DISEÑO E IMPLEMENTACIÓN DE UN ACELERADOR POR HARDWARE PARA FILTRO FIR PASO BANDA EN LA TARJETA PYNQ-Z1.

Fecha enviada: 15 octubre 2019 **Fecha revisado: 27 octubre 2019**

Manuel Ríos, MEng[1]

RESUMEN

Esta investigación presenta al lector el proceso de diseño e implementación de un filtro FIR pasa banda por medio del entorno de trabajo PYNQ en la tarjeta PYNQ-Z1. El objetivo principal de este estudio consiste en acelerar el cálculo computacional que conlleva el uso de un filtro FIR pasa banda por medio del uso de hardware dedicado a este proceso. Para ello, se realizó de forma experimental dicha investigación, utilizando un FPGA como dispositivo de destino de hardware. Dentro de los resultados más importantes destaca el efecto de una menor atenuación en la banda de rechazo del filtro al truncar los coeficientes del filtro de punto flotante a valores enteros, lo que significa un compromiso entre precisión y velocidad al momento de realizar esta implementación. Finalmente, al comparar esta implementación en hardware con implementaciones por *software*, se puede comprobar que existe un incremento en el rendimiento de nuestra implementación, llegando a ser hasta 45 veces más eficiente que la implementación por software.

Palabras clave: acelerador, *Blackman*, filtro digital, FIR, FPGA, *hardware*.

ABSTRACT

“Design and implementation of a hardware accelerator for a bandpass FIR filter on the PYNQ-Z1 board.”

This research introduces the reader to the design and implementation process of a bandpass FIR filter through the PYNQ work environment on the PYNQ-Z1 board. The main objective of this study is to accelerate the computational calculation that involves the use of a band-pass FIR filter through the use of hardware dedicated to this process. For this, this research was carried out in an experimental way, using an FPGA as a hardware target device. Among the most important results, the effect of less attenuation in the filter rejection band by truncating the coefficients of the floating-point filter to integer values stands out, which means a compromise between precision and speed when carrying out this implementation. Finally, when comparing this hardware implementation with software implementations, it can be seen that there is an increase in the performance of our implementation, becoming up to 45 times more efficient than the software implementation.

Keywords: accelerator, Blackman, digital filter, FIR, FPGA, hardware.

INTRODUCCIÓN

Los filtros digitales han sido utilizados para dos grandes tareas: separar señales combinadas con otras señales y recuperar señales distorsionadas. Dentro de los filtros digitales se pueden distinguir dos grupos: los filtros de respuesta finita al impulso, FIR, y los filtros de respuesta infinita al impulso, IIR. La implementación de los filtros FIR permiten un menor rizo en la banda de paso, mejor corte y atenuación en la banda de rechazo, así como también pueden configurarse para ser de fase lineal, cuya contraparte analógica no es capaz de ejecutar. La desventaja, al utilizar filtros digitales, radica en que una señal analógica primero debe de ser digitalizada por medio de conversores analógicos - digitales que operan a la frecuencia de Nyquist para un muestreo adecuado de la señal en cuestión.

Dentro de las aplicaciones de los filtros FIR destacan los sistemas de radio definida por *software* (Collins et ál., 2008), la codificación predictiva lineal (Vaidyanathan, 2008), el análisis del habla (Antoniou,

[1] MEng. en Electrónica y Radio Ingeniería. Director del Departamento de Ingeniería en Electrónica y Telecomunicaciones de la Universidad Rafael Landívar, marios@url.edu.gt.

2018), procesamiento de señales de múltiple tasa (Mertins, 2000), análisis de datos (Hyndman et ál., 2018), entre otras muchas aplicaciones.

La implementación de un filtro FIR puede ser realizada por medio de *software* o por *hardware*. Matlab, Octave y SciPy utilizan implementaciones en *software* que se basan en el uso de librerías en C/C++ (Hill, 2016), las cuales buscan acelerar el tiempo de ejecución de los filtros aprovechando distintas técnicas de programación como el desenroscado de bucles (Cooper et. ál., 2013). Por su parte, las implementaciones en *hardware*, además de buscar la reducción en el tiempo de ejecución, también permiten el procesamiento en paralelo y la segmentación del *hardware*, lo que produce un aumento en la tasa de transferencia efectiva (Kastner et ál., 2018). Dentro de los dispositivos que mejor se han posicionado en el procesamiento digital de señales, DSP, se encuentran los arreglos de compuertas lógicas programables en campo o FPGAs, debido a su capacidad de ajuste en aplicaciones que requieren alto rendimiento y paralelismo (Lapedus, 2006, 13 de noviembre). Actualmente los FPGAs poseen bloques o cortes de DSP, los cuales son regiones de *hardware* especializado dentro del FPGA con soporte para múltiples funciones independientes, que de ser conectados con otros cortes de DSP permiten generar funciones matemáticas más complejas, filtros digitales y aritmética compleja sin utilizar la lógica general del FPGA. (Xilinx, 2018)

La complejidad de los sistemas electrónicos ha dado lugar no solo a la utilización de FPGAs, sino que también a los sistemas en chip o SoC. Los SoCs tienen la característica de integrar distintos componentes electrónicos en un único chip de silicio. El Zynq-7000 SoC fue el primer SoC desarrollado por Xilinx y en su arquitectura cuenta con dos partes principales: el sistema de procesamiento y la lógica programable. El sistema de procesamiento incluye la unidad de procesamiento de aplicaciones, interfaces e interconexiones de memoria y periféricos de entrada/salida. La lógica programable incluye alguna variación de FPGAs de la Serie-7 de Xilinx, cortes de DSP48x, *Block* RAMs, transceptores de alta velocidad, bloques de comunicación, así como también lógica de propósito general.

Los sistemas embebidos basados en Zynq pueden ser concebidos como una pila de capas, extendiéndose desde un diseño de *hardware* en la base; hacia una capa de sistema operativo, que incluye *drivers* de *software* de bajo nivel y APIs para interconectar con el *hardware*; hasta aplicaciones que corren sobre la parte superior del SO. Los *Overlays*, o librerías de *hardware*, son diseños configurables en el FPGA que permiten extender la aplicación del usuario desde el sistema de procesamiento en el Zynq hacia la lógica programable (Xilinx, 2020b).

La complejidad en el diseño de estos sistemas embebidos se mitiga con el entorno de trabajo PYNQ, el cual consiste en una serie de componentes de software y hardware que el desarrollador puede utilizar para los sistemas basados en el uso del SoC Zynq. En particular, el entorno PYNQ acelera el co-diseño de *software*/*hardware* de sistemas embebidos Zynq, y hace más sencilla la tarea de interconectar la lógica programable y el sistema de procesamiento (Crockett et ál., 2019).

Esta investigación presenta al lector el diseño e implementación de un filtro FIR paso banda utilizando el entorno de trabajo PYNQ, así como también los distintos compromisos que se pueden incurrir al momento de diseñar el filtro. Finalmente, esta implementación es comparada con otras implementaciones en software con la finalidad de mostrar la aceleración por *hardware* obtenida por la implementación de un filtro FIR.

MÉTODOS Y PROCEDIMIENTOS

La implementación de un filtro FIR pasa banda en *hardware* es un proceso que conlleva tres etapas que se describen a continuación:

Diseño y determinación de coeficientes del filtro FIR

El diseño del filtro se basa en el método de ventana, el cual consiste en combinar un filtro *Sinc* con una función de ventana tipo *Blackman*, para obtener los

coeficientes del filtro FIR. El *kernel* o núcleo del filtro se encuentra dado por la expresión:

$$h[k] = \frac{sinc\left(2\pi f_c\left(k-\frac{M}{2}\right)\right)}{\pi f_c\left(k-\frac{M}{2}\right)} \times \left[0.42 - \frac{1}{2}\cos\left(\frac{2\pi k}{M}\right) + 0.08\cos\left(\frac{4\pi k}{M}\right)\right] \quad (1)$$

Donde f_c es la frecuencia normalizada de corte de un filtro paso bajo; M es la longitud del núcleo y está dada por:

$$M = \frac{4}{BW} \quad (2)$$

siendo BW la banda de transición del filtro, es decir la transición del 99 % al 1 % del filtro; y k es una muestra que se encuentra entre 0 y M.

En el caso del filtro pasa banda, también es necesario realizar la inversión espectral de algunos filtros calculados. La inversión espectral consiste en la obtención de la respuesta en frecuencia complementaria de un filtro por medio de la sustracción de la respuesta del filtro original a toda la respuesta en frecuencia del sistema. La respuesta del filtro complementario puede ser obtenida como:

$$\overline{h[k]} = \delta[k - M/2] - h[k] \quad (3)$$

El procedimiento para obtener los coeficientes del filtro pasa banda es el siguiente:

1. Diseñar dos filtros paso bajo con frecuencias de corte f_a y f_b, donde $f_a < f_b$.
2. Realizar una inversión espectral al filtro de frecuencia de corte f_b.
3. Sumar vectorialmente ambos núcleos del filtro.
4. Realizar una inversión espectral al filtro generado en el paso anterior.

El resultado de este proceso consiste en un filtro pasa banda de longitud M que puede ser implementado en *hardware*. Para este caso los coeficientes son valores enteros de 16 bits que pueden ser fácilmente trasladados en *hardware*.

Para implementar el filtro pasa banda se utilizó una señal de prueba con una frecuencia de muestreo $f_s = 44100\,Hz$, la cual es una frecuencia de muestreo estándar para la mayoría de audio de consumo. El filtro se seleccionó para tener las frecuencias de corte normalizadas $f_a = 0.14$ y $f_b = 0.25$; una banda de transición $BW = 0.08$. En consecuencia, la longitud del filtro a implementar tiene una longitud M de 50 coeficientes. Estos valores fueron seleccionados debido a que en este rango de frecuencias es donde se da la mayor pérdida de capacidad auditiva en las personas, y la separación y amplificación de estas componentes espectrales tienen mucha aplicación en la corrección de este problema, por lo que hacen de ellas especial interés en el análisis de señales digitales (Krug, E., 2015).

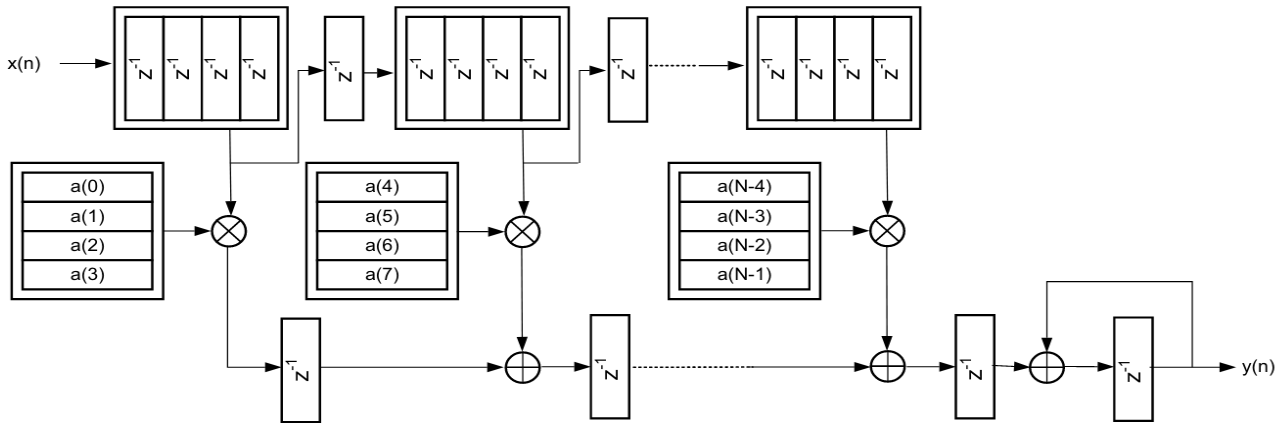
Implementación de Lógica Programable

La implementación en la lógica programable se encuentra limitada por números enteros de 16 bits, por lo que es necesario un cambio de escala y la cuantización de los coeficientes del filtro. El cambio de escala puede ser obtenido al multiplicar por algún factor numérico que permita incrementar el rango de los coeficientes del filtro, sin caer en sobre flujo, por lo que en este caso se ha multiplicado por el factor de 2^{14}, que a su vez es un corrimiento de 14 bits en su forma binaria. De forma similar, la cuantización consiste en truncar los coeficientes a 16 bits. Cabe destacar que este proceso introduce un error de cuantización (Smith, 1997) que puede ser disminuido por el nuevo rango de los coeficientes del filtro.

La implementación del filtro FIR se ha realizado en el *software* Vivado 2019.1 de Xilinx, y su generación

con el FIR Compiler de Xilinx usando los coeficientes del filtro anteriormente obtenidos. Se ha especificado una optimización para **velocidad de ejecución con una arquitectura sistólica de multiplicador y acumulador,** como se muestra en la Figura 1, pues la misma permite explotar la simetría de los coeficientes del filtro (Xilinx., 2020a)**.** Su integración al Sistema de Procesamiento se realiza por medio de una interfaz AXI con Acceso Directo a Memoria, DMA. Todo este sistema es sintetizado por el *software* Vivado 2019.1 para generar un *overlay* en la tarjeta PYNQ-Z1.

Figura 1. Arquitectura sistólica de multiplicador y acumulador.



Fuente: Xilinx., 2020a

Integración de Lógica Programable y Sistema de Procesamiento

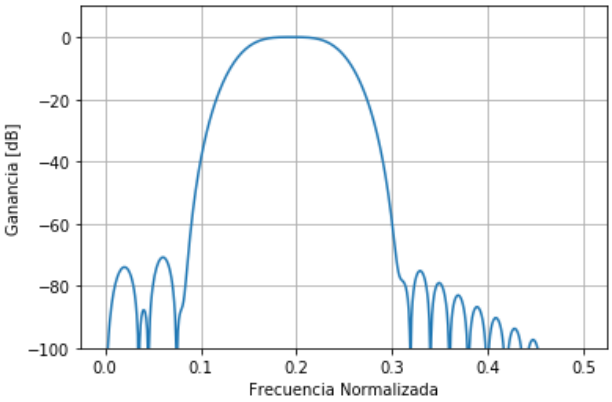
La integración de todo el sistema se realiza en *software* por medio del lenguaje de programación Python. En este caso, el *overlay* generado funcionará como un acelerador por *hardware* de un filtro FIR, por lo que al momento de que el sistema tenga una llamada a ejecutar una rutina de filtrado todo el procesamiento será delegado al mismo y el movimiento de datos se realizará por DMA en una arquitectura maestro-esclavo.

RESULTADOS

La Figura 2 muestra los resultados obtenidos de la implementación del filtro pasa banda por medio del algoritmo descrito anteriormente. En la Figura 3 se muestra el filtro rechaza banda que puede ser generado a partir del mismo algoritmo al omitir la última inversión espectral.

En la Figura 4 se observa el efecto de cuantización y multiplicación sobre la respuesta en frecuencia del filtro. Se puede observar un mayor efecto en la banda de rechazo y en la generación de lóbulos más grandes que resultan en una menor atenuación en dicha banda. Resultados similares pueden ser encontrados en (Mehboob et ál, 2009) que concuerdan con los obtenidos en la experimentación.

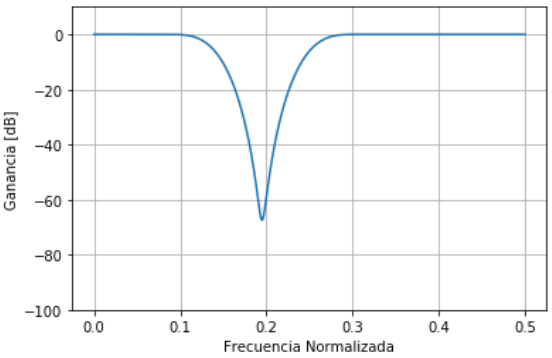
Figura 2. Filtro pasa banda con $f_a = 0.14, f_b = 0.25$y banda de transición $BW = 0.08$.



Fuente: Propia

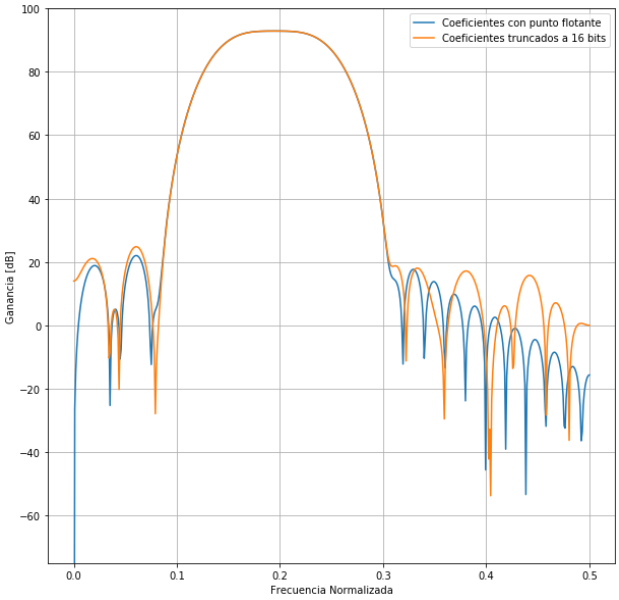
Los coeficientes cuantizados a 16 bits fueron entonces utilizados para el filtro FIR en *hardware* y generar una arquitectura maestro esclavo por DMA entre el filtro y el procesador ZYNQ como se muestra en la Figura 6.

Figura 3. Filtro rechaza banda con $f_a = 0.14, f_b = 0.25$y banda de transición $BW = 0.08$.



Fuente: Propia

Figura 4. Efecto de cuantización y multiplicación sobre coeficientes de punto flotante.



Fuente: Propia

Para la implementación del filtro se reportan 101 cortes de DSP utilizados de un total de 220 disponibles, es decir que un 46 % de los cortes de DSP fueron utilizados para esta implementación. Así mismo la potencia total del sistema es de 1.325 W y la del filtro es de 0.035W lo que representa un 2.6 % de potencia total del sistema.

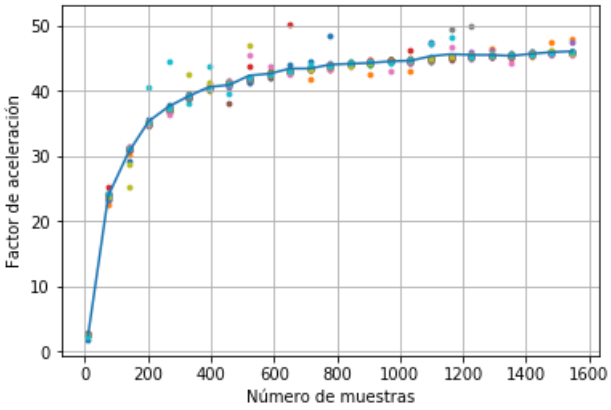
Tabla 1. Valores promedio del factor de aceleración contra el número de muestras de la señal de entrada.

Muestras	Aceleración
10	2
138	32
266	38
394	41
522	42

649	43
777	44
906	44
1034	45
1161	45
1290	45
1418	45
1545	46

Fuente: Propia

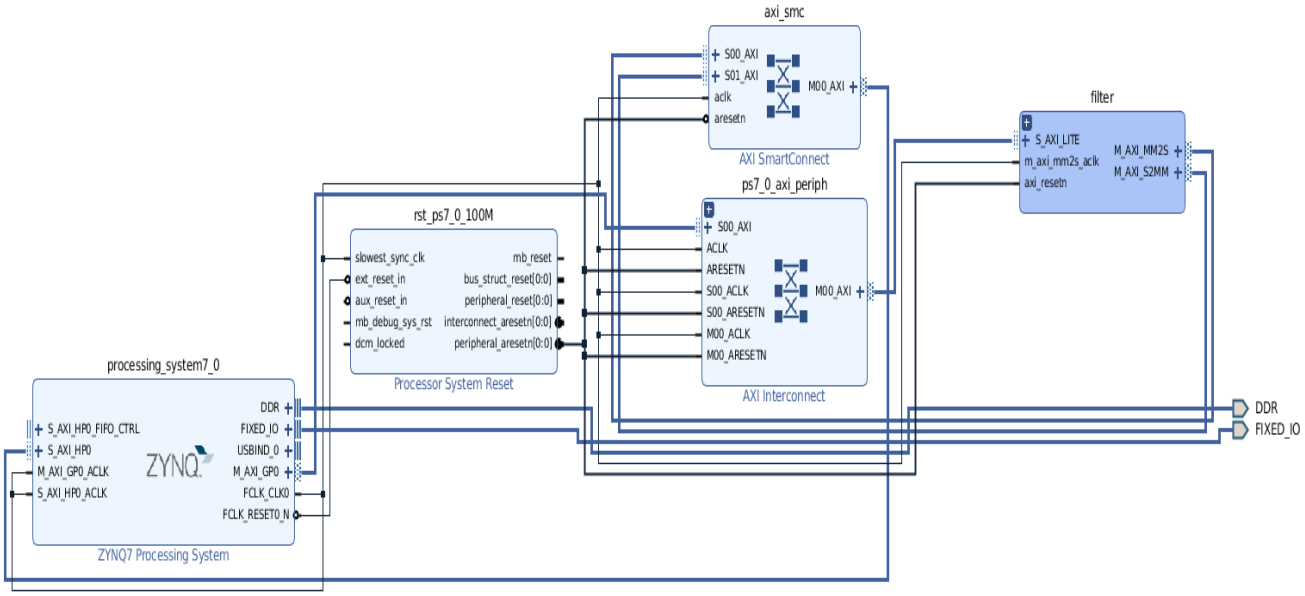
Figura 5. Resultados del factor de aceleración contra el número de muestras de la señal de entrada.



Si se define al factor de aceleración como la relación entre el tiempo de ejecución por *software* y el tiempo de ejecución por *hardware*, en la Tabla 1 se muestran los valores promedio del factor de aceleración que se alcanza al implementar el filtro FIR en *hardware* en función de la longitud que tiene la señal de entrada. Por ejemplo, para una señal que tiene una longitud de 1,750,592 muestras, la implementación por *hardware* alcanza un tiempo de ejecución de 19ms, en comparación de los 939 ms que le toma la ejecución por *software*, por lo que el factor de aceleración por *hardware* es de 49. La Figura 5 presenta los resultados del factor de aceleración contra el número de muestras de la señal de entrada, la línea continua presenta el valor promedio frente a varios resultados obtenidos.

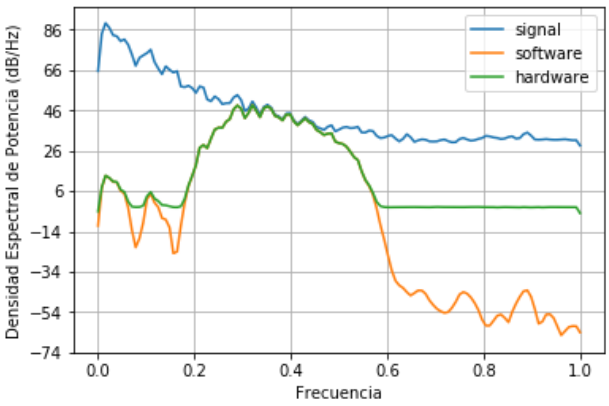
Finalmente, los resultados obtenidos al filtrar por *software* y por *hardware* se muestran en la Figura 7. Para obtener estos resultados, luego del proceso de filtrado, ambos sistemas se normalizan y amplifican, con lo cual es posible una comparación directa del resultado de los filtros. Se puede observar que en la banda de paso ambos filtros se comportan de la misma manera, sin embargo, existe un mejor rechazo en la banda de rechazo para el filtro por *software*.

Figura 6. Arquitectura maestro esclavo por DMA entre el filtro y el procesador ZYNQ.



Fuente: Propia

Figura 7. Comparación de filtro por *hardware* y *software* para una señal de entrada.



Fuente: Propia

DISCUSIÓN DE RESULTADOS

Los resultados obtenidos muestran que existe un factor de aceleración en la implementación del filtro FIR pasa banda por *hardware*. La aceleración que se puede alcanzar depende en gran medida de la cantidad de muestras que deben de ser procesadas por el filtro; y a una menor cantidad de muestras, el factor de aceleración será menor que al utilizar una mayor cantidad de muestras. Sin embargo, la aceleración tiene un comportamiento asintótico conforme el número de muestras se incrementa. La importancia de la implementación por *hardware* radica en la capacidad de poder realizar sistemas en tiempo real, pues la cantidad de muestras a procesar es muy alta y la latencia de un sistema por *software* sería inaceptable para este tipo de aplicaciones.

La implementación del filtro FIR pasa banda por *hardware* implica truncar los coeficientes a números enteros de 16 bits, lo cual resulta en una menor atenuación en la banda de rechazo del filtro pasa

banda. Aun cuando esta diferencia en la atenuación de ambas implementaciones parece ser significativa, la atenuación del filtro por hardware cumple con estar más de 3dB por debajo de la frecuencia de corte, lo que hace que sea aceptable para sistemas de audio digital. Sin embargo, en sistemas en los cuales una atenuación específica en la banda de rechazo fuera necesaria, dicha implementación no cumpliría con los requisitos del filtro.

La implementación del filtro FIR pasa banda consume un total de 101 cortes de DSP para 50 coeficientes del filtro, por lo que como regla de oro podremos establecer que la cantidad de cortes de DSP será del doble al número de coeficientes del filtro. Así mismo, aún si se utilizaran todos los cortes de DSP la potencia total consumida por los mismos no se incrementa en más del 5 % del total del sistema.

Finalmente, la versatilidad del entorno de trabajo PYNQ permite que la aceleración por *hardware* sea integrable a sistemas más complejos, y desde el punto de vista de un desarrollador, un *overlay* sea únicamente una librería que permite acceder a una funcionalidad más del sistema.

CONCLUSION

La implementación de un filtro FIR en hardware permite acelerar el rendimiento de cómputo frente a una versión en software. Para un filtro FIR de 50 coeficientes es posible alcanzar un factor de aceleración de hasta 45 veces su rendimiento en software al implementarlo en hardware utilizando números enteros de 16 bits. De igual manera, este factor de aceleración aumenta conforme el número de muestras a procesar por el filtro se incrementa, por lo que el beneficio de esta implementación es notable en situaciones donde existan altas tasas de procesamiento de datos. Finalmente, el costo de una implementación en hardware con valores enteros se da en el dominio de la frecuencia, en la banda de rechazo se obtiene una menor atenuación frente a su contraparte de punto flotante.

REFERENCIAS BIBLIOGRÁFICAS

Antoniou, A. (2018). *Digital filters: Analysis, design, and signal processing applications*. McGraw Hill Education.

Collins, T. F., Getz, R., Pu, D., y Wyglinski, A. M. (2018). *Software-defined radio for engineers*. Artech House.

Cooper, K. D., y Torczon, L. (2013). *Engineering a compiler*. Morgan Kaufmann.

Crockett, L. H., Northcote, D., Ramsay, C., Robinson, F. D., y Stewart, R. W. (2019). *Exploring Zynq MPSoC: With PYNQ and machine learning applications*. Strathclyde Academic Media.

Hill, C. (2016). *Learning Scientific Programming with Python*. Cambridge University Press.

Hyndman, R. J., y Athanasopoulos, G. (2018). *Forecasting principles and practice*. O Texts, online open-access textbooks.

Kastner, R., Matai, J., y Neuendorffer, S. (2018). Parallel programming for fpgas. *ArXiv:1805.03648 [Cs]*. http://arxiv.org/abs/1805.03648

Krug, E. (2015). Hearing loss due to recreational exposure to loud sounds: a review. Geneva, Switzerland: World Health Organization.

Lapedus, M. (2006, November 13). FPGAs can outperform DSPs, says study - Mark Lapedus. Obtenido de https://www.edn.com/fpgas-can-outperform-dsps-says-study/

Mehboob, R., Khan, S., & Qamar, R. (2009). FIR filter design methodology for hardware optimized implementation. IEEE Transactions on Consumer Electronics, 55(3), 1669-1673.

https://doi.org/10.1109/tce.2009.5278041

Mertins, A. (2000). *Signal analysis: Wavelets, filter banks, time-frequency transforms and applications*. John Wiley & Sons.

Smith, S. W. (1997). The scientist and engineer's guide to digital signal processing. California Technical Publ.

Vaidyanathan, P. P. (2008). *The theory of linear prediction*. Morgan & Claypool.

Xilinx. (2018). *7 Series DSP48E1 Slice User Guide*. https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

Xilinx. (2020a). *FIR Compiler v7.2LogiCORE IP Product Guide*. https://www.xilinx.com/support/documentation/ip_documentation/fir_compiler/v7_2/pg149-fir-compiler.pdf

Xilinx. (2020b). *Python productivity for Zynq (Pynq) Documentation*. https://buildmedia.readthedocs.org/media/pdf/pynq/latest/pynq.pdf