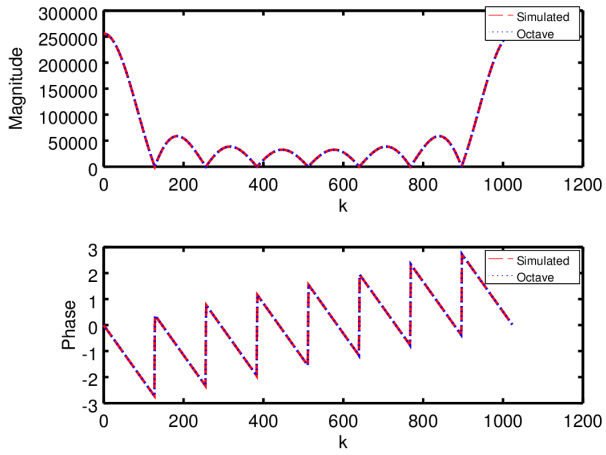


Figura 3. Salida del core simulado vs Octave

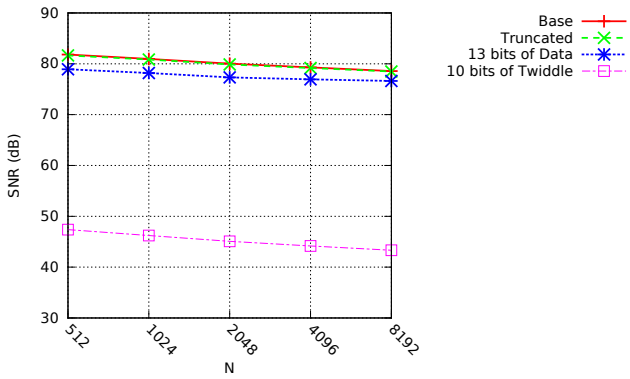


Figura 4. Resultados de simulación

Se utilizó VHDL 93 estándar sin instanciación de primitivas. Se desarrollo sobre FPGA de Xilinx y luego se portó a Altera. Con este fin, los únicos cambios necesarios fueron reescribir un *resize* que la herramienta de Altera no soportaba y agregar atributos VHDL a la hora de especificar cantidad de memoria en bloques y distribuida.

IV. Testeo y validación

Para evaluar mediante simulación las transformadas directa e inversa del *core* desarrollado, se utilizó GHDL [10] y se optó por realizar un modelo de referencia con Octave [11], donde se ingresan símbolos OFDM reales que abarquen todo el rango dinámico, y se calcula la Relación Señal a Ruido (SNR), como la estimación de potencia de la señal transformada calculada en precisión doble con Octave sobre la estimación de potencia del ruido introducido por el *core*. En cada caso se estableció un cierto umbral aceptable de SNR, el cual para los parámetros de uso en el modulador de TV digital debió ser mayor a 40 dB, estableciendo también automatismos para su chequeo. En la Fig. 3 puede verse como ejemplo la transformada de una señal escalón con 1024 muestras, modo no escalado y redondeo convergente, donde el resultado del *core* presenta un SNR mayor a 80 dB.

Se realizó también validación en *hardware* sobre una Spartan 6 de Xilinx y una Cyclone 4 de Altera, haciendo uso de los *kits Avnet S6 micro* y *Terasic DE0-Nano*. Básicamente, se instanció un *core* RS232 junto al *core* FFT, mediante un *script* Python se le transfirieron datos a transformar y se tomaron las salidas en una PC, y de manera similar a la simulación se corroboró el ruido introducido utilizando varios indicadores.

Se incorporó al modulador de TV digital como reemplazo del *core* de Xilinx sobre una Virtex 6, con resultados satisfactorios desde las primeras versiones que utilizaban arquitectura *pipeline* R2MDC, funcionando actualmente con la arquitectura R2SDF.

V. Resultados

En la Fig. 4 se pueden apreciar valores de SNR para transformada directa obtenidos mediante simulación para señales extraídas de un sistema modulador. Se utilizó modo continuo con una configuración Base sin prefijo cíclico ni *runtime*, datos de entrada y *twiddle factors* de 16 *bits*, y redondeo convergente sin escalar, donde la cantidad de muestras se variaron entre los valores 512, 1K, 2K, 4K y 8K. Luego, se fueron variando de a una las configuraciones, pasando por ajustar mediante truncado y modificando cantidad de *bits* a 13 para datos y 10 para *twiddle factors*.

Cuadro I
Resultados de síntesis

(a) Modo Ráfaga

Variante	FF	LUT	BRAM	MUL	MHz
Base	111	633	20	8	99
Truncated	111	543	20	8	110
Runtime	93	567	20	8	99
Scaled	111	460	12	4	102
N=1K	87	549	4	8	99

(b) Modo Continuo (R2SDF)

Variante	FF	LUT	BRAM	MUL	MHz
Base	1334	4133	62	48	61
Truncated	1334	3391	62	48	68
Runtime	1317	3817	62	48	61
Scaled	934	3123	44	44	47
N=1K	683	2101	28	32	61

(c) Otros

	FF	LUT	BRAM	MUL	MHz
R22SDF	1680	5700	60	28	11

Puede observarse que en general se está por encima de los 40 dB de SNR, rondando los 80 dB, lo cuál es aceptable. Se visualiza también como a mayor cantidad de muestras disminuye, debido a más operaciones agregando ruido.

En el Cuadro I se pueden apreciar resultados de síntesis para una Spartan 6 con el *software* ISE Web Pack 14.7, representados gráficamente en la Fig. 5 para el modo por ráfagas y en la Fig. 6 para el modo continuo R2SDF. El modo R22SDF se muestra en el Cuadro sólo en su variante Base por ser experimental, para indicar diferencias con R2SDF. Se estableció un caso Base de 8K muestras, 16 *bits* de datos y *twiddle factors*, sin escalar y redondeo convergente, con salida en orden natural, *runtime* habilitado, inserción de

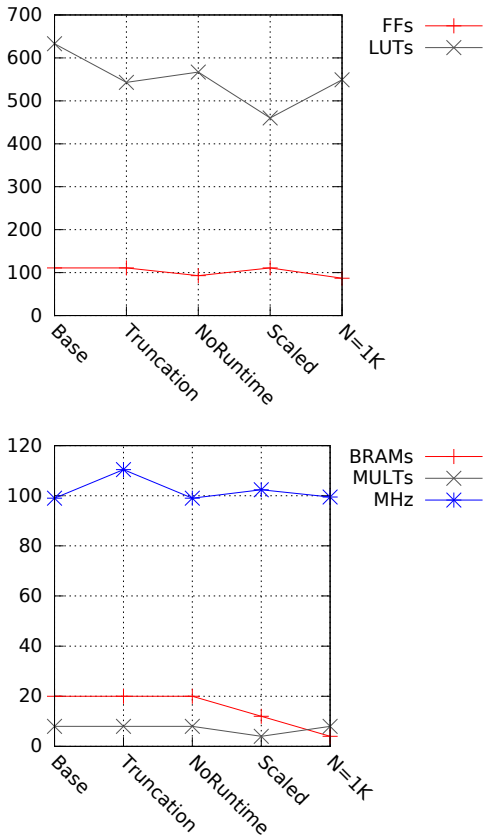


Figura 5. Resultados de síntesis modo Ráfagas

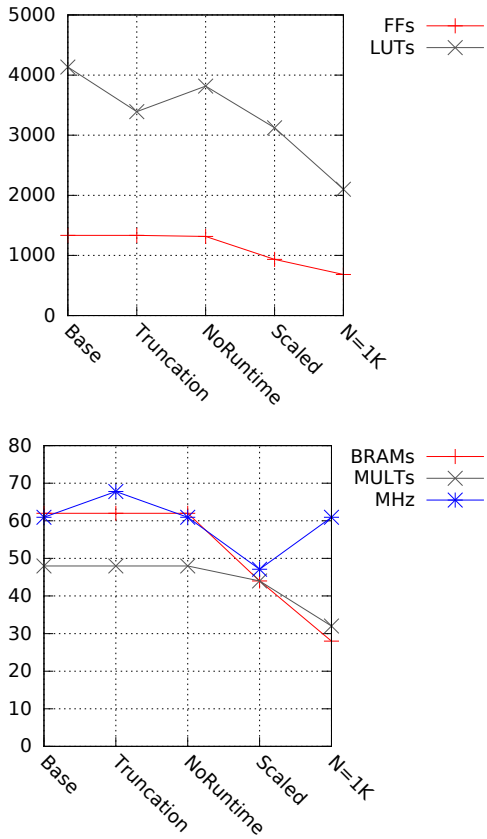


Figura 6. Resultados de síntesis modo Continuo

prefijo cíclico y utilización de bloques de memoria para las seis primeras etapas. Luego se consideraron casos donde se modifican de a una, características como datos truncados, deshabilitar *runtime*, escalar y pasar a 1K muestras.

Puede apreciarse cómo el modo continuo consume más recursos que por ráfaga, siendo este último donde se consiguieron las mayores frecuencias máximas de operación. En cuanto a FFs y LUTs se observa lo esperable, es decir, disminución de recursos insumidos respeto a la versión Base, dado que todas las variantes tienden a reducir el *hardware*. Escalar es una buena forma de disminución de recursos, a expensas de desmejorar la SNR como se observó previamente. En el caso R22SDF disminuye drásticamente la cantidad de multiplicadores, pero empeora el resto de aspectos en contraste con R2SDF. La frecuencia máxima de operación da excesivamente baja en esta tercer arquitectura, pero se identificaron posibles fuentes del problema y serán corregidos en futuras versiones.

VI. Conclusiones

Se obtuvo una implementación de un *core* FFT versátil, configurable en *runtime*, con un SNR aceptable. Se utilizó satisfactoriamente dentro de un sistema modulador de TV digital, pero tiene características que lo hacen apto otros usos.

El diseño se validó en FPGAs de Xilinx y Altera, pero es fácilmente portable a otros fabricantes.

Referencias

[1] INTI. (2014, Nov.) Modulador para televisión digital de industria nacional. [Online]. Available: http://www.inti.gob.ar/noticias/-innovacionDesarrollo/moduladorTVdigital.htm

[2] Xilinx, *LogiCORE IP Fast Fourier Transform v7.1*, Mar. 2011.

[3] Spiral. Dft/fft ip core generator. [Online]. Available: http://www.spiral.net/hardware/dftgen.html

[4] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, Apr. 1965.

[5] M. Sun, L. Tian, and D. Dai, "Radix-8 fft processor design based on fpga," in *Image and Signal Processing (CISP), 2012 5th International Congress on*, Oct. 2012, pp. 1453–1457.

[6] S. Sahoo, A. Ashati, R. Sahoo, and C. Shekhar, "A high-speed radix-64 parallel multiplier using a novel hardware implementation approach for partial product generation based on redundant binary arithmetic," in *Emerging Trends in Engineering and Technology, 2008. ICETET '08. First International Conference on*, Jul. 2008, pp. 474–479.

[7] R. Yavne, "An economical method for calculating the discrete fourier transform," in *AFIPS Fall Joint Computer Conf. 33*, 1968, pp. 115–125.

[8] S. He and M. Torkelson, "A New Approach to Pipeline FFT Processor," in *Proc. IEEE Parallel Processing Symposium*, 1996, pp. 766–770.

[9] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-time Signal Processing*, 2nd ed. Prentice-Hall, 1999, ch. 9.

[10] T. Gingold. Where VHDL meets gcc. [Online]. Available: http://ghdl.free.fr/

[11] J. W. Eaton, D. Bateman, S. Hauberg, and R. Wehbring, *GNU Octave version 3.8.1 manual: a high-level interactive language for numerical computations*, 2014. [Online]. Available: http://www.gnu.org/software/-octave/doc/interpreter