SDSoC como herramienta de codiseño HW/SW para Trabajos Fin de Grado en Informática

Eduardo Magdaleno, Manuel Rodríguez, Cristhian García
Departamento de Ingeniería Industrial
Universidad de La Laguna
San Cristóbal de La Laguna, Canarias, España
emagcas, mrvalido@ull.edu.es

Fernando Pérez
Departamento de Ingeniería Informática
Universidad de La Laguna
San Cristóbal de la Laguna, Canarias, España
fdoperez@ull.edu.es

***Abstract*— Este trabajo presenta los resultados obtenidos en un Trabajo Fin de Grado en Informática que tenía como cometido optimizar el algoritmo de cálculo de la NFFT (non-uniform fast Fourier transform) empleando un dispositivo Zynq. Como novedad, la implementación hardware del algoritmo fue realizada empleando la nueva herramienta SDSoC (software-defined system-on-chip), que facilita el empleo de tecnologías basadas en FPGA (field-programmable gate array) a desarrolladores que no disponen de un conocimiento avanzado en ellas, como, por ejemplo, un estudiante de Informática.**

Keywords—Trabajo Fin de Grado; FPGA; Diseño Digital; Zynq; SDSoC; APSoC; codiseño HW/SW

I. INTRODUCCIÓN

Los alumnos del Grado en Informática de la Universidad de La Laguna adquieren conocimientos notables en diversos lenguajes de programación de alto nivel entre los que se incluye C/C++. Por otro lado, han recibido en el primer curso un contacto básico con el lenguaje VHDL, realizando ejercicios sencillos en una FPGA [1].

La nueva herramienta de desarrollo SDSoC podría permitir la realización de diseños digitales complejos basados en tecnologías híbridas FPGA/uC, empleando C/C++, por parte de alumnos sin un conocimiento exhaustivo de esta metodología (como los graduados informáticos), diseñando a alto nivel y soslayando en buena media las fases más complejas. De hecho, el análisis de *profiling* permite determinar qué componentes del diseño son más indicadas para su realización HW (lógica programable) y cuáles en SW (micro).

Para probar la bondad de la herramienta, se propuso un TFG en Informática para la optimización hardware de un algoritmo ya implementado en C, en este caso la Transformada Rápida de Fourier para datos no equiespaciados, NFFT [2], que es un algoritmo que se emplea en astrofísica, procesado de imágenes/vídeo, medicina, análisis sismológico, etcétera [3-5].

Se parte de códigos de la implementación del algoritmo tanto en Matlab como en C, con unos datos de entrada y salida para comprobar la bondad de la implementación. El alumno debe particionar el código C en varias funciones que sean susceptibles de ser pasadas a una implementación hardware, pues esto es un requisito de la herramienta. SDSoC estudia qué particionado HW/SW es el óptimo en relación a recursos disponibles/aceleración. Esto se hace a través de la herramienta de *profiling* de la que dispone el SDSoC [2]. Es preciso que el ingeniero introduzca en el código una serie de directivas que empleará la herramienta para implementar en hardware las partes deseadas del código con mayor o menor eficiencia (paralelizando bucles, por ejemplo).

La implementación del algoritmo se realiza en un dispositivo Zynq de la compañía Xilinx, que es un APSoC (all-programmable system-on-chip), un dispositivo híbrido que combina un ARM con una parte de lógica programable pensada para procesamiento de alto rendimiento explotando técnicas de paralelismo y pipeline en las etapas más complejas de cualquier algoritmo.

Este trabajo presenta las experiencias más relevantes encontradas usando la herramienta SDSoC para diseñar e implementar una versión computacionalmente más eficiente en términos de velocidad de ejecución. El algoritmo se modificó para su implementación en un SoC empleando una codificación en C con pragmas. De esta manera, el alumno no necesitó invertir tiempo en detalles de un nivel de abstracción bajo con VHDL y pudo explorar varias alternativas de arquitectura en un periodo de tiempo relativamente corto.

Este trabajo está estructurado en cinco secciones incluyendo esta Introducción. Primero se introduce brevemente el algoritmo de la NFFT y las modificaciones a realizar en el código C para que éste pueda ser tratado convenientemente por el SDSoC. En el apartado 3 se describe la tecnología empleada para la implementación del algoritmo, así como lo metodología de diseño. En la sección cuarta se muestran los resultados obtenidos y en la quinta las conclusiones.

II. DESCRIPCIÓN DEL ALGORITMO Y ADAPTACIÓN DEL CÓDIGO C

Para tratar de implementar cualquier algoritmo en un dispositivo Zynq es suficiente partir de un código C de dicho algoritmo; en este caso la NFFT. El código debe ser adaptado para que SDSoC pueda manejarlo convenientemente.

A. La NFFT

En esta sección se describe brevemente el algoritmo NFFT que se desea implementar en el dispositivo Zynq, La NFFT calcula aproximaciones de sumas de:

$$f(x_j) = \sum_{k=-N/2}^{\frac{N}{2}-1} \hat{f}_k e^{-2\pi i k x_j}, j = 1,\ldots,M \qquad (1)$$

en nodos arbitrarios $x_j \in [-1/2,1/2)$. Para el cálculo eficiente se recurre a la aproximación de un polinomio trigonométrico que emplea una función ventana φ, un factor de sobremuestreo σ y un parámetro de corte m [2,5,6]. Este cálculo aproximado de $f(x_j)$ involucra los pasos que se describen a continuación:

- Paso 1 (Deconvolución): para los k coeficientes complejos de entrada $\hat{f}_k$,

$$\hat{g}_k = \frac{\hat{f}_k}{c_k(\tilde{\varphi})} \qquad (2)$$

- Paso 2 (Transformada rápida de Fourier, FFT): se calcula para l puntos, empleando los coeficientes obtenidos en el paso anterior,

$$g_l = \frac{1}{n} \sum_{l \in I_n} \hat{g}_k e^{-2\pi i k l / n}, n = \sigma N \qquad (3)$$

- Paso 3 (Convolución): para los M puntos donde se desea evaluar la transformada, $j=0,\ldots,M-1$, se calcula con una versión periódica de la función ventana, $\tilde{\psi}$:

$$f_j = \sum_{I_{n,m}(x_j)} g_l \tilde{\psi}(x_j - \frac{l}{n}) \qquad (4)$$

Para mantener pequeño el error de la aproximación, debe seleccionarse una función ventana φ bien conocida en ambos dominios transformados. Varias de estas funciones han sido propuestas en [5,7,8].

B. Adaptación del código C de la NFFT

La herramienta SDSoC requiere que las partes del código que la NFFT o cualquier algoritmo que sea susceptible de implementar en hardware debe estar explícitamente en una función. Así, se modificó el código C de partida para conformar en tres funciones los pasos descritos en el apartado anterior: deconvolución, FFT y convolución (figura 1).

De esta manera, cada módulo fue implementado C como una función independiente. Esto va a permitir que podamos tratar de implementar en hardware las funciones una o una o todas a la vez.

En cada una de las funciones se añaden una serie de *pragmas*, o directivas al SDSoC. Se usan para optimizar la implementación hardware usando pipeline, desenrollos de bucles, particionado de array de datos, etcétera. Los *pragmas* sirven para explorar micro-arquitecturas que satisfagan los objetivos de rendimiento y ocupación en la parte de lógica configurable del chip. Se han usado tres *pragmas* en las funciones de la NFFT.

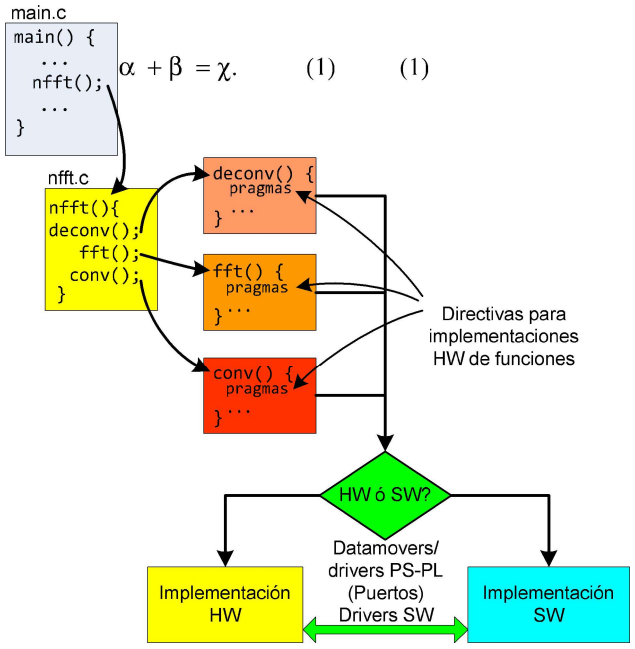

Fig. 1. Estructura del código C adaptado a SDSoC

El *pragma* de desenrollamiento de bucles se ha usado en los bucles de las tres funciones para desenrollar y crear varias operaciones simultáneas en lugar de implementar un único operador. Esto crea varias copias del cuerpo del bucle y ajusta el contador de iteraciones en consonancia. Hay que tener en cuenta que más paralelismo supone mayor productividad (throughput en inglés) y mayor velocidad de cómputo, pero el empleo de más recursos hardware. El *pragma* de pipeline supone otra técnica para explotar el paralelismo en las iteraciones de los bucles. Permite implementar las operaciones del bucle de una manera concurrente como la mostrada en la figura 2. Debe especificarse un intervalo de iniciación, que detalla el número de ciclos de reloj entre los comienzos de iteraciones del bucle consecutivas.

Por último, el *pragma* de acceso a datos se ha usado al comienzo de las tres funciones para permitir la implementación de un acceso streaming de arrays de datos en una transferencia. Con esto se evita la implementación no deseada de memoria compartida, normalmente implementada en una memoria DDR externa de la lógica programable para arrays de datos grandes.

III. Descripción de la tecnología y metodología

En esta sección se describe la plataforma hardware Zynq usada para la implementación del algoritmo. Además, se describe la herramienta de desarrollo SDSoC que se ha empleado en la etapa de diseño.

.

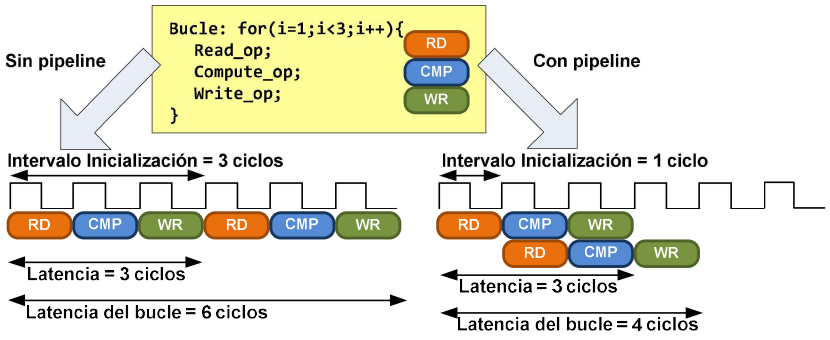

Fig. 2. Ejemplo de hacer pipeline en un bucle. La productividad (throughput) del bucle aumenta.

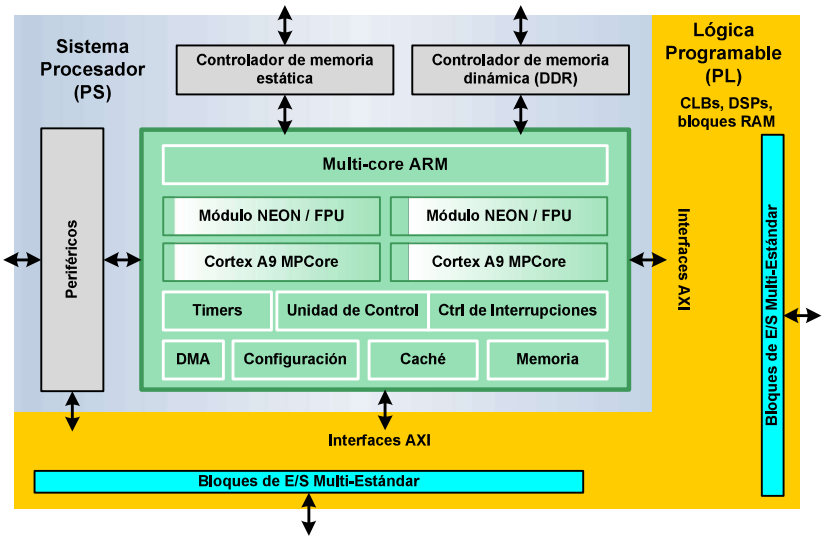

Fig. 3. Arquitectura simplificada del dispositivo Zynq 7020.

A. La plataforma hardware

El prototipo se implementó en una tarjeta de desarrollo Zedboard [9]. Lo más relevante de la misma es el dispositivo Zynq-7020 [10]. Básicamente, Zynq es un dispositivo que combina procesador dual-core ARM Cortex-A9 [11] con una región con los componentes típicos de una FPGA, en lo que Xilinx, el fabricante, ha bautizado como APSoC (All programmable system-on-chip) [12].

La arquitectura general de un dispositivo de estas características contiene dos regiones: un sistema de procesador (Processing System – PS) y la lógica reconfigurable (Programmable Logic – PL). Un esquema de su arquitectura se muestra en la Figura 2. Ambas regiones se pueden usar de manera independiente o conjuntamente.

El procesador es un procesador hardware dual-core ARM Cortex, que es una alternativa hardware a los procesadores firmware tipo PicoBlaze o MicroBlaze que también maneja Xilinx [13, 14]. Estos procesadores se implementan usando los recursos genéricos de la propia FPGA, configurando procesadores muy flexibles. Los procesadores hardware son más rígidos, pero lo compensan un rendimiento considerablemente mayor.

Cada core del ARM tiene un motor de procesamiento NEON y una unidad de punto flotante (FPU) pensada para realizar tareas de cómputo. La Unidad de Control realiza tareas relacionadas con la interfaz entre los procesadores y la memoria caché. También gestiona las transacciones que tienen lugar entre el procesador y la lógica programable. El ARM opera con un reloj de 866 MHz [12].

Por otra parte, la parte FPGA del dispositivo Zynq-7020 se basa en la familia Artix-7 de Xilinx, y se muestra en la figura 3. Esta parte es una red de bloques lógicos configurables (CLBs) compuestos, a su vez, por dos slices. Estos elementos se conectan entre sí a través de interconexiones programables y matrices de conmutación. Dentro de cada slice, 4 memorias del tipo 6-LUT son capaces de implementar funciones de hasta 6 variables, memorias distribuidas o registros de desplazamiento.

Cada slice incluye también 8 flip-flops para implementar circuitos secuenciales. Zynq-7020 tiene 53200 6-LUTs y 106400 flip-flops [12].

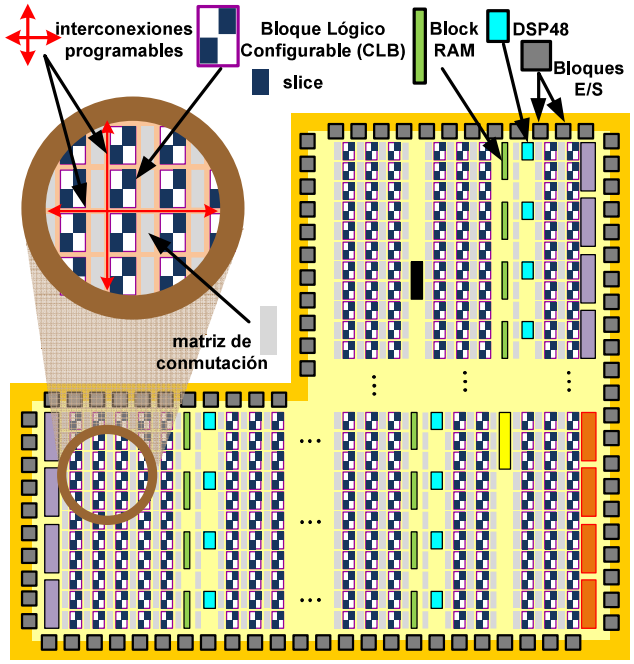

Fig. 4. La región reconfigurable de la Zynq-7020 y sus elementos básicos.

Además de esta lógica genérica, la zona FPGA contiene algunos components específicos, como Block RAMs para requerimientos de mayor almacenaje que la memoria distribuida de los slices y bloques DSP48 para aritmética de alta velocidad. Este componente tiene un pre-sumador/restador, un multiplicador y un post-sumador/restador, resultando un módulo muy versátil. Todos estos elementos se encuentran embebidos en la red de componentes de la FPGA. Zynq-7020 tiene 140 36Kb Block RAMs y 220 módulos DSP48 (18 x 25 bit) [12].

B. Metodología de diseño con SDSoC

Como se ha comentado, el dispositivo Zynq está conformado por una parte de procesador y otra de lógica reconfigurable. Así, este dispositivo híbrido es capaz de integrar las ventajas de ambas tecnologías en un único chip. La parte PS se puede encargar de las rutinas software, GUIs o un sistema operativo que controle tareas y aplicaciones (incluyendo procesado de datos); y la parte PL, de implementar eficientemente algoritmos altamente paralelizables. Estos algoritmos contienen operaciones matemáticas que deben realizarse para un gran número de muestras simultáneamente, en las que las implementaciones software suponen un cuello de botella. El algoritmo NFFT forma parte de este tipo de algoritmos.

La metodología clásica de co-diseño HW/SW empleada hasta ahora, con procesadores firmware tipo MicroBlaze se ilustra en la figura 5 [15]. El ingeniero debía identificar cada uno de los subsistemas del diseño y decidir apropiadamente qué partes implementar en hardware y cuáles en software de forma manual. La comunicación entre las partes del sistema tenía que ser definida. Así, Vivado se empleaba para el desarrollo de los componentes hardware, mientras que la herramienta SDK se encargaba de la programación software. Vivado puede incluir módulos VHDL/verilog, desarrollados con System Generator, también permite descripciones de alto nivel con Vivado HLS y también se puede recurrir a un extenso catálogo IP o IPs comerciales. Por su parte, SDK incluye drivers para los IPs, soporte de librerías para ARM y NEON usando C y C++, y herramientas de depuración y *profiling*.

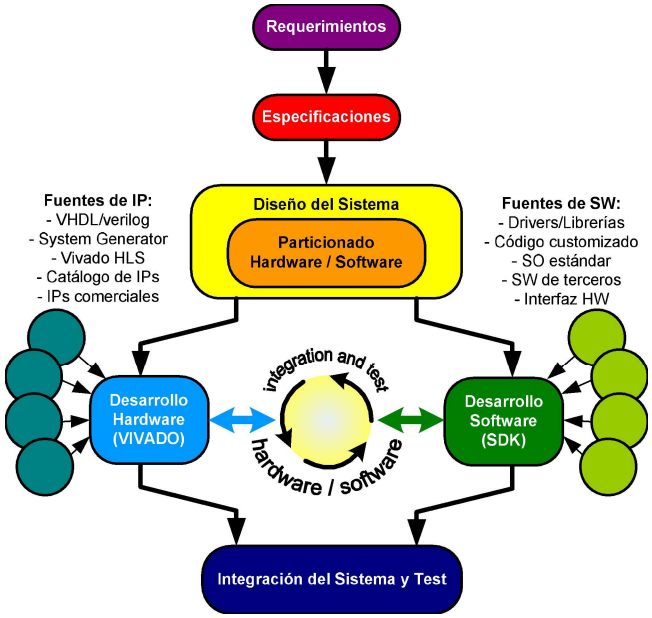

Fig. 5. Flujo de co-diseño hardware/software tradicional.

Seleccionar desde el inicio del diseño un buen particionado entre el hardware y el software es clave y debe realizarse en etapas muy temprana en la fase de diseño Es por ello que el ingeniero, por lo general, tarde o temprano, debe refinar y mejorar este particionado, y no es tarea baladí. Así, estos cambios en el diseño son del todo indeseables y a evitar, porque retrasan significativamente el desarrollo del proyecto.

La reciente herramienta SDSoC automatiza y simplifica el particionado y la interfaz entre el hardware y el software. Para ello únicamente es necesario un algoritmo escrito en C ó C++ con una serie de directivas en las partes que se desea implementar en la región de lógica programable del dispositivo Zynq. Evidentemente, esto permite al ingeniero probar varias alternativas de diseño de una manera rápida y sencilla.

La metodología de diseño con SDSoC se muestra en la figura 6 [16]. Simplemente, el usuario selecciona qué funciones desea implementar en hardware con el único requisito que cada función debe estar codificada en su propio fichero C. La herramienta incluso es capaz de analizar qué funciones son las más adecuadas para su implementación hardware por medio de la herramienta de profiling. Esto constituye lo que se denomina exploración de macro-arquitecturas. A continuación, cada función a hardware es refinada usando las directivas que ya han

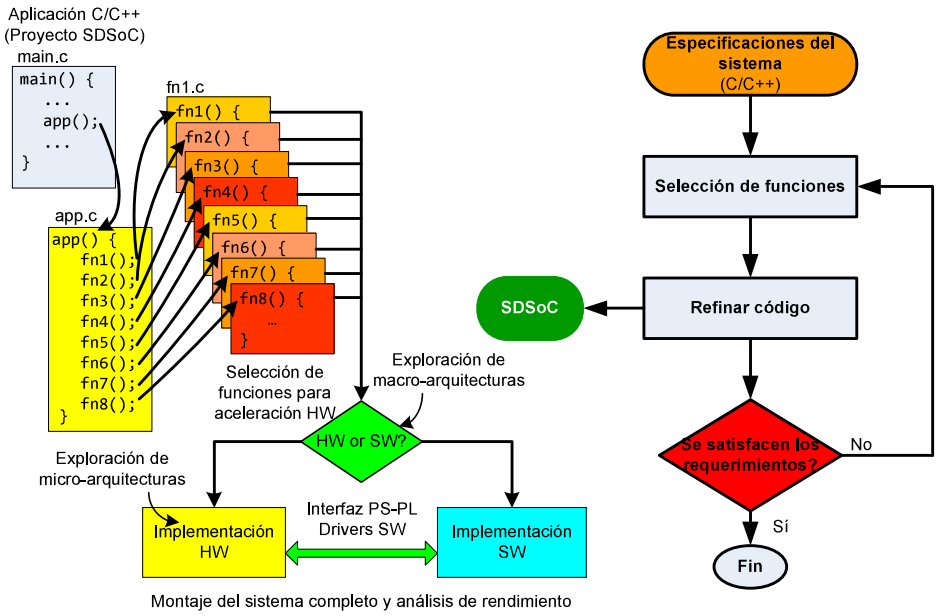

Fig. 6. Esquema de la metología de diseño usando SDSoC.

sido comentadas en el apartado anterior, con el fin de buscar su implementación óptima (análisis de micro-arquitecturas).

IV. Caso de estudio: implementación de la NFFT

En el caso de la implementación de la NFFT tenemos por separado tres funciones susceptibles a ser pasadas a hardware: las etapas de deconvolución, FFT y convolución (figura 7).

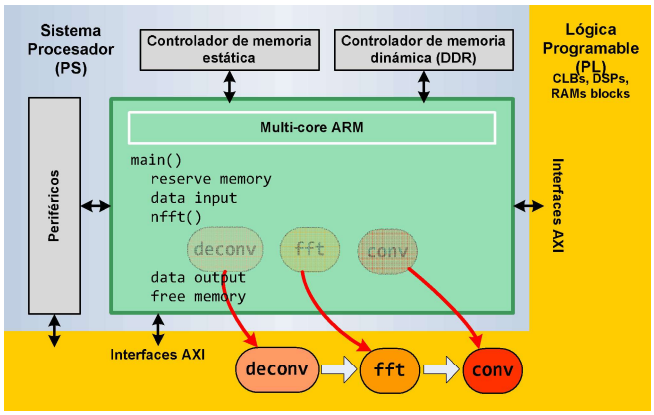

Fig. 7. Aceleración HW/SW del código C de la NFFT con Zynq. Se pueden seleccionar una, dos o las tres funciones para ser implementadas en HW.

Para evaluar la bondad de la herramienta sobre el algoritmo de la NFFT, para cada función, de manera independiente, se ha implementado una solución hardware y otra software. También se ha analizado el algoritmo NFFT completo de la misma manera. La función ventana seleccionada para la NFFT ha sido la función gaussiana φ, el factor de sobremuestreo σ igual a 2 y el parámetro de corte m igual a 6.

La Tabla 1 muestra los resultados de las soluciones SW y HW para el módulo deconvolución. Estos resultados han sido calculados para un número de muestras variable de 32 hasta 1024 en potencia de 2. Se muestra los ciclos de reloj para el cálculo de esta fase, así como el porcentaje de recursos que han sido necesario. Puede apreciarse que la solución hardware mejora la alternativa software hasta un factor 37 para 1024 muestras.

Para la FFT se ha repetido el estudio, obteniéndose una mejora significativa entre ambas soluciones. En este caso, se obtiene una aceleración de hasta 380 para 1024 puntos (Tabla 2). En cambio, los resultados para el módulo de convolución no mejoran apreciablemente, con una aceleración de 1.2 (Tabla 3). En este caso, si se quisiera mejorar la eficiencia, habría que describir la arquitectura a más bajo nivel (empleando VHDL), que no se ha considerado, por no formar parte del objetivo de este trabajo. El hecho de obtener tan pobres resultados en este módulo puede explicarse por el acceso irregular de datos en la convolución de la ecuación (3), no pudiendo usar los mecanismos de SDSoC para acceso a memoria contigua usando las directivas *sds alloc* y *sds free*. Esto causa acceso múltiple a direcciones de memoria no contigua y, consecuentemente, la velocidad del coprocesador hardware se reduce significativamente.

TABLE I. ACELERACIÓN DE LA FUNCIÓN DECONVOLUCIÓN

Número de puntos	Solución Software			Solución Hardware			Aceleración del algoritmo
	Ciclos CPU	*% PS[a]*	*% PL[b]*	*Ciclos CPU*	*% PS[a]*	*% PL[b]*	
32 (2^5)	265,328	24.6%	0.0%	14,562	4.1%	19.7%	18.11
64 (2^6)	278,355	24.9%	0.0%	14,763	4.3%	20.0%	18.85
128 (2^7)	304,407	25.3%	0.0%	14,983	4.5%	20.6%	20.32
256 (2^8)	356,512	26.2%	0.0%	15,423	4.9%	21.7%	23.12
512 (2^9)	460,722	27.8%	0.0%	16,303	5.7%	23.9%	28.26
1024 (2^{10})	669,142	31.1%	0.0%	18,064	7.4%	28.3%	37.04

[a.] Procentaje del Sisteme de Procesador PS empleado (ARM)

[b.] Porcentaje de Lógica Programable PL empleado (FPGA)

TABLE II. ACELERACIÓN DE LA FUNCIÓN FFT

Número de puntos	Solución Software			Solución Hardware			Aceleración del algoritmo
	Ciclos CPU	*% PS[a]*	*% PL[b]*	*Ciclos CPU*	*% PS[a]*	*% PL[b]*	
32 (2^5)	252,298	25.2%	0.0%	7,844	4.1%	23.4%	32.16
64 (2^6)	342,447	25.5%	0.0%	7,849	4.3%	23.8%	43.63
128 (2^7)	522,744	25.9%	0.0%	7,859	4.5%	24.6%	66.52
256 (2^8)	883,339	26.7%	0.0%	7,880	4.9%	26.2%	112.10
512 (2^9)	1,604,528	28.4%	0.0%	7,919	5.7%	29.4%	202.62
1024 (2^{10})	3,046,906	31.9%	0.0%	7,998	7.4%	35.7%	380.96

TABLE III. ACELERACIÓN DE LA FUNCIÓN CONVOLUCIÓN

Número de puntos	Solución Software			Solución Hardware			Aceleración del algoritmo
	Ciclos CPU	*% PS[a]*	*% PL[b]*	*Ciclos CPU*	*% PS[a]*	*% PL[b]*	
32 (2^5)	159,932	61.3%	0.0%	143,336	4.1%	54.5%	1.12
64 (2^6)	348,085	62.3%	0.0%	311,964	4.3%	55.1%	1.12
128 (2^7)	724,390	64.1%	0.0%	649,220	4.5%	55.9%	1.12
256 (2^8)	1,476,390	67.8%	0.0%	1,323,732	4.9%	57.7%	1.12
512 (2^9)	2,982,218	75.2%	0.0%	2,682,756	5.7%	61.2%	1.12
1024 (2^{10})	5,992,656	89.9%	0.0%	5,370,804	7.4%	68.3%	1.12

TABLE IV. ACELERACIÓN DEL SISTEMA ENTERO

Número de puntos	Solución Software			Solución Hardware			Aceleración del algoritmo
	Ciclos CPU	*% PS[a]*	*% PL[b]*	*Ciclos CPU*	*% PS[a]*	*% PL[b]*	
32 (2^5)	678,094	69.9%	0.0%	166,582	4.1%	61.5%	4.07
64 (2^6)	969,428	70.7%	0.0%	335,397	4.3%	62.5%	2.90
128 (2^7)	1,552,094	72.2%	0.0%	673,027	4.5%	64.4%	2.31
256 (2^8)	2,717,426	75.3%	0.0%	1,348,287	4.9%	68.6%	2.02
512 (2^9)	5,048,090	81.4%	0.0%	2,698,807	5.7%	76.9%	1.87
1024 (2^{10})	9,709,418	93.6%	0.0%	5,399,846	7.4%	91.3%	1.80

Por último, la Tabla 4 muestra la mejora lograda cuando los tres módulos fueron seleccionados para su aceleración en hardware. Para todas las longitudes de datos, la solución hardware mejora a software, obteniéndose aceleraciones de 1.80 a 4.07. En este caso particular, la aceleración del peor módulo pasado a hardware (módulo de convolución), se compensa con la aceleración obtenida en los otros dos módulos.

V. CONCLUSIONES

Se ha conseguido que un alumno con unas nociones elementales de VHDL y diseño de circuitos electrónicos con FPGA realice de una manera fácil e intuitiva un análisis de varias alternativas de co-diseño HW-SW en una plataforma híbrida Zynq.

En concreto, se ha realizado una implementación eficiente del algoritmo de la NFFT usando dicha plataforma que incluye una parte de procesador y una parte de lógica programable tipo FPGA.

El uso de la herramienta *profiling* del entorno de desarrollo SDSoC, junto con los *pragmas* insertados en el código C nos permitió detectar los cuellos de botella del algoritmo y mejorar su ejecución.

Computacionalmente, el uso de esta tecnología muestra una mejora importante cuando es posible el empleo de una interfaz de datos en forma *streaming*. Para este caso en particular, es posible obtener aceleraciones de 380 en el cálculo de la FFT. El acceso irregular del último módulo hace que el SDSoC no elija la interfaz óptima y el resultado final se resiente bastante.

No obstante, nuestro principal objetivo en este trabajo consistió en explorar el potencial del empleo de la herramienta SDSoC para implementar algoritmos complejos en hardware sin recurrir a una descripción VHDL, abriendo el camino del uso de esta metodología en futuros Trabajos Fin de Grado.

REFERENCES

[1] E. Magdaleno, B. Rodríguez, M. Rodríguez, "Guía Docente de Sistemas Electrónicos Digitales", Grado en Ingeniería Informática, Escuela Superior de Ingeniería y Tecnología, 2018. Disponible en: https://e-guia.ull.es/etsii/query.php?codigo=139261024 (accedido el 19/02/2018).

[2] D. Potts, G. Steidl, M. Tasche, "Fast Fourier transforms for nonequispaced data: A tutorial", en Modern Sampling Theory: Mathematics and Applications; Benedetto, J., Ferreira, P., Eds.; Birkhäuser: Boston, MA, USA, 2001; pp. 247–270.

[3] T. Knopp, S. Kunis, D. Potts, A note on the iterative MRI reconstruction from nonuniform k-space data. Int. J. Biomed. Imaging 2007, 6, pp. 4089–4091.

[4] A.J.W. Duijndam, M.A. Schonewille, "Nonuniform fast Fourier transform", en Geophysics 1999, 64, pp. 539–551.

[5] H. Schomberg, J. Timmer, "The Gridding method for image reconstruction by Fourier transformation" en IEEE Trans. Med. Imaging 1995, 14, pp. 596–607.

[6] A. Dutt, V. Rokhlin, "Fast Fourier transforms for nonequispaced data" en SIAM J. Sci. Comput. 1993, 146, pp. 1368–1393.

[7] J.A. Fessler, B.P. Sutton, "Nonuniform fast Fourier transforms using min-max interpolation" en IEEE Trans. Signal Process. 2003, 51, pp. 560–574.

[8] S. Kunis, D. Potts, G. Steidl, "Using NFFT 3 – A sofware library for marious nonequispaced fast Fourier transforms en ACM Trans. Math. Softw. 2009, 36, 19.

[9] Avnet. Zedboard (Zynq Evaluation and Development) Hardware User's Guide. Version 2.2. 2014. Available online: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf (accedido el 2 de noviembre 2016).

[10] Xilinx. Zynq-7000 All Programmable SoC. Technical Reference Manual. UG585 (v1.11). 2016. Available online: https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf (accedido el 2 Noviembre 2016).

[11] ARM. Cortex-A9 MPCore Technical Reference Manual. Revision r4p1. 2012. Available online: https://static.docs.arm.com/ddi0407/i/DDI0407.pdf (accedido el 2 Noviembre 2016).

[12] Xilinx. Zynq-7000 All Programmable SoC Overview. DS190 (v1.10). 2016. Available online: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf (accedido el 2 Noviembre 2016).

[13] K. Chapman. PicoBlaze for Spartan-6, Virtex-6, 7-Series, Zynq and UltraScale Devices (KCPSM6). Release 9. 2014. Available online: https://www.xilinx.com/ipcenter/processor_central/picoblaze/member/ (accedido el 2 Noviembre 2016).

[14] Xilinx. LogiCORE IP MicroBlaze Micro Controller System (v1.1). DS865. 2012. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ds865_microblaze_mcs.pdf (accedido el 2 Noviembre 2016).

[15] L.H. Crockett, R.A. Elliot, M.A. Enderwitz, R.W. Stewart, "Designing with Zynq" en The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC, 1st ed.; Strathclyde Academic Media: Scotland, UK, 2014; pp. 47–75.

[16] Xilinx. SDSoC Environment User Guide. UG1027 (v2016.2). 2016. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_2/ug1027-sdsoc-user-guide.pdf accedido el 3 Noviembre 2016).