UNIVERSIDAD POLITECNICA DE VALENCIA

ESCUELA POLITECNICA SUPERIOR DE GANDIA

Master en Ingeniería Acústica

Implementación de un procesador digital de audio basado en FPGA

TESIS DE MASTER

Autor:
David Lorente Izquierdo

Director/es:
D. Germán Ramos Peinado
D. Rafael Gadea Gironés

GANDIA, Julio de 2013.

IMPLEMENTACIÓN DE UN PROCESADOR DIGITAL DE AUDIO BASADO EN FPGA

Autor: David Lorente Izquierdo
Director: Germán Ramos Peinado
Director: Rafael Gadea Gironés

Resumen — El procesado digital de señal aplicado al audio es un campo bien conocido y viene siendo aplicado con éxito creciente desde hace varias décadas. Sin embargo, los dispositivos utilizados para implementar algoritmos digitales de audio han sido tradicionalmente dispositivos de tipo DSP o procesadores de propósito general. Por otra parte, el procesado digital de señal basado en dispositivos FPGA es un campo en continuo desarrollo, mostrándose los dispositivos FPGA como una alternativa en crecimiento frente a los DPS, en algoritmos difícilmente implementables sobre DSP o sobre procesadores de propósito general. En el presente trabajo se estudia la viabilidad de adaptar algoritmos clásicos de procesado de audio a las características particulares de los dispositivos FPGA.

Palabras clave: DSP, FPGA, AUDIO, IIR, FIR

Abstract — Digital signal processing applied to audio is a well known area, and it has been successfully applied for several decades. However, the devices used to implement digital audio algorithms have been traditionally DSP type devices or general purpose processors. Moreover, the digital signal processing based on FPGA devices is a continuously developing field, showing FPGA devices as a growing alternative compared to DSP when algorithms are difficult to implement on DSP or general purpose microprocessors. In this paper we study the feasibility of adapting classical digital audio processing algorithms to the particular characteristics of FPGA devices.

Keywords: DSP, FPGA, AUDIO, IIR, FIR

Autor: David Lorente Izquierdo, email: daloiz@epsg.upv.es
Fecha de entrega: 8 de julio de 2013

ÍNDICE

I OBJETO DE ESTUDIO

El procesado digital de señal aplicado al audio es un campo bien conocido y muy utilizado desde hace ya varias décadas. La mayoría de investigaciones sobre procesado digital aplicado al audio se orientan principalmente hacia el desarrollo de algoritmos y su implementación final en dispositivos de tipo DSP (*Digital Signal Processor*), condicionando en parte la investigación sobre algoritmos y estructuras de procesado digital hacia la implementación óptima en este tipo de dispositivos.

Sin embargo, en los últimos años se están empezando a utilizar otras alternativas para la implementación de procesadores digitales de audio, principalmente en procesadores con un gran número de canales, y con nuevos algoritmos que requieren o bien más capacidad de proceso de los dispositivos en los que se implementa el algoritmo, o bien nuevas implementaciones utilizando distintas aproximaciones al uso tradicional de implementación software sobre DSP.

El objetivo de esta tesis es estudiar la viabilidad de implementar el procesado de audio sobre dispositivos FPGA (*Field Programmable Gate Array*), los cuales permiten una gran paralelización de los cálculos y una capacidad de proceso superior en varios órdenes de magnitud. Los dispositivos FPGA constan de una serie de células hardware programables, de forma que se implementan diseños hardware en lugar de algoritmos en software.

La principal justificación del uso de nuevas tecnologías, distintas a la implementación tradicional sobre DSP para la implementación de algoritmos de procesado digital es, por un lado el aumento del número de canales que se ha producido en los últimos años en los procesadores y mesas de mezclas, y por otro lado la necesidad de implementar nuevos algoritmos para el desarrollo de nuevas aplicaciones, como sistemas de acústica variable basados en procesado digital o sistemas de direccionamiento electrónico de arrays de altavoces.

Hace apenas diez años las mesas de mezclas contaban habitualmente con un máximo de 48 a 52 canales de mezcla, especialmente en la sonorización en directo. En producciones en estudio el número de canales podía ser superior, pero en cualquier caso no era habitual utilizar un número de canales mucho más elevado.

Hoy en día, con el auge del procesado digital para audio, y con todas las ventajas que ha supuesto, como memorización de todos los parámetros de ajuste, automatización de mezclas, transporte digital de señal, etc… no es nada difícil que las producciones de cierta entidad utilicen más de cien canales de mezcla, tanto en sonorizaciones en directo como en estudio de grabación.

Por otro lado, se está utilizando el procesado digital de señal en audio para aplicaciones innovadoras en las que se requiere un gran número de canales de procesado y también una gran capacidad de proceso por canal, mucho más exigente desde el punto de vista de la capacidad de proceso que la tradicionalmente utilizada hasta ahora. Algunos ejemplos de este tipo de

aplicaciones son los sistemas de arrays de altavoces de tipo columna, formados por decenas de altavoces, en los que cada altavoz es procesado y amplificado individualmente, de forma que es posible controlar mediante el procesado digital tanto el ángulo de cobertura como el ángulo de inclinación del haz del sistema [1]. Otros ejemplos de sistemas que requieren un gran número de canales de procesado son los sistemas de acústica variable mediante algoritmos de procesado digital, en los que se utiliza un gran número de micrófonos y altavoces para variar digitalmente la acústica de una sala [2]. Un factor adicional muy importante a tener en cuenta que influye de forma importante en el aumento de la capacidad de proceso necesaria es el auge en el transporte digital de audio. Cobranet y Ethersound [3][4] son dos protocolos de transporte digital de audio con varios años en el mercado, pero la verdadera implementación masiva de transporte digital de audio se está produciendo actualmente, mediante un protocolo estandarizado por el IEEE, desarrollado y adoptado por la gran mayoría de los fabricantes más importantes en el campo del audio digital. El protocolo es el AVB, desarrollado por la alianza AVnu [5].

El enfoque de implementación sobre DSP se basa en utilizar una estructura hardware fija, e implementar en software las operaciones a realizar por dicha estructura. Este enfoque permite que la unidad de proceso hardware, MAC (*Multiplier-Accumulator*), sea muy especializada, y por lo tanto las operaciones se realizan a velocidades muy altas. Sin embargo el proceso es secuencial, por lo que la unidad de proceso debe utilizarse tantas veces como operaciones se deban realizar.

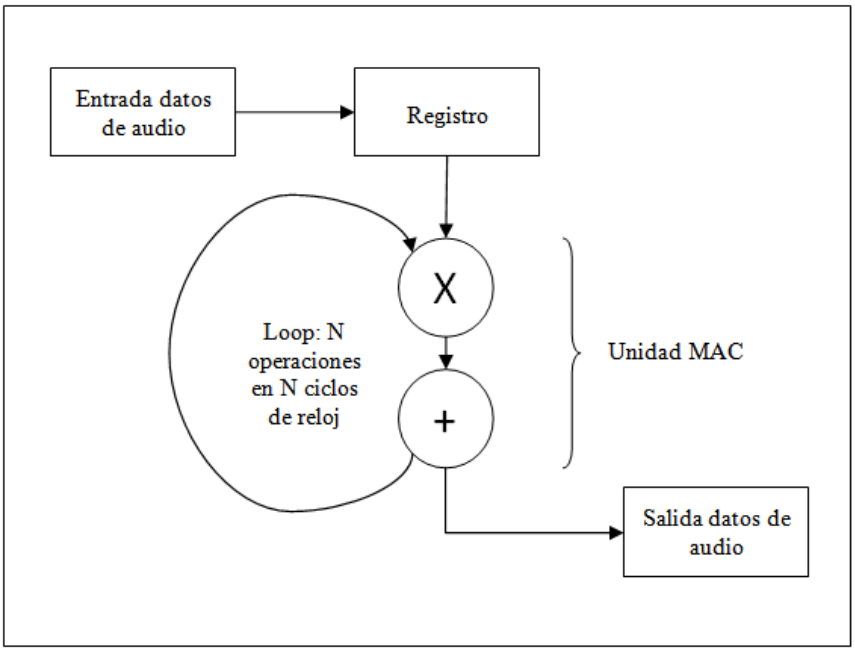

Fig. 1 Unidad de proceso de un dispositivo DSP

Un ejemplo de este tipo de arquitectura es el dispositivo específico para audio SigmaDSP del fabricante Analog Devices.

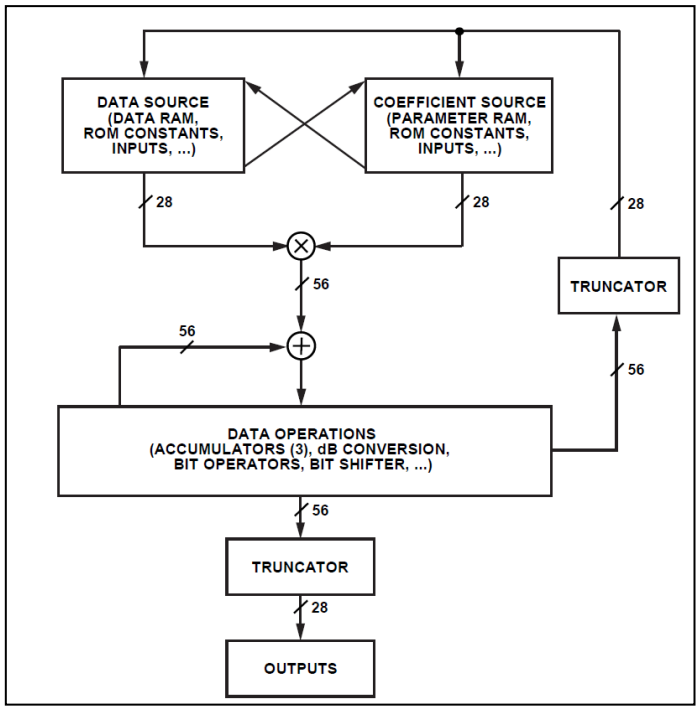

Fig. 2 Unidad de proceso del dispositivo SigmaDSP de Analog Devices

El enfoque basado en FPGA permite paralelizar el proceso, implementando tantos bloques hardware como sea necesario. Cada uno de estos bloques puede a su vez implementar secuencialmente las operaciones tantas veces como sea necesario. De esta forma se puede optimizar mucho más eficientemente el proceso, adaptando la estructura hardware al algoritmo a implementar y disponiendo de nuevos grados de paralelismo no presentes en DSP, que en general suelen disponer de 1 ó 2 unidades MAC por CPU como máximo.

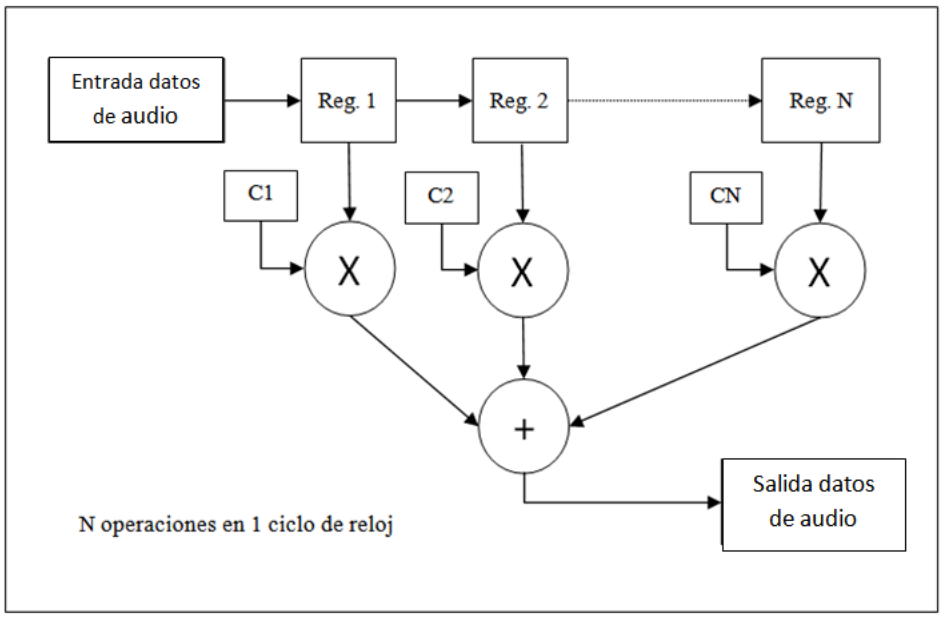

Fig. 3 Procesado distribuido en un dispositivo FPGA

II OBJETIVOS Y METODOLOGÍA

El objetivo del presente trabajo es estudiar la implementación tanto de algoritmos clásicos de procesado digital como de nuevos algoritmos, optimizados para ser implementados en una plataforma de desarrollo basada en dispositivos FPGA. El hecho de que se pueda distribuir el proceso entre muchas unidades MAC, no implica necesariamente que se deban usar todos los recursos de la FPGA para un proceso determinado. Cada una de las unidades MAC puede a su vez realizar operaciones en loop como en la arquitectura DSP. Habrá que encontrar un compromiso entre número de recursos utilizados o área, y velocidad de procesado. Ambos factores influirán notablemente en el dispositivo necesario para implementar determinado procesado, y por lo tanto en el precio del dispositivo a utilizar en la implementación.

La metodología para obtener estructuras de procesado de audio óptimas para ser implementadas en FPGA será estudiar distintas opciones utilizando para ello la herramienta System Generator proporcionada por Xilinx. Esta herramienta permite desarrollar algoritmos de procesado digital orientados a su implementación en FPGA utilizando los entornos Matlab y Simulink, los cuales permiten obtener medidas de ruido y otras características de interés mediante simulaciones, por lo que la implementación final en la FPGA sólo se realiza una vez que el algoritmo ha sido analizado y optimizado. Esta aproximación permite importantes ahorros en tiempo ya que la implementación final en la FPGA requiere generalmente tiempos de compilación muy altos que pueden llegar a ser de varias horas en sistemas complejos que incorporen muchas unidades funcionales y estructuras, e incluso uno o varios microprocesadores embebidos.

La investigación está enfocada a la implementación final en un dispositivo específico, por lo que se tendrán en cuenta los recursos disponibles en dicha FPGA. El dispositivo a emplear es el modelo de Xilinx Zynq-7020 de reciente aparición en el mercado. Este dispositivo incluye en el mismo chip, una parte de procesador y una parte de FPGA. La parte de procesador está compuesta por dos ARM CortexA-9 capaces de funcionar a frecuencias de 800 MHz. En la parte FPGA, que es el objeto de estudio de este trabajo, los recursos disponibles son:

- 85.000 células lógicas
- 53.200 Look-Up Tables
- 106.400 registros Flip-Flop
- 560 KB de bloques RAM
- 220 células específicas DSP para procesado digital

En cada uno de los algoritmos implementados se incluirá una tabla de utilización de recursos de dicho algoritmo, de forma que podremos saber qué porcentaje de recursos de la FPGA necesita cada implementación específica. En especial se intentará optimizar el uso de células DSP, especialmente previstas para implementar el tipo de algoritmos objeto de estudio. La célula DSP

que incluye la familia de dispositivos de la serie 7 de Xilinx, es el modelo DSP48E1 mostrado en la siguiente figura.

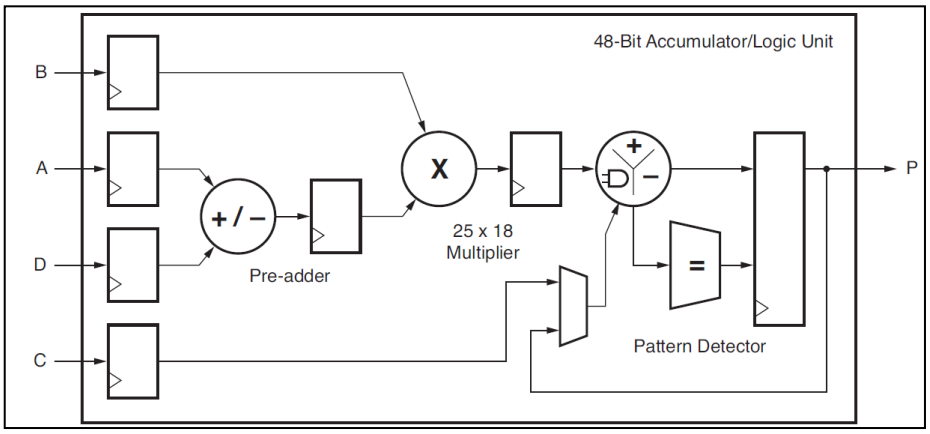

Fig. 4 Célula DSP48E1 incluida en los dispositivos de la serie 7 de Xilinx

Cada célula DSP incluye un multiplicador de 18x25 bits y un acumulador de 48 bits, por lo que se tratará de optimizar cada algoritmo adaptándolo a esta estructura en concreto.

III ESTADO DEL ARTE

La implementación de procesado digital de señal aplicado al audio ha sido tradicionalmente realizada sobre DSP, sobre los cuales existen infinidad de aplicaciones y herramientas de desarrollo optimizadas específicamente para audio. En sistemas multicanal como mesas de mezclas o grandes matrices de procesado, la solución empleada ha sido la de utilizar un conjunto de DSPs interconectados mediante FPGA. Conforme la potencia de cálculo requerida y el número de canales aumentan, el proceso de señal empieza a realizarse con procesadores de propósito general multi-core de altas prestaciones como Intel Xeon, utilizándose técnicas especiales de paralelización del procesado, como las instrucciones SIMD (*Single-Instruction-Multiple-Data*) implementados en algunos de estos procesadores de propósito general o DSPs como la serie SHARC de Analog Devices. Un ejemplo de este tipo de procesador multicanal implementado mediante un procesador de propósito general y uso de instrucciones SIMD es el sistema D-Mitri de Meyer Sound [6].

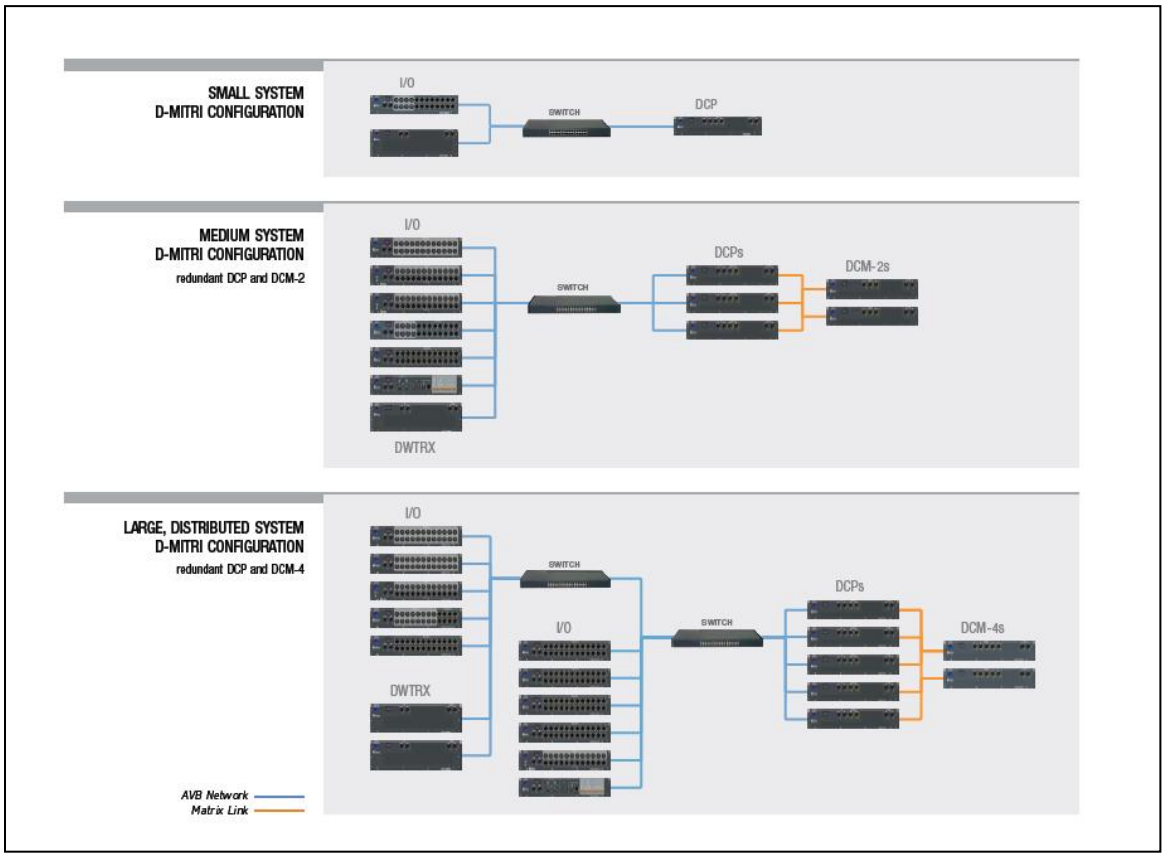

Fig. 5 Sistema de Procesado D-Mitri de Meyer Sound Labs, Inc.

Este sistema se compone de varios módulos configurables y es capaz de procesar hasta 288 canales de entrada, 288 matrices de rutado y 288 canales de salida, e incluye capacidad de transporte de audio sobre red ethernet.

Este tipo de sistemas se utilizan en grandes parques temáticos, producciones teatrales, musicales, sistemas de acústica variable, etc… El hecho de que estos sistemas incluyan normalmente el transporte digital de señal hace que una de las principales tendencias que se observa en la industria para este tipo de soluciones sea la de centralizar todo el procesado en una gran unidad principal, lo cual implica que las necesidades de capacidad de proceso de este tipo de unidades son cada vez mayores.

Los algoritmos de procesado y su implementación software, bien sea en DSP o en procesadores de propósito general, están muy estudiados, y en general bien resueltos. Sin embargo, la implementación hardware sobre FPGA no está tan estudiada, y mucho menos su implementación generalizada. Apenas algunas pocas empresas a nivel mundial están estudiando y aplicando las enormes posibilidades de desarrollo que las FPGA ofrecen en el campo del procesado digital de audio [7].

Aunque las FPGA se utilizan cada vez más para implementar procesado de señal, en sustitución de los DSP o de los procesadores de propósito general, este uso se limita a aplicaciones de muy alta frecuencia o con grandes necesidades de cálculo, donde los DSPs no son capaces de asumir todo el

proceso. La frecuencia de muestreo empleada en audio profesional, de 48 ó 96 KHz habitualmente, es relativamente baja comparada con otras disciplinas como procesado de video o comunicaciones. Sin embargo, las especiales características propias del procesado del audio, la creciente complejidad de los procesos de señal necesarios en audio, el aumento del número de canales así como la cada vez más presente transmisión de audio digital, hace que cada vez tenga más sentido el uso de FPGA como opción en la implementación de procesadores de audio.

El principal problema que presenta el desarrollo de algoritmos de audio sobre FPGA es la falta de herramientas, investigaciones y soluciones específicas de algoritmos de audio aplicados a la arquitectura particular de las FPGA, lo que hace que el tiempo de desarrollo sea sensiblemente superior al utilizado en el desarrollo de algoritmos de audio sobre DSP.

IV COMA FIJA Y COMA FLOTANTE EN AUDIO

Las diferencias entre coma fija y coma flotante son bien conocidas en general, aunque algunas particularidades de los sistemas de audio introducen factores que hay que tener en cuenta [8]. Hablar de coma fija o coma flotante significa que un número definido mediante un tamaño fijo de bits, se puede representar en distintos sistemas. Por ejemplo, si tenemos 24 bits de tamaño de palabra, y decidimos utilizar una representación en coma fija con la coma posicionada en el bit 23, tendremos un sistema que nos permite representar un gran número de decimales, pero muy pocas posiciones de números enteros. Tendremos por lo tanto mucha resolución pero muy poca diferencia entre el máximo y el mínimo número que podemos representar. Si desplazamos la coma de forma que la parte entera ocupe más bits, tendremos mayor rango de representación entre el máximo y el mínimo valor, pero a costa de reducir las posibilidades de representar números decimales, por lo que disminuye la resolución. Es decir, mediante la posición de la coma, se busca una solución de compromiso entre dos parámetros fundamentales en un sistema de representación: la resolución y el rango dinámico. La resolución es la mínima distancia entre dos números que el sistema es capaz de representar, mientras que el rango dinámico es la diferencia entre el número más grande y el más pequeño en una representación determinada.

En un sistema de coma flotante, la posición de la coma cambia de forma automática en función del número que se desea representar, de forma que siempre se alcanza un compromiso entre resolución y rango dinámico. Además, la forma de mover dicha posición de la coma se realiza mediante el uso adecuado de exponentes, por lo que el rango dinámico se extiende enormemente. Sin embargo, aunque en general el uso de coma flotante facilita y optimiza la implementación de casi cualquier tipo de algoritmo, en audio se dan circunstancias especiales que pueden presentar algunos problemas específicos. Cuando en un sistema de coma flotante se intentan sumar dos números en rangos distintos, es decir: se desea sumar un número muy grande con otro muy pequeño, la falta de resolución en el rango del número grande hace que el pequeño desaparezca, y

la suma no se realice correctamente. En general esto no supone un problema, pero en audio se utilizan estructuras en las que este hecho produce ruido de cuantización que se realimenta y puede llegar a ser inaceptable bajo determinadas circunstancias [9].

Además los sistemas de coma flotante alcanzan frecuencias de operación menores, necesitan mayor uso de recursos, tienen mayor consumo y requieren más área para su implementación. Aunque en dispositivos FPGA es posible implementar algoritmos mediante representaciones en coma flotante, su coste en términos de área y utilización de recursos las hace inviables para la implementación de algoritmos de procesado de audio en coma flotante, por lo que se estudiarán especialmente las implicaciones de utilizar representaciones en coma fija para cada uno de los algoritmos presentados.

V ESTRUCTURAS DE FILTROS IIR

V.1 FILTROS IIR EN AUDIO

En el procesado digital clásico, el que implementan hoy por hoy la gran mayoría de procesadores de audio de propósito general, las mesas de mezclas, etc… el filtrado, en todas sus variantes y tipos es sin duda el elemento básico de procesado. En general, los bloques de filtrado pueden ser, desde el punto de vista de su respuesta en frecuencia, de tipos muy variados: all-pass, pasa-bajos, pasa-altos, crossovers en todas sus diferentes formas, filtros paramétricos, ecualizadores gráficos, etc…[10]. Sin embargo, todos estos tipos de filtros se implementan generalmente en forma de filtros IIR, utilizando en la mayoría de los casos el equivalente digital a diseños analógicos clásicos, y aplicando una transformación adecuada para pasar del dominio analógico al digital. La transformación más utilizada es la transformación bilineal, en la que se mapean las infinitas frecuencias analógicas al conjunto finito de frecuencias digitales [11].

$$s = f(z) = \frac{z-1}{z+1} = \frac{1-z^{-1}}{1+z^{-1}} \tag{1}$$

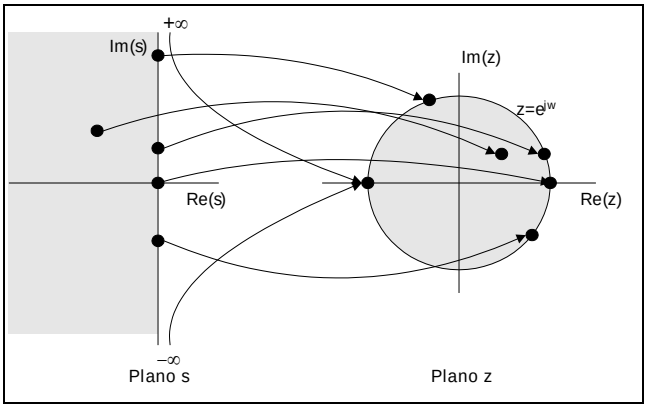

Fig. 6 Mapeado desde el plano analógico al plano digital.

La relación entre la frecuencia digital w y la analógica Ω vendrá dada al sustituir $s=j\Omega$ y $z=e^{jw}$ en la transformación anterior.

$$j\Omega = \frac{1-e^{-jw}}{1+e^{jw}} = \frac{e^{jw/2}-e^{-jw/2}}{e^{jw/2}+e^{-jw/2}} = j\cdot\frac{sen(w/2)}{\cos(w/2)} = j\cdot\tan\left(\frac{w}{2}\right) \tag{2}$$

$$\Omega = g(w) = \tan\left(\frac{w}{2}\right) \tag{3}$$

La función de transferencia digital en función de la función de transferencia analógica será por lo tanto:

$$H(z) = H_a(s)\Big|_{s=\frac{1-z^{-1}}{1+z^{-1}}} \tag{4}$$

Existen otras transformaciones para pasar del dominio analógico al digital, como la transformación impulso-invariante, el método de las derivadas o la transformación Matched-Z [12], pero todas ellas presentan inconvenientes importantes en su aplicación a sistemas de audio. En el caso de la transformación bilineal, el principal inconveniente es que se introduce una distorsión de fase y compresión espectral adicional a la existente en el dominio analógico. Sin embargo, mediante la adecuada elección de los parámetros de la transformación, dicha distorsión adicional es lo suficientemente pequeña como para no ser relevante en sistemas de audio.

En audio, el filtrado IIR es la opción predominante cuando se trata de implementar filtros clásicos como los mencionados anteriormente, por el amplio conocimiento que se tiene de estos filtros, y también por su eficiencia en cuanto a requerimientos de capacidad de proceso. El filtrado FIR, del que se habla más adelante, necesita mayor potencia de cálculo, y se reserva en general para aplicaciones de filtrado con requerimientos más específicos, o con especificaciones imposibles de conseguir con filtros IIR.

V.2 IMPLEMENTACIÓN MEDIANTE SECCIONES DE SEGUNDO ORDEN

La implementación de filtros IIR en audio, tanto en el dominio analógico como en el digital, se realiza mayoritariamente mediante el uso de secciones de segundo orden encadenadas en serie [11], lo que se conoce normalmente con el nombre de bloques *biquad*. La denominación biquad procede de la palabra ‘biquadratic’, que se usa por el hecho de que se trata de secciones con ecuaciones cuadráticas, es decir: de segundo orden, tanto en el numerador como en el denominador de la función de transferencia.

$$H(z) = G\cdot\frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{5}$$

Para conseguir filtros de orden mayor, se encadenan en serie tantas secciones biquad como sea necesario, pudiendo existir una sección de primer orden en el caso de filtros de orden impar.

Centrándonos únicamente en el dominio digital, las principales ventajas de utilizar secciones de segundo orden en lugar de funciones de transferencia de orden mayor, son básicamente dos: por un lado se minimizan los errores numéricos asociados a ecuaciones de órdenes altos, y por otro lado se estandariza la estructura, de forma que es más sencillo trabajar con las ecuaciones y obtener los coeficientes de las mismas.

El hecho de trabajar con bloques biquad como elemento fundamental para implementar filtros IIR, es en general independiente del hardware final utilizado. Trabajando con estructuras biquad como bloque fundamental, se puede aprovechar la extensa literatura e investigación que existe referente a los mismos, de forma que encontrando una implementación del bloque biquad sobre FPGA que cumpla los requisitos deseados, será posible adaptar de forma relativamente sencilla todos los filtros IIR existentes hasta el momento.

Por lo tanto los bloques biquad, y las diferentes estructuras utilizadas en su implementación, son el objetivo a estudiar para obtener implementaciones eficientes de filtros IIR sobre FPGA.

V.3 ESTRUCTURAS DE IMPLEMENTACIÓN BIQUAD EN COMA FLOTANTE

Tal y como se ha comentado anteriormente, la implementación en coma fija y la implementación en coma flotante presentan importantes diferencias. La implementación natural sobre FPGA es la implementación en coma fija, mientras que la implementación más habitual sobre DSPs en audio actualmente es la implementación en coma flotante.

La implementación de filtros IIR y estructuras biquad en coma flotante sobre DSP está muy estudiada, y en general presentan unas características buenas en cuanto a ruido y estabilidad. Sin embargo, dichas implementaciones biquad en coma flotante presentan algunos problemas característicos que pueden ser mejorados en la implementación sobre FPGA si la implementación en coma fija se realiza adecuadamente.

El principal problema en la implementación en coma fija tiene que ver con el tipo de estructura utilizada y el ruido que generan determinados tipos de filtros. Existen diferentes estructuras para implementar un bloque biquad. Las más recomendables por sus propiedades para ser usadas en audio en coma flotante y en coma fija son la forma directa 1 y la forma traspuesta 2 [13].

La forma directa 1 se corresponde con la siguiente ecuación en diferencias:

$$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) - a_1 y(n-1) - a_2 y(n-2) \quad (6)$$

Y se puede representar mediante el siguiente diagrama:

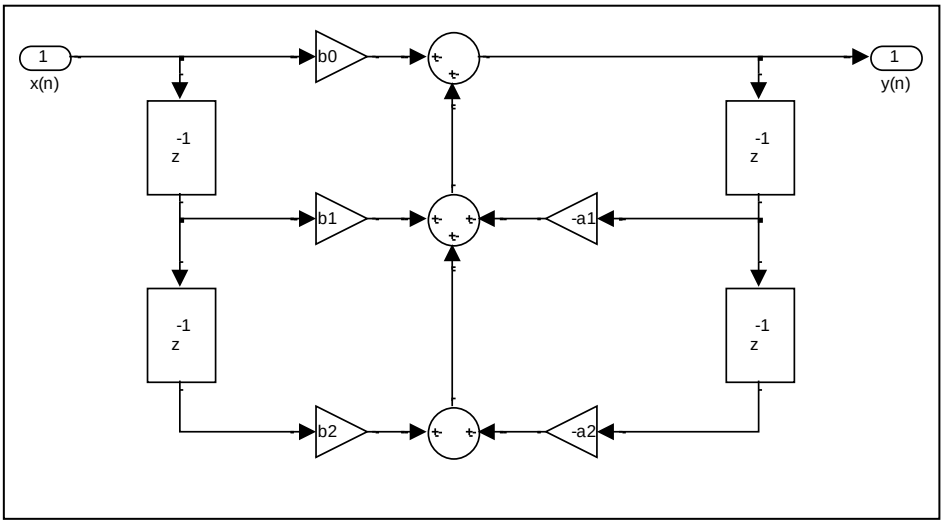

Fig. 7 Estructura biquad en Forma Directa 1

El problema que presenta esta estructura es el ruido de cuantización que genera cuando se implementan filtros paramétricos, muy habituales en audio, con una frecuencia central baja respecto a la frecuencia de muestreo y un factor de calidad Q alto. Un filtro paramétrico es un filtro de segundo orden que se define mediante tres parámetros principales: frecuencia central, ganancia y ancho de banda o factor Q.

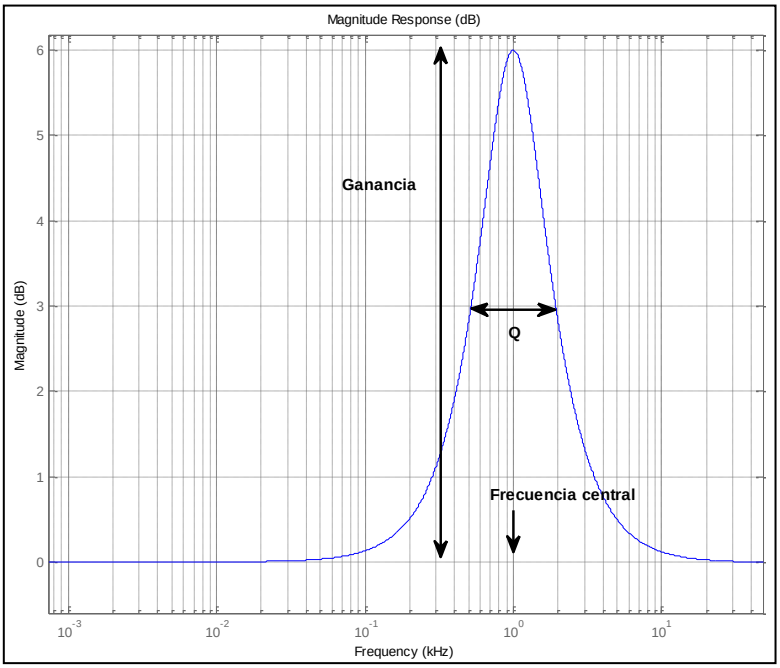

Fig. 8 Filtro paramétrico. Fc = 1KHz, G = 6 dB, Q = 1

Para observar los problemas de ruido que presenta esta implementación, realizamos una simulación en 32 bits en coma flotante mediante simulink en la que se implementa un filtro paramétrico con los siguientes parámetros:

- Frecuencia de muestreo = 96 KHz

- Fc = 30 Hz
- Q = 14
- G = 0.1 dB

Se introduce una entrada senoidal a la frecuencia central, en este caso 30Hz, y se observa el ruido producido a la salida del filtro:

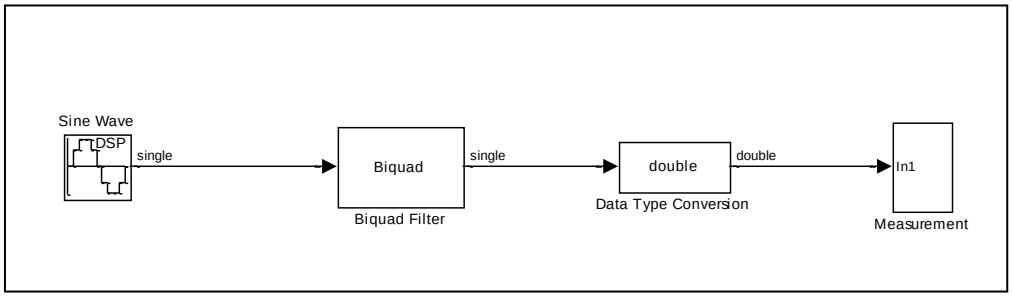

Fig. 9 Modelo para medición de ruido de cuantización en filtros biquad

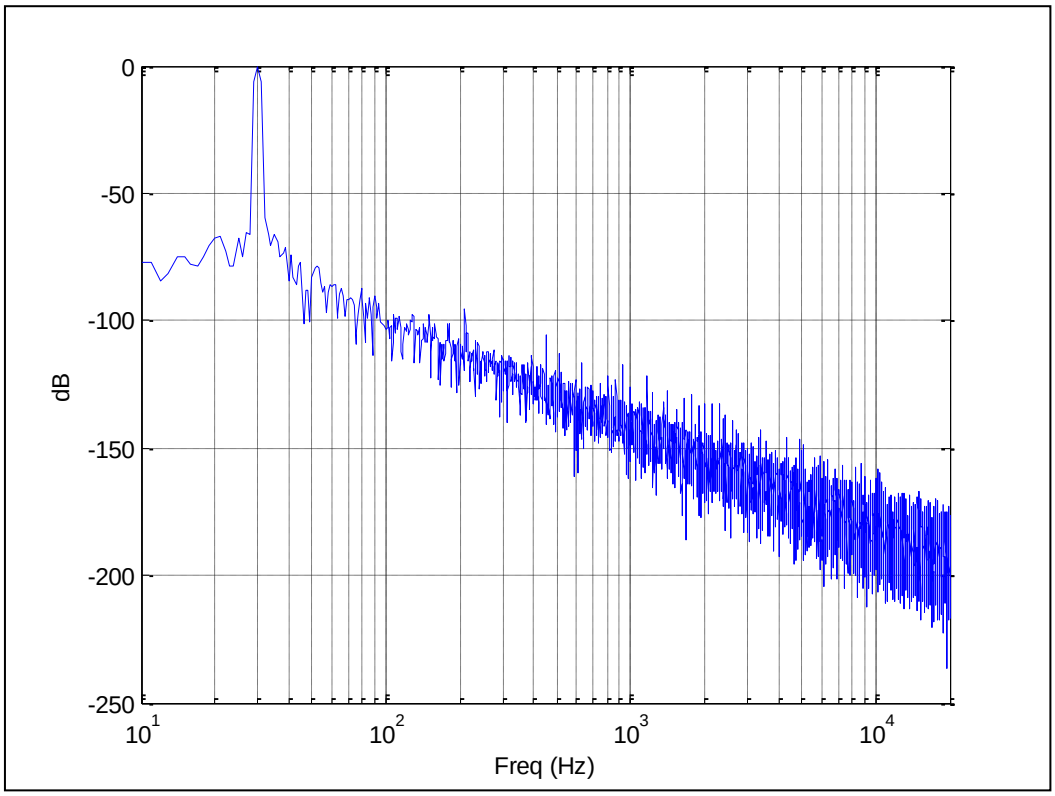

Fig. 10 Ruido de cuantización para una estructura biquad de 32 bits en coma flotante en FD 1

Se observa cómo el ruido en bajas frecuencias alcanza niveles no aceptables en audio profesional. La causa del excesivo ruido en baja frecuencia es la ganancia interna que ve la etapa de realimentación del filtro, cuya función de transferencia (los polos del sistema) presenta una ganancia enorme en frecuencias centrales bajas. Incluso en punto flotante, esta ganancia fuerza cambios de exponente considerables que deben normalizarse luego en el acumulador con valores más bajos.

Si los parámetros del filtro se centran en altas frecuencias, el ruido en bajas frecuencias desaparece, pero aparece distorsión en altas frecuencias, aunque con esta estructura el nivel es lo suficientemente bajo como para que pueda ser utilizada sin demasiados problemas. Por ejemplo, con los siguientes parámetros:

- Frecuencia de muestreo = 96 KHz
- Fc = 3200 Hz
- Q = 0.7
- G = 1 dB

Introduciendo una señal senoidal a la entrada, cuya frecuencia no sea submúltiplo de la frecuencia de muestreo, podemos observar el ruido a la salida del filtro. La señal utilizada a la entrada es de 997 Hz, y se observa el siguiente espectro a la salida del filtro:

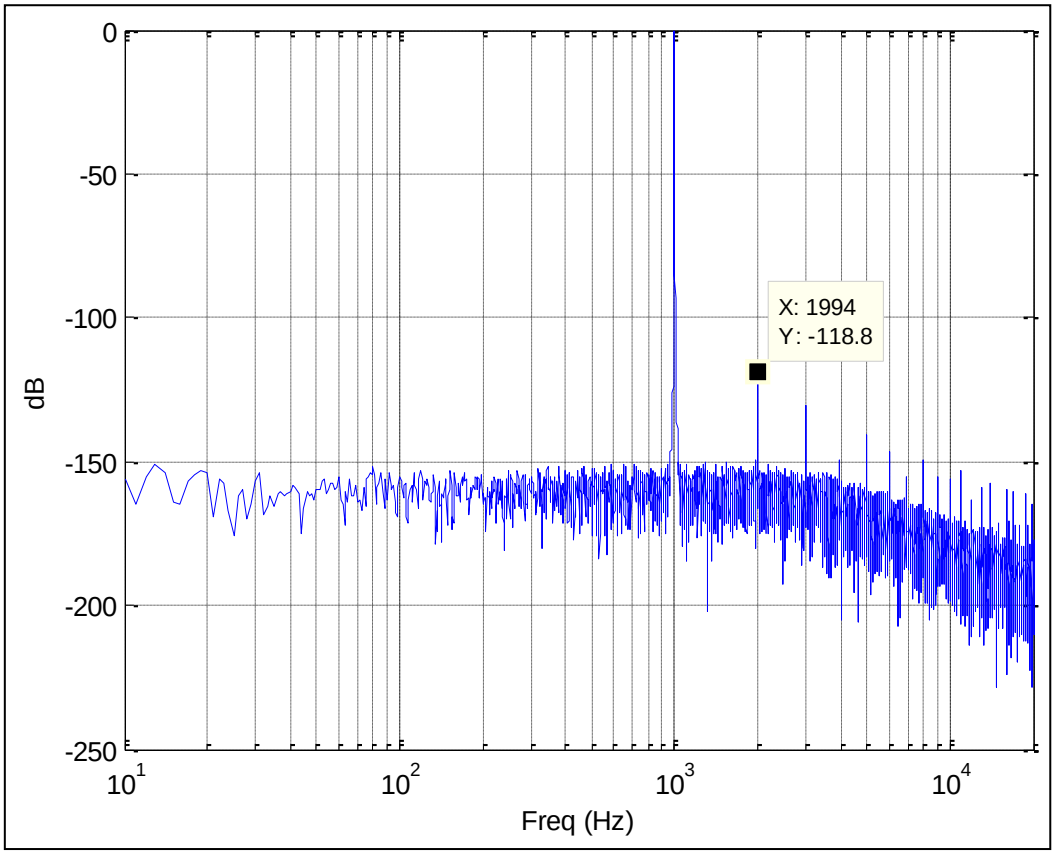

Fig. 11 Distorsión en altas frecuencias para una estructura biquad de 32 bits en coma flotante en FD 1

La forma traspuesta 2 es también ampliamente utilizada en audio, puesto que permite ahorrar memoria en la implementación, ya que es una estructura canónica, en la que los retardos son los mínimos posibles.

La estructura es:

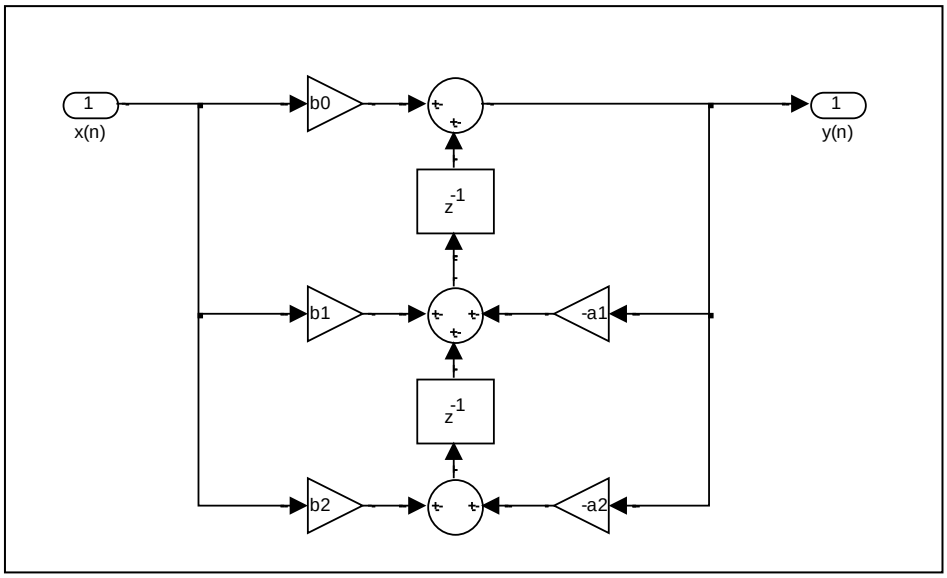

Fig. 12 Estructura biquad en Forma Traspuesta 2

Y la ecuación en diferencias correspondiente:

$$y(n) = b_0x(n) + b_1x(n-1) - a_1y(n-1) + b_2x(n-2) - a_2y(n-2) \quad (7)$$

En este tipo de estructura también presenta problemas de ruido, similares a la forma directa 1, cuando se implementa un filtro con los parámetros descritos anteriormente. Por ejemplo para un filtro paramétrico centrado en bajas frecuencias se obtiene:

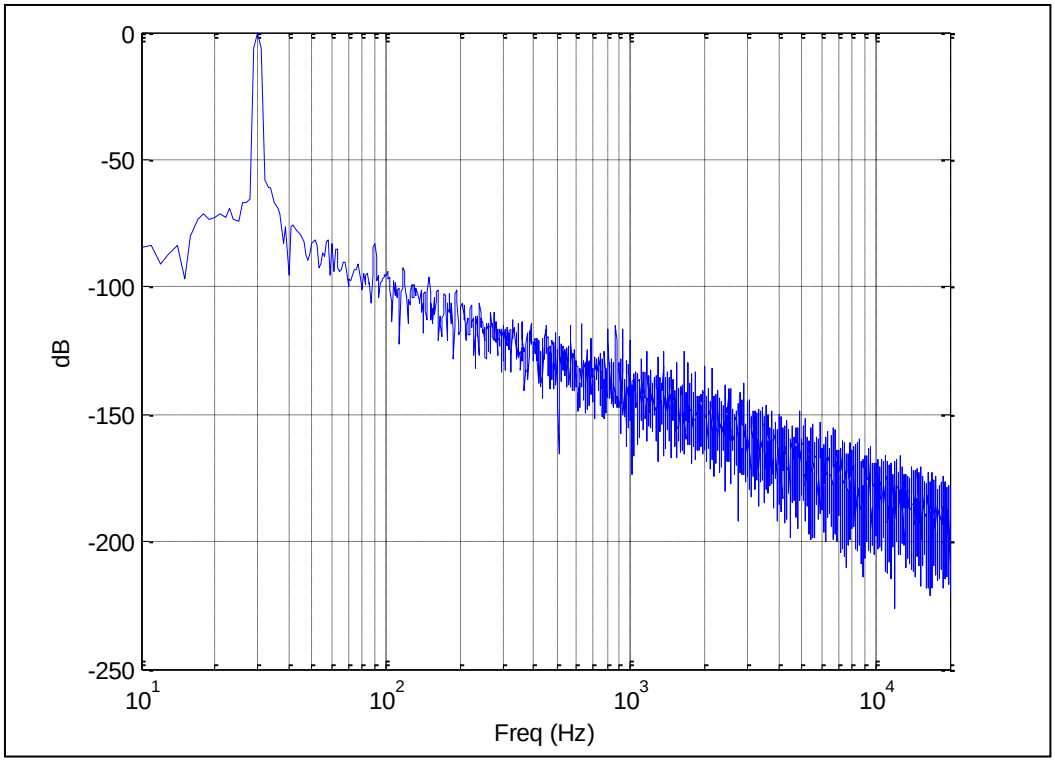

Fig. 13 Ruido de cuantización para una estructura biquad de 32 bits en coma flotante en FT 2

Para un filtro paramétrico centrado en altas frecuencias:

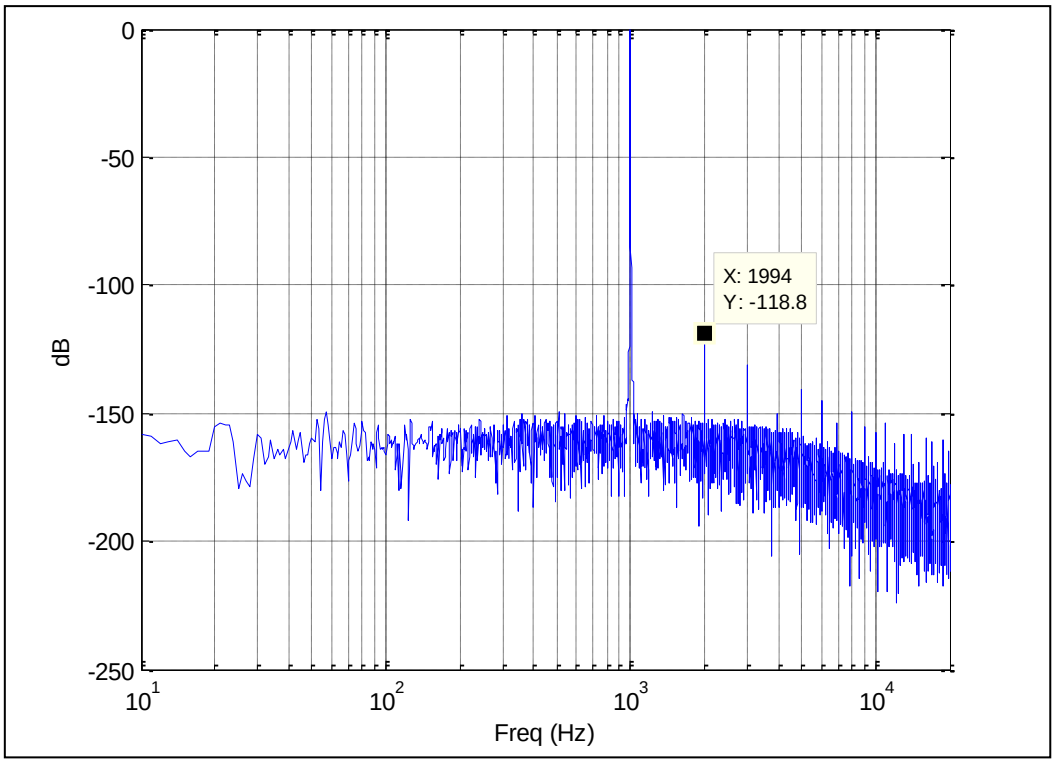

Fig. 14 Distorsión en altas frecuencias para una estructura biquad de 32 bits en coma flotante en FT 2

En otras estructuras, como la forma traspuesta 1 o la forma directa 2, el comportamiento del ruido en baja frecuencia es peor cuando los filtros se centran en bajas frecuencias. Por ejemplo para una estructura en forma directa 2, el ruido que se obtiene con el filtro en bajas frecuencias del ejemplo anterior es:

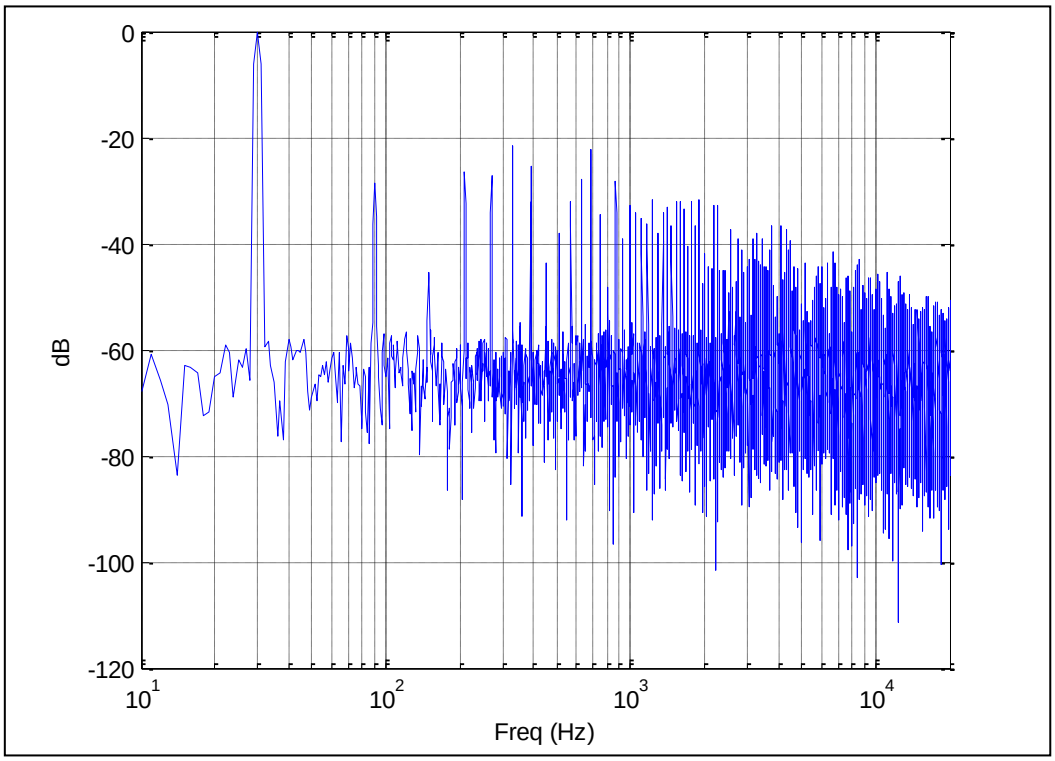

Fig. 15 Ruido de cuantización para una estructura biquad de 32 bits en coma flotante en FD 2

La conclusión es que, a pesar de ser la estructura más utilizada en audio profesional, no existe una estructura óptima implementada en 32 bits en coma flotante que se comporte de forma eficiente ante todos los posibles parámetros de entrada. La solución es o bien implementar el proceso en 64 bits, o bien utilizar estructuras optimizadas, que mejoran las características de ruido a costa de emplear más operaciones por cada bloque biquad.

V.4 ESTRUCTURAS DE IMPLEMENTACIÓN BIQUAD EN COMA FIJA

La implementación de filtros IIR en coma fija, presenta unas características específicas, distintas a la implementación en coma flotante, que hay que tener en cuenta si queremos implementar una estructura biquad de forma eficiente y robusta sobre una FPGA.

Los principales problemas que presenta la implementación en coma fija son los errores debidos al proceso de cuantización y al overflow. Mientras que en coma flotante estos errores son minimizados por el enorme rango dinámico y resolución que se obtiene al operar en coma flotante, en coma fija hay que tener especial cuidado al escalar adecuadamente las secciones biquad, así como al elegir el formato en número de bits de cada una de las operaciones que componen el biquad y la estructura a utilizar en la implementación. Aunque el proceso en sí es más complicado de analizar e implementar, el resultado obtenido optimiza mucho más los recursos para una determinada especificación de ruido que una implementación en coma flotante. Este hecho es especialmente importante en las FPGA, donde se puede ajustar el número de bits independientemente en cada una de las operaciones a realizar, y en las que las operaciones en coma flotante implican un alto coste en recursos.

Las posibles soluciones para evitar los problemas asociados a la implementación de estructuras biquad en coma fija son, o bien aumentar el número de bits, lo cual es siempre posible y sencillo de implementar en FPGA y supone una de sus mayores ventajas, o bien utilizar estructuras especiales que implementen el bloque biquad minimizando los errores que se producen, mediante el uso de operaciones adicionales.

Una de las posibles técnicas para obtener una estructura biquad en coma fija robusta y de bajo ruido consiste en añadir una rama de realimentación en la que se tiene en cuenta el error de cuantización cometido y se aplica una corrección en la realimentación [14]. Un ejemplo de dicha implementación se puede ver en la siguiente figura:

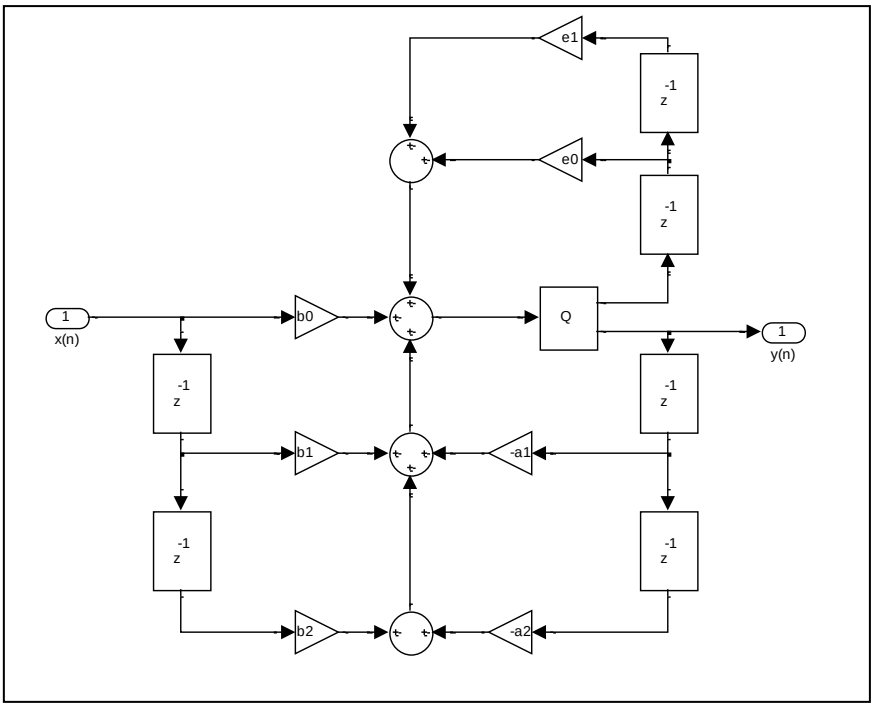

Fig. 16 Estructura biquad en Forma Directa 1 con realimentación del error

En este caso, a una estructura biquad implementada mediante la forma directa 1, se le añade una rama adicional en la que se realimenta el error de cuantización. Mediante esta técnica es posible reducir el error de cuantización a costa de utilizar dos posiciones de memoria, dos multiplicaciones y dos sumas adicionales.

Sin embargo, una de las formas más sencillas de mejorar el ruido de cuantización es incrementar el número de bits utilizados. Obviamente ello implica mayor utilización de recursos, pero una de las ventajas principales de las FPGA es la flexibilidad que permiten a la hora de ajustar los recursos utilizados a los algoritmos a implementar. Por ejemplo, implementando una estructura biquad en forma directa de tipo 1, hemos visto que para determinados ajustes de un filtro paramétrico se producía ruido de cuantización en implementaciones en coma flotante de 32 bits. Sin embargo, podemos realizar una implementación en coma fija en 48 bits con 20 bits para la parte entera y 28 bits para la parte decimal como punto de partida. Para realizar la simulación de la implementación de diferentes estructuras sobre la FPGA se usa System Generator como herramienta de análisis. Mediante esta herramienta es posible implementar algoritmos DSP utilizando los recursos de la FPGA, y observar y analizar los resultados usando herramientas y flujos de desarrollo típicos de Matlab y Simulink. Una vez los resultados obtenidos son los deseados, es posible exportar el diseño para ser implementado directamente sobre la FPGA.

La implementación de esta estructura adaptada a la FPGA, usando System Generator, se muestra en la siguiente figura:

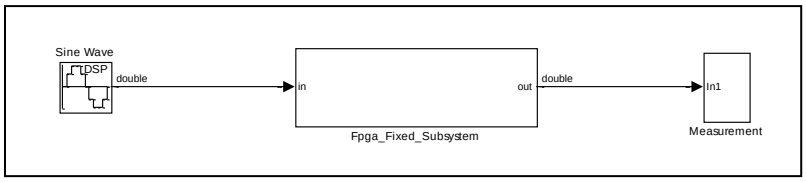

Fig. 17 Modelo System Generator para estudio de estrucutura biquad sobre FPGA

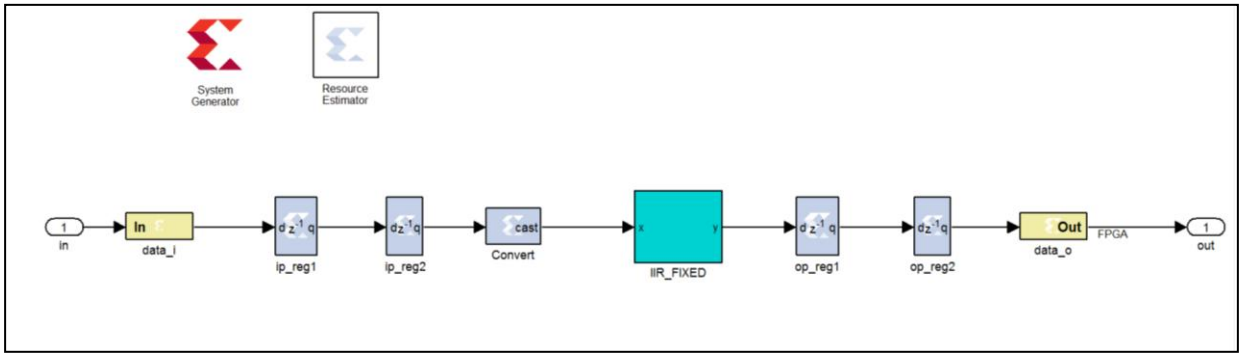

Fig. 18 Subsistema FPGA

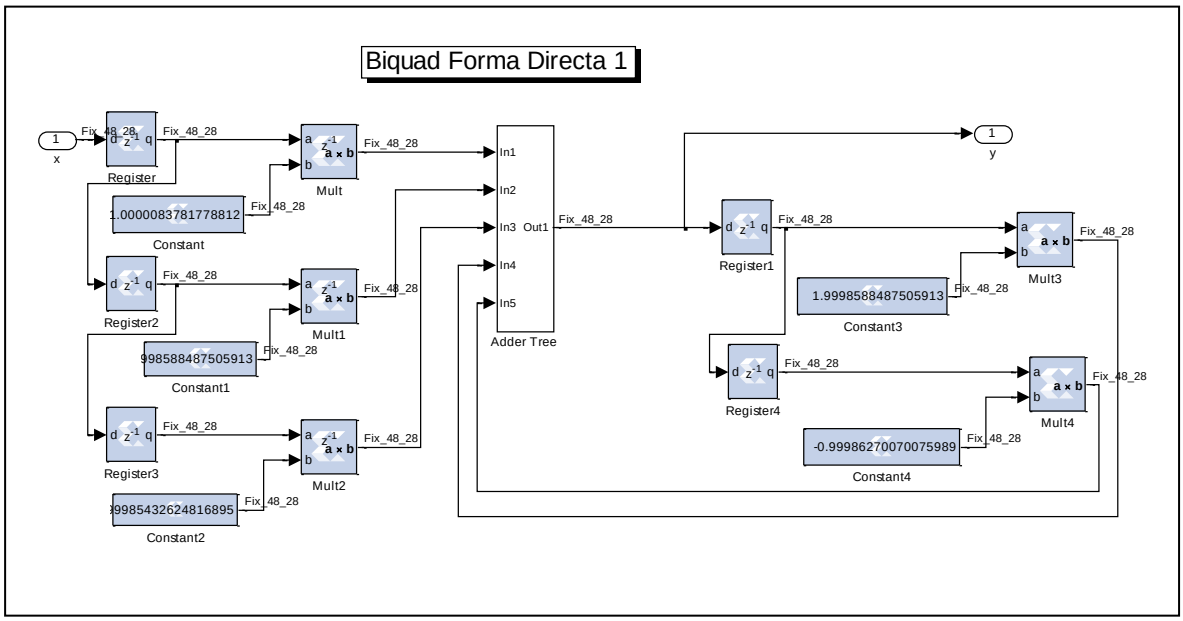

Fig. 19 Implementación biquad sobre FPGA

En la simulación con los mismos parámetros utilizados en coma flotante, se obtiene el siguiente espectro de ruido:

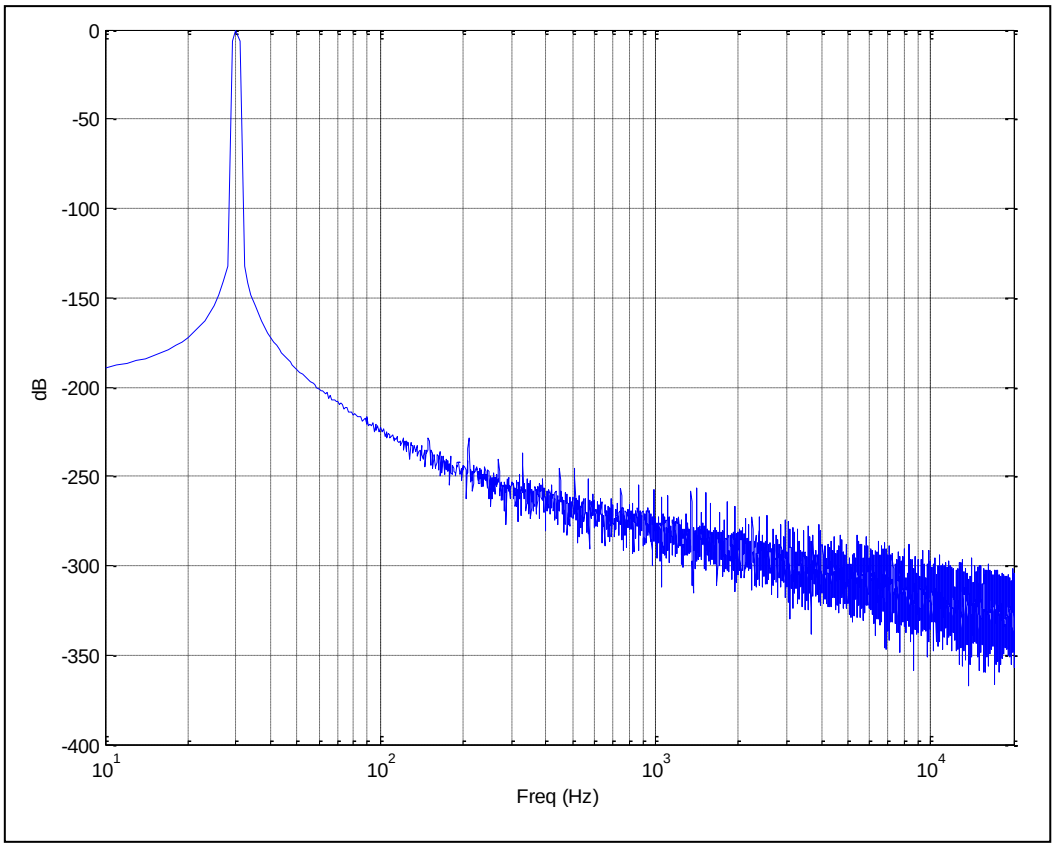

Fig. 20 Ruido para una estructura biquad sobre FPGA de 48 bits con 28 bits decimales en coma fija en FD 1

Esta aproximación ofrece un muy buen comportamiento frente al ruido de cuantización, pero evidentemente, al no hacer ningún tipo de optimización, ni escalado ni estudio sobre el número de bits óptimo, el uso de recursos de la FPGA es alto.

Utilizando células DSP para los multiplicadores, y puertas lógicas para la implementación de sumadores, el diseño necesita los recursos mostrados en la siguiente tabla.

Slices	1536
FFs	777
BRAMs	0
LUTs	2410
IOBs	96
Mults/DSP48s	45
TBUFs	0

Fig. 21 Recursos FPGA necesarios para 48 bits

Vemos que son necesarias 45 células DSP, de las 220 disponibles en el dispositivo FPGA para implementar el algoritmo para una sola sección biquad. Sin tener en cuenta la posibilidad de reutilización de células comentada anteriormente, el uso de recursos es por lo tanto alto.

V.5 OPTIMIZACIÓN Y NÚMERO DE BITS DE COEFICIENTES.

El primer paso para reducir el uso de recursos de la FPGA es optimizar el número de bits utilizados para representar los coeficientes de la función de transferencia. El problema al cuantizar los coeficientes en una estructura con realimentación se produce generalmente en los polos de la función de transferencia. Para un sistema de segundo orden:

$$H(z) = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{8}$$

$$a_1 = -2r\cos(\theta) \tag{9}$$

$$a_2 = r^2 \tag{10}$$

Al cuantizar los coeficientes a_1 y a_2 no estamos cuantizando la parte real y la parte imaginaria de los polos, como sería deseable, sino que se cuantiza la parte real y la distancia de los polos al origen. En el diagrama de ceros y polos, se observa que en el eje *x* la distribución de la cuantización es uniforme, ya que el coeficiente a_1 coincide con la parte real, pero en el eje *y* se cuantiza r^2 en lugar de la parte imaginaria.

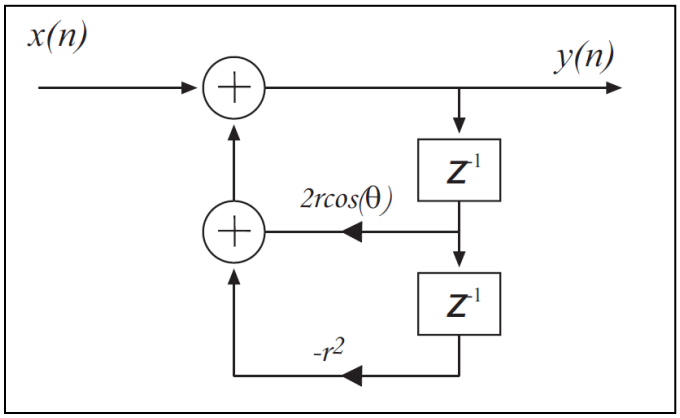

Fig. 22 Polos en una estructura biquad en Forma Directa 1

Este tipo de cuantización produce, para un esquema de 4 bits (1 bit de signo y 3 bits fraccionales) las posiciones cuantizadas de los polos que se muestran en la siguiente figura:

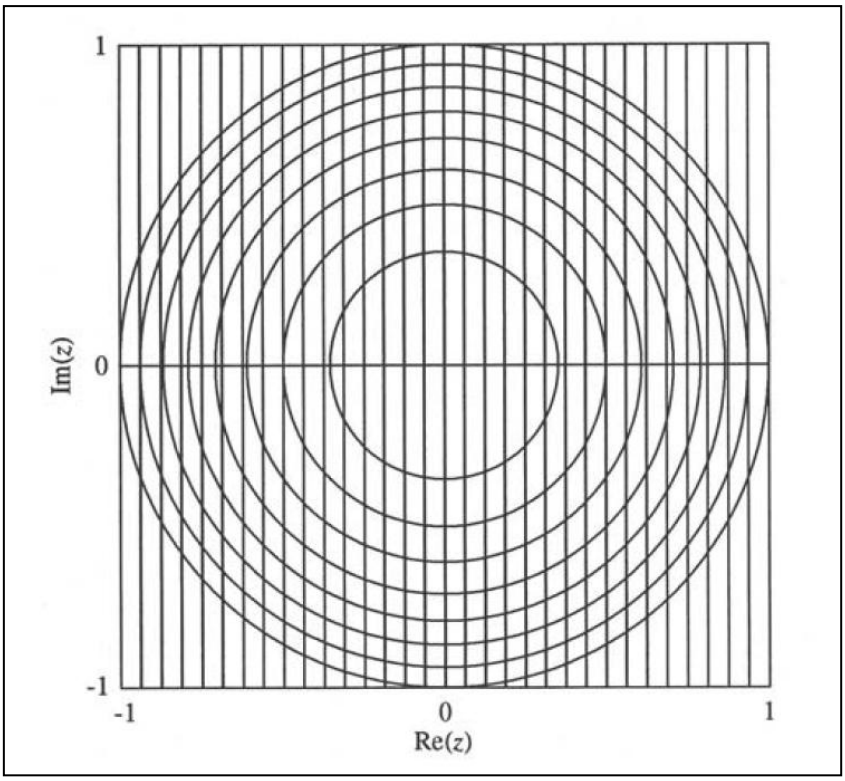

Fig. 23 Posiciones permitidas al cuantizar los polos con 4 bits

Las posiciones permitidas para los coeficientes cuantizados son las intersecciones entre las líneas verticales y las circunferencias. Se puede observar que en las posiciones cercanas a $z = \pm 1$ es donde mayores problemas se producen, ya que la malla de posibles posiciones es muy poco densa. Por otra parte, cuando los polos están muy cercanos a la circunferencia unidad, se pueden producir inestabilidades debidas a la cuantización. Teniendo en cuenta estos factores, podemos diseñar un filtro que presente estas características, como caso más desfavorable para estudiar los efectos de la cuantización de los coeficientes. El siguiente filtro es un filtro paramétrico diseñado con una frecuencia central muy baja con respecto a la frecuencia de muestreo, lo cual hace que los polos se sitúen en el entorno de $z = 1$, y con un factor de calidad alto que produce que tanto polos como ceros se sitúen cerca de la circunferencia unidad. Es una vez más el filtro presentado anteriormente, cuyos parámetros de diseño son:

- Frecuencia de muestreo = 96 KHz
- Fc = 30 Hz
- Q = 14
- G = 0.1 dB

El hecho de utilizar una ganancia muy pequeña, de sólo 0.1 dB hace que los polos y los ceros se posicionen muy cerca entre ellos, de forma que al cuantizar los coeficientes puede ocurrir que los polos y los ceros cuantizados ocupen la misma posición, y se anulen.

En las siguientes figuras podemos ver la respuesta en frecuencia y el diagrama de polos y ceros del filtro diseñado:

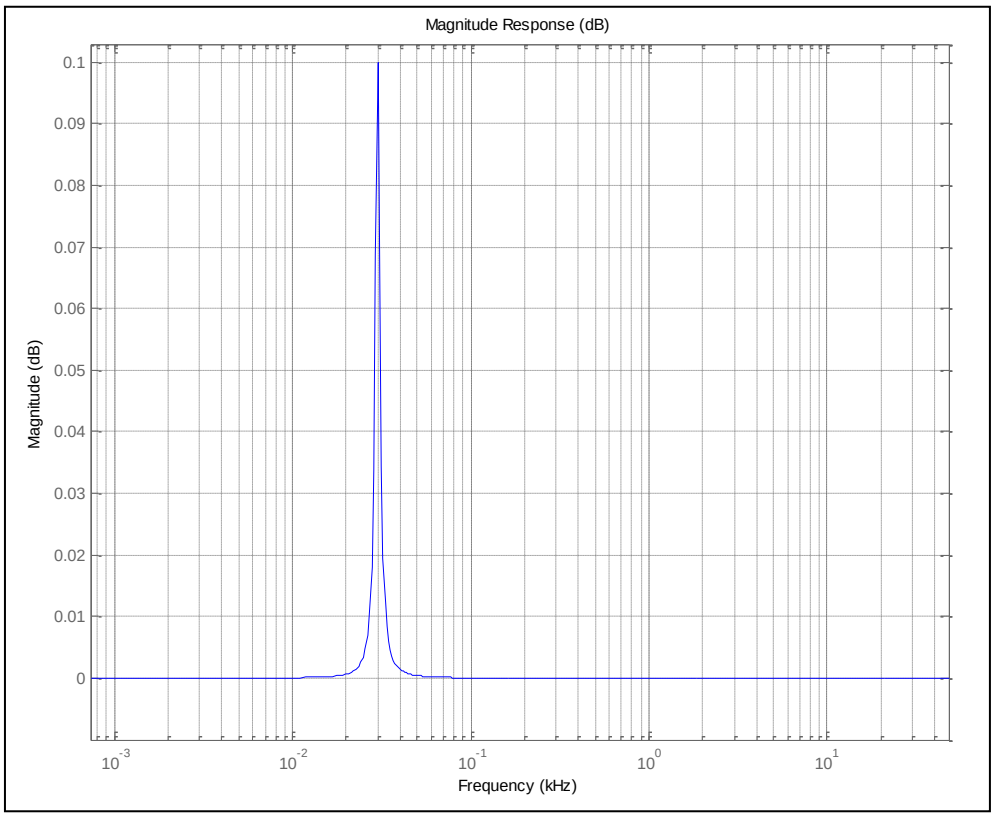

Fig. 24 Módulo de la respuesta en frecuencia del filtro estudiado

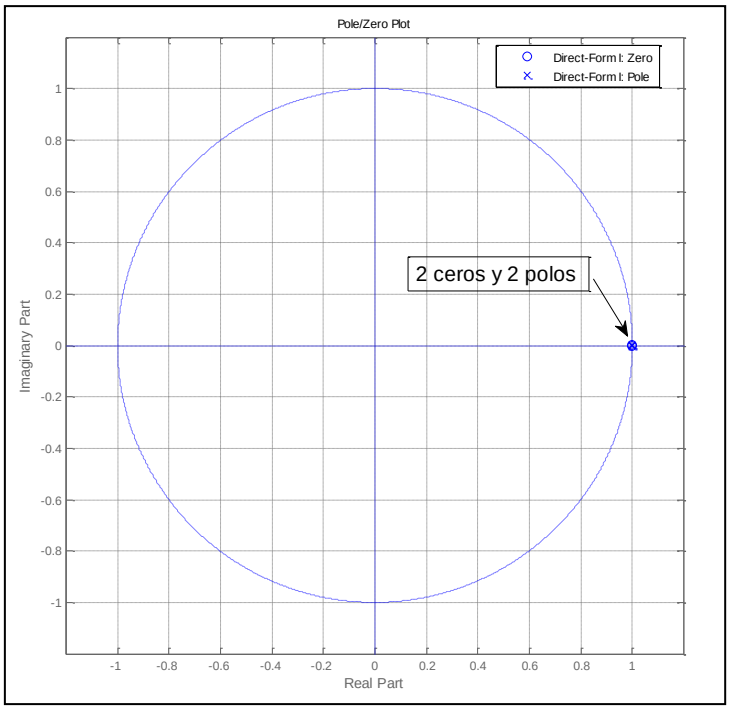

Fig. 25 Diagrama de polos y ceros del filtro estudiado

Vemos que efectivamente los polos y los ceros están muy cerca de la circunferencia unidad, en la zona de $z = 1$ y a la vez muy cerca entre ellos, lo que nos sitúa en el peor caso posible respecto a la cuantización de coeficientes. La siguiente figura muestra un acercamiento a la zona $z = 1$ del diagrama de polos y ceros, en la que se observa la cercanía de polos y ceros a la circunferencia unidad:

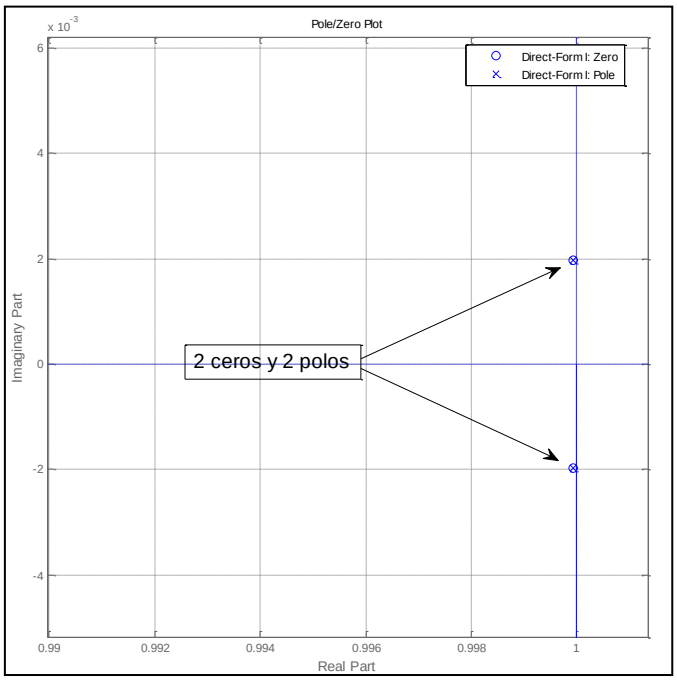

Fig. 26 Polos y ceros muy cercanos a la circunferencia unidad en el filtro estudiado

Haciendo un acercamiento aun mayor a uno de los pares cero-polo, podemos observar también que cero y polo se sitúan muy cerca uno de otro:

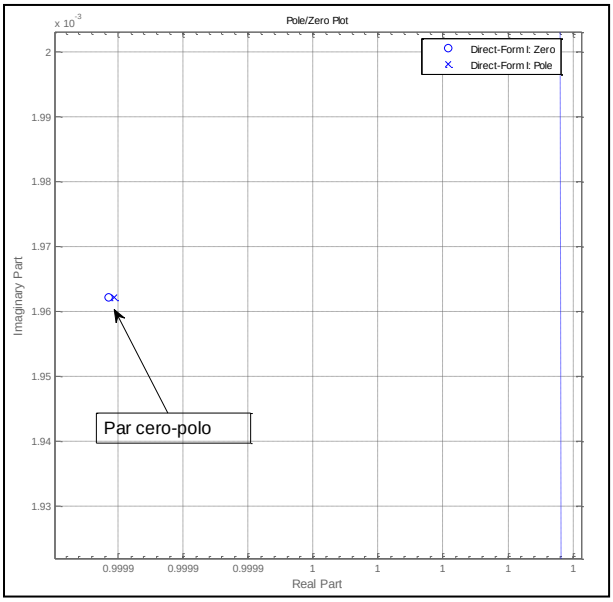

Fig. 27 Par cero-polo del filtro estudiado

Con este filtro como base de estudio para el caso más desfavorable en cuanto a cuantización de coeficientes, vamos a simular diferentes cuantizaciones en coma fija para ver hasta dónde podemos optimizar la estructura presentada anteriormente.

En primer lugar hemos implementado anteriormente el filtro con una cuantización de 48 bits, con 20 bits para la parte entera y 28 bits para la parte decimal. En la siguiente figura se observa que dicha cuantización funciona correctamente. El filtro se mantiene estable y la respuesta en frecuencia es igual a la respuesta en frecuencia del filtro de referencia.

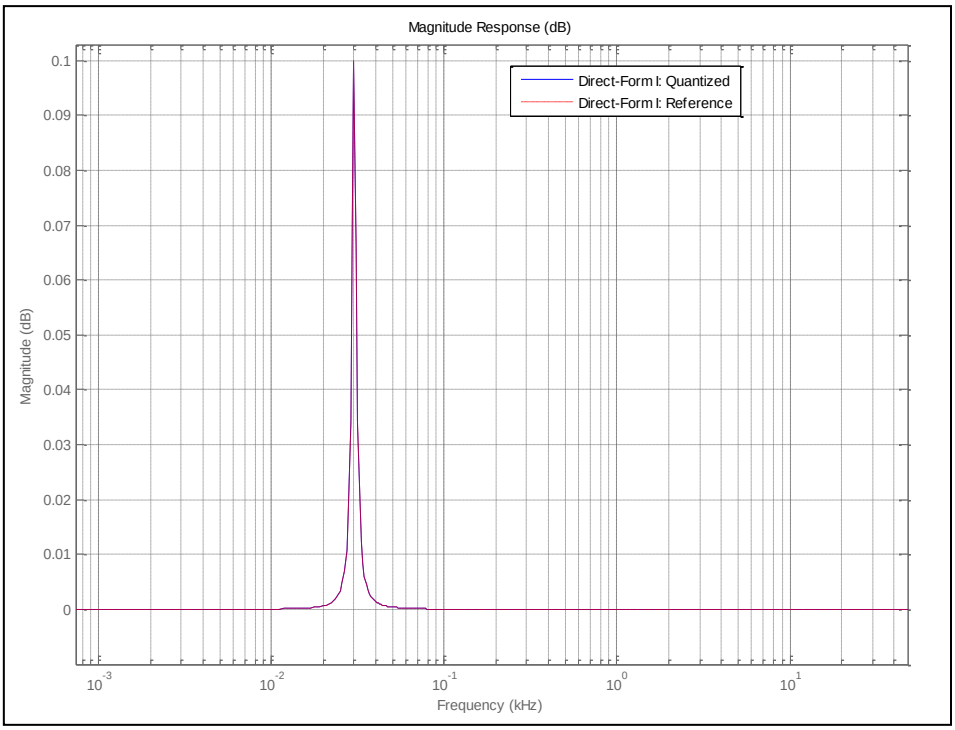

Fig. 28 Módulo de la respuesta en frecuencia del filtro estudiado con cuantización de 48 bits

Teniendo en cuenta que las células DSP de las FPGA que se usarán en la implementación tienen un multiplicador de 18x25 bits, la primera optimización que se puede intentar es la de usar 18 bits para representar los coeficientes. Puesto que los coeficientes están en el rango ±2 utilizamos 16 bits para la parte decimal, 1 bit para la parte entera y un bit de signo.

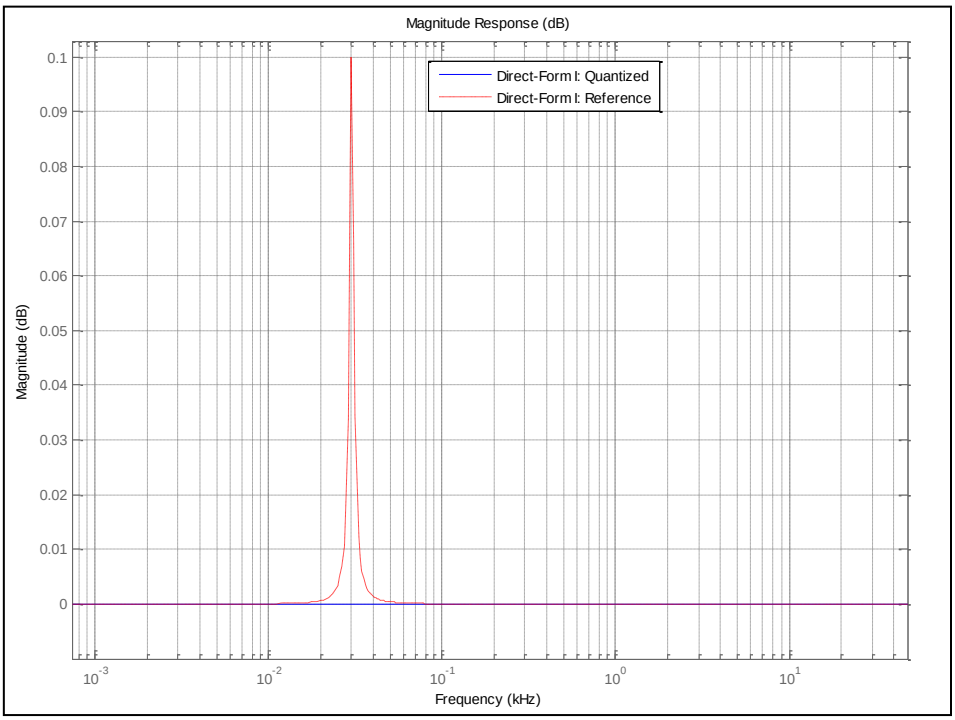

Fig. 29 Módulo de la respuesta en frecuencia del filtro estudiado con cuantización de 18 bits

Vemos que en este caso la cuantización es insuficiente y se produce una cancelación de ceros y polos. Se puede observar que los coeficientes cuantizados producen el mismo numerador y denominador en la función de transferencia, por lo que el filtro deja de funcionar al cuantizarlo.

```
Quantized Numerator:
 1
-1.9998626708984375
 0.9998626708984375
Quantized Denominator:
 1
-1.9998626708984375
 0.9998626708984375

Reference Numerator:
 1.0000007949171275
-1.9998588470946841
 0.99986190722587398
Reference Denominator:
 1
-1.9998588470946841
 0.99986270214300155
```

Fig. 30 Coeficientes de la función de transferencia del filtro cuantizados a 18 bits

En caso necesario los coeficientes del numerador se pueden escalar de forma que ningún coeficiente sea mayor que 1, e introducir el escalado como un término de ganancia, por lo que se puede ganar un bit de resolución, a costa de usar un multiplicador más, usando un bit de signo y 17 bits para la parte decimal. El resultado obtenido se muestra en la siguiente figura:

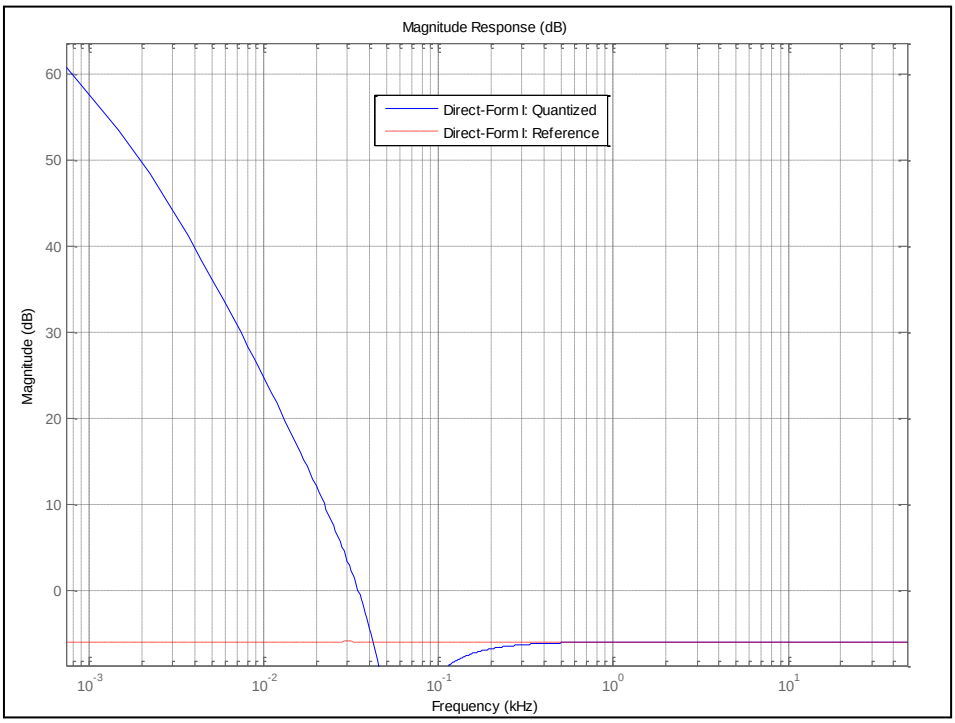

Fig. 31 Módulo de la respuesta en frecuencia del filtro estudiado con cuantización de 18 bits y escalado

El resultado es en este caso incluso peor, puesto que el filtro se hace inestable. Los ceros se cuantizan efectivamente a una posición distinta de los polos, pero uno de los polos se cuantiza sobre la circunferencia de radio unidad, lo que hace que el filtro sea inestable.

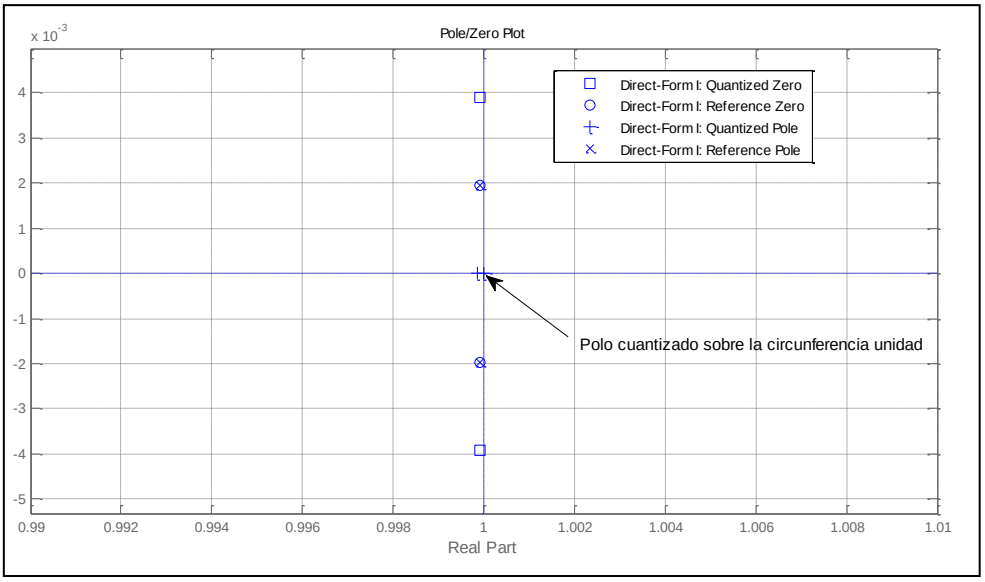

Fig. 32 Diagrama de polos y ceros para cuantización de 18 bits y escalado del numerador

```
Discrete-Time IIR Filter (real)
-------------------------------
Filter Structure     : Direct-Form I
Numerator Length     : 3
Denominator Length   : 3
Stable               : No
Linear Phase         : No
Arithmetic           : fixed
Numerator            : s18,17 -> [-1 1)
Denominator          : s18,16 -> [-2 2)
Input                : s48,28 -> [-524288 524288)
Output               : s48,28 -> [-524288 524288)
Numerator Prod       : s66,45 -> [-1048576 1048576)
Denominator Prod     : s66,44 -> [-2097152 2097152)
Numerator Accum      : s68,45 -> [-4194304 4194304)
Denominator Accum    : s68,44 -> [-8388608 8388608)
Round Mode           : round
Overflow Mode        : saturate

Implementation Cost
Number of Multipliers             : 4
Number of Adders                  : 4
Number of States                  : 4
Multiplications per Input Sample  : 4
Additions per Input Sample        : 4
```

Fig. 33 Características del filtro con cuantización de 18 bits y escalado del numerador

Por lo tanto, puesto que 18 bits no son suficientes para cuantizar las estructuras de audio objeto del estudio, existen dos opciones: o bien adaptar otras estructuras en las que la cuantización proporcione una malla simétrica y más densa para los ejes x e y, o bien incrementar el número de bits. Un ejemplo de estructura que optimiza la cuantización de los coeficientes de un filtro de segundo orden es la estructura acoplada que se muestra en la siguiente figura:

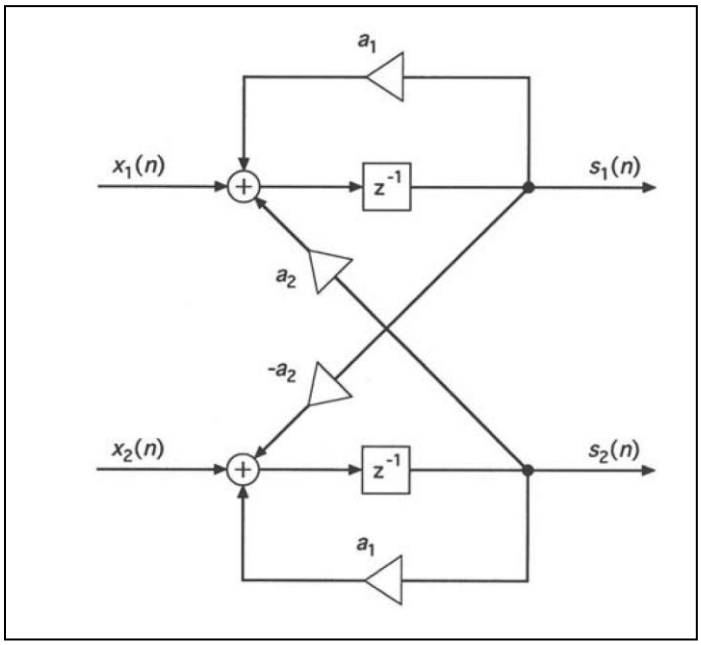

Fig. 34 Estructura acoplada para optimización de la cuantización de coeficientes

En esta estructura se obtiene una malla más densa en las áreas alrededor de $z = \pm 1$ y además la malla es regular para todas las frecuencias. Se consigue una rejilla rectangular a costa de incrementar el número de multiplicadores de dos a cuatro. Las ecuaciones de esta implementación son:

$$x_1(n) = x(n) \tag{11}$$

$$s_2(n) = y(n) \tag{12}$$

$$x_2(n) = 0 \tag{13}$$

$$H(z) = \frac{-a_2^2 z^{-2}}{1 - 2a_1 z^{-1}(a_1^2 + a_2^2) z^{-2}} \tag{14}$$

Si $a_1 = r\cos(\theta)$ y $a_2 = r\sin(\theta)$, los polos de la función de transferencia coinciden con el sistema de segundo orden inicial, pero en esta estructura los coeficientes a_1 y a_2 son la parte real e imaginaria de los polos, por lo que la cuantización da lugar a la rejilla mostrada en la siguiente figura.

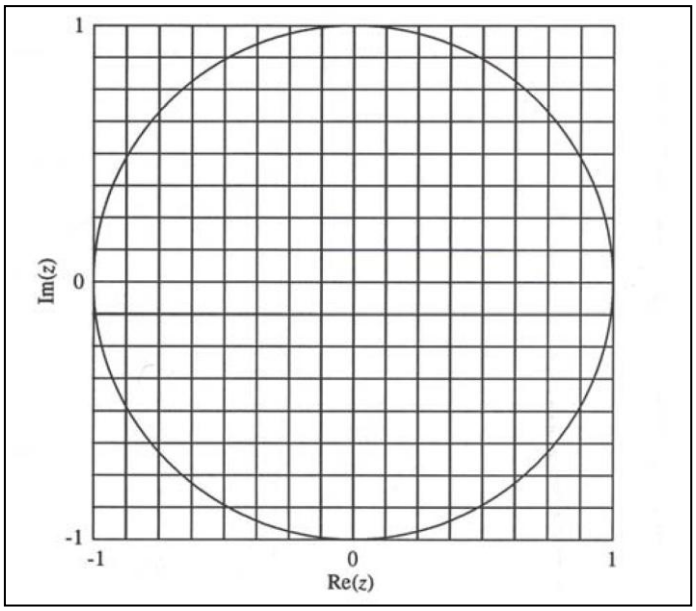

Fig. 35 Posiciones permitidas al cuantizar los coeficientes de la estructura acoplada con 4 bits

Otra opción, si no se quiere incrementar el número de multiplicadores, es aumentar el número de bits. Teniendo en cuenta este filtro en concreto, y sin utilizar escalado de coeficientes, vamos a incrementar el número de bits hasta obtener una respuesta del filtro satisfactoria.

Para una estructura con coeficientes cuantizados a 20 bits se obtiene:

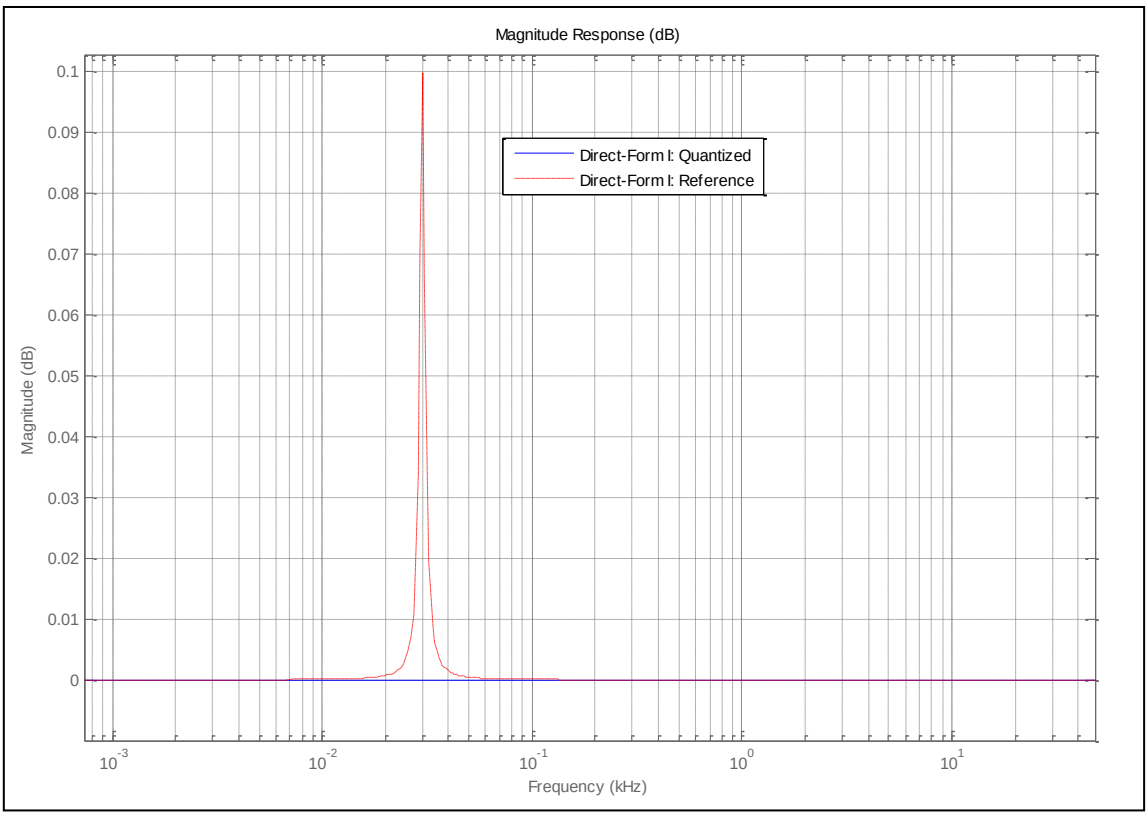

Fig. 36 Módulo de la función de transferencia para el filtro cuantizado a 20 bits

```
Discrete-Time IIR Filter (real)
-------------------------------
Filter Structure     : Direct-Form I
Numerator Length     : 3
Denominator Length   : 3
Stable               : Yes
Linear Phase         : No
Arithmetic           : fixed
Numerator            : s20,18 -> [-2 2)
Denominator          : s20,18 -> [-2 2)
Input                : s48,28 -> [-524288 524288)
Output               : s48,28 -> [-524288 524288)
Numerator Prod       : s68,46 -> [-2097152 2097152)
Denominator Prod     : s68,46 -> [-2097152 2097152)
Numerator Accum      : s70,46 -> [-8388608 8388608)
Denominator Accum    : s70,46 -> [-8388608 8388608)
Round Mode           : round
Overflow Mode        : saturate

Implementation Cost
Number of Multipliers            : 5
Number of Adders                 : 4
Number of States                 : 4
Multiplications per Input Sample : 5
Additions per Input Sample       : 4
```

Fig. 37 Características del filtro para cuantización de coeficientes de 20 bits

El filtro es estable, pero sigue sin tener suficiente resolución, por lo que polos y ceros se anulan al cuantizar.

Para la misma estructura, pero cuantizando los coeficientes a 21 bits:

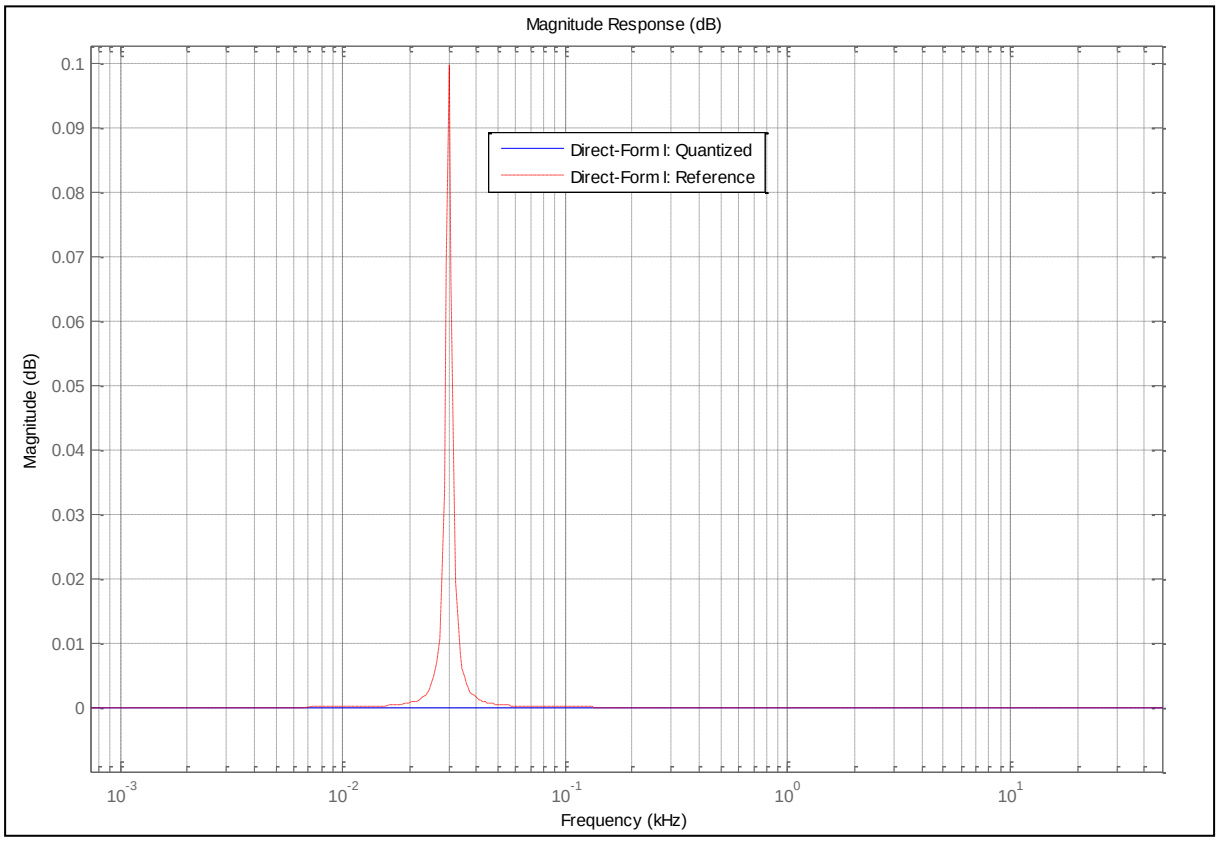

Fig. 38 Módulo de la función de transferencia para el filtro cuantizado a 21 bits

```
Discrete-Time IIR Filter (real)
-------------------------------
Filter Structure     : Direct-Form I
Numerator Length     : 3
Denominator Length   : 3
Stable               : Yes
Linear Phase         : No
Arithmetic           : fixed
Numerator            : s21,19 -> [-2 2)
Denominator          : s21,19 -> [-2 2)
Input                : s48,28 -> [-524288 524288)
Output               : s48,28 -> [-524288 524288)
Numerator Prod       : s69,47 -> [-2097152 2097152)
Denominator Prod     : s69,47 -> [-2097152 2097152)
Numerator Accum      : s71,47 -> [-8388608 8388608)
Denominator Accum    : s71,47 -> [-8388608 8388608)
Round Mode           : round
Overflow Mode        : saturate

Implementation Cost
Number of Multipliers            : 5
Number of Adders                 : 4
Number of States                 : 4
Multiplications per Input Sample : 5
Additions per Input Sample       : 4
```

Fig. 39 Características del filtro para cuantización de coeficientes de 21 bits

El filtro es estable, pero de nuevo la cuantización produce cancelación de ceros y polos.

Incrementando la cuantización de los coeficientes a 22 bits:

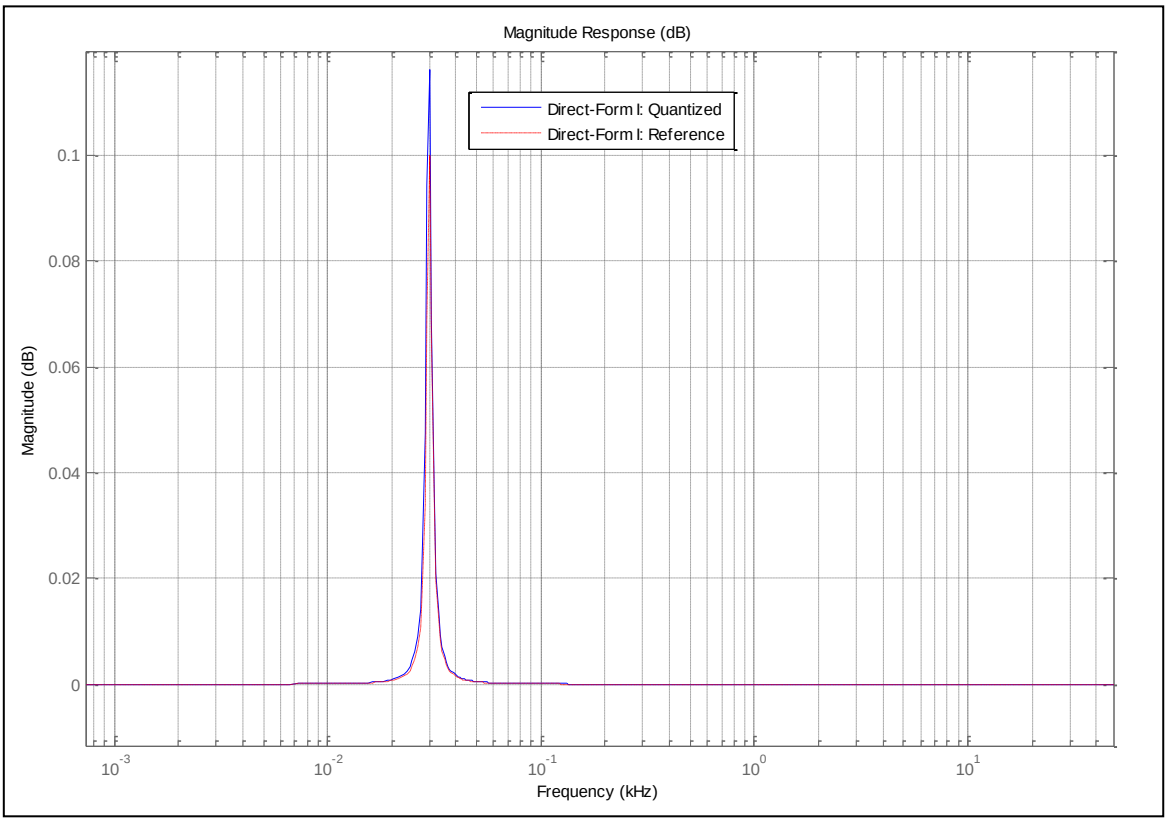

Fig. 40 Módulo de la función de transferencia para el filtro cuantizado a 22 bits

```
Discrete-Time IIR Filter (real)
-------------------------------
Filter Structure    : Direct-Form I
Numerator Length    : 3
Denominator Length  : 3
Stable              : Yes
Linear Phase        : No
Arithmetic          : fixed
Numerator           : s22,20 -> [-2 2)
Denominator         : s22,20 -> [-2 2)
Input               : s48,28 -> [-524288 524288)
Output              : s48,28 -> [-524288 524288)
Numerator Prod      : s70,48 -> [-2097152 2097152)
Denominator Prod    : s70,48 -> [-2097152 2097152)
Numerator Accum     : s72,48 -> [-8388608 8388608)
Denominator Accum   : s72,48 -> [-8388608 8388608)
Round Mode          : round
Overflow Mode       : saturate

Implementation Cost
Number of Multipliers            : 5
Number of Adders                 : 4
Number of States                 : 4
Multiplications per Input Sample : 5
Additions per Input Sample       : 4
```

Fig. 41 Características del filtro para cuantización de coeficientes de 22 bits

En este caso sí se obtiene la respuesta en frecuencia esperada, salvo por un pequeño incremento en la ganancia, por lo que el número mínimo de bits para implementar los coeficientes de este filtro, representativo del peor caso posible, es de 22 bits, con un bit de signo, un bit para la parte entera y 19 bits para la parte fraccional.

Si simulamos de nuevo la implementación en bloques FPGA con System Generator, y actualizamos la longitud de los coeficientes a 22 bits obtenemos el ruido que se muestra en la siguiente figura:

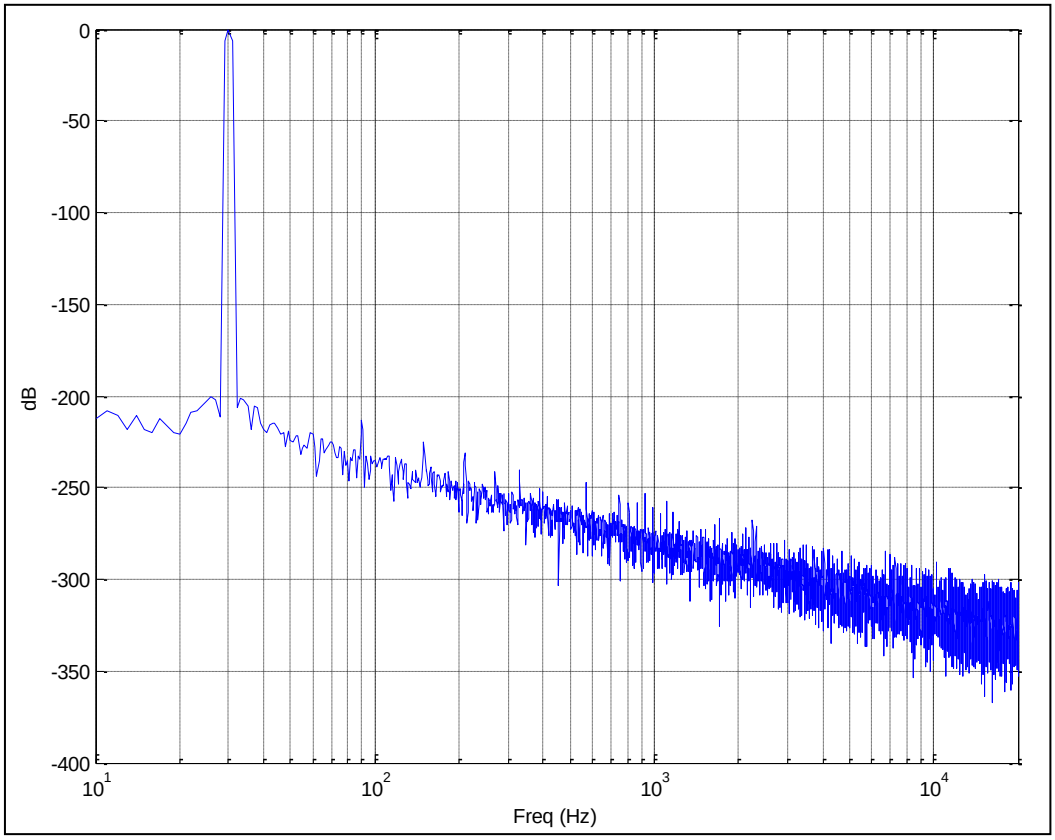

Fig. 42 Ruido para la estructura biquad sobre FPGA con cuantización de coefiecientes a 22 bits.

El ruido conseguido es extremadamente bajo a bajas frecuencias, y hemos conseguido reducir considerablemente los recursos de la FPGA utilizados:

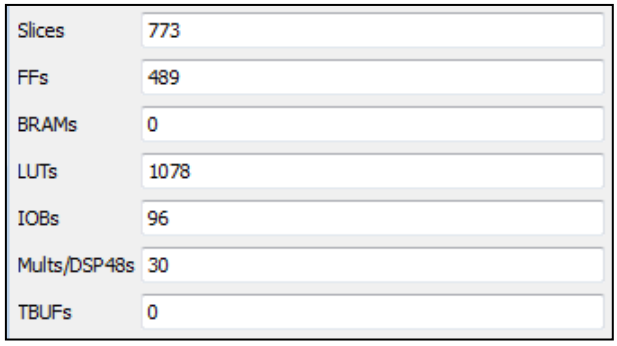

Fig. 43 Recursos necesarios para implementación en FPGA con cuantización de coeficientes a 22 bits

El siguiente paso en la optimización de recursos es adaptar la cuantificación en cada una de las operaciones realizadas en la estructura.

Para adaptar cada una de las operaciones, podemos disminuir el número de bits en los mutiplicadores y sumadores. Sin embargo, hay que tener en cuenta que disminuir demasiado el número de bits en las células DSP puede producir overflow, o bien saturación si se usa aritmética saturada, así como ruido de cuantización.

La siguiente optimización a realizar es adaptar entradas y salidas al número de bits del que se dispone realmente tras la conversión analógico-digital. En el campo del audio profesional, los conversores analógico-digital (ADC) son generalmente de 24 bits. No tiene sentido por lo tanto utilizar más de 24 bits en las entradas del primer filtro, ya que no disponemos de más resolución en la propia señal a filtrar. Con entradas de 24 bits, adaptando convenientemente la ganancia a la entrada para utilizar la máxima resolución del convertidor, y manteniendo multiplicadores y sumadores como en el caso de partida, la estructura queda como se muestra en la siguiente figura:

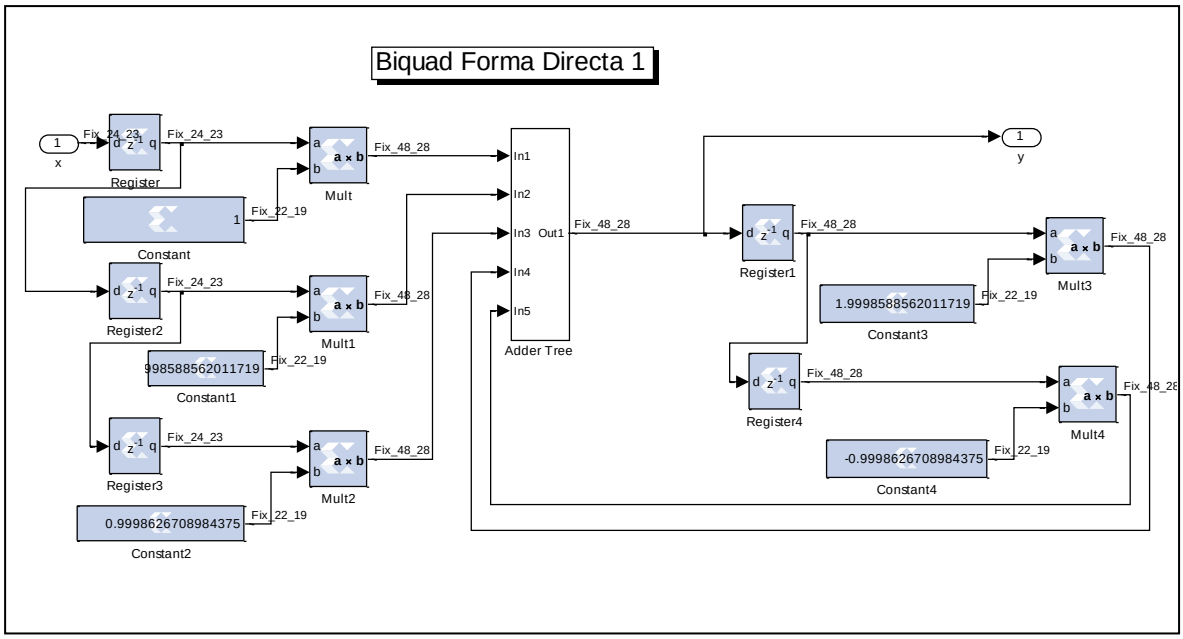

Fig. 44 Implementación biquad sobre FPGA con entrada de 24 bits

Y obtenemos la siguiente tabla de recursos utilizados:

Slices	563
FFs	369
BRAMs	0
LUTs	784
IOBs	72
Mults/DSP48s	24
TBUFs	0

Fig. 45 Recursos FPGA necesarios para entrada de 24 bits

Se ha reducido el número de bloques DSP necesarios, sin embargo el ruido del filtro ha subido hasta prácticamente -100 dB a bajas frecuencias, lo cual no es admisible. Teniendo en cuenta que la relación señal a ruido de cuantización para 24 bits es de 144 dB [11], y que ese es el punto de partida impuesto por el ADC, podemos marcar como un objetivo realista en cuanto a ruido estar por debajo de -120 dB en todo el espectro, lo cual es mucho mejor que el ruido obtenido con coma flotante a 32 bits para este tipo de estructuras, tal y como se ha presentado anteriormente.

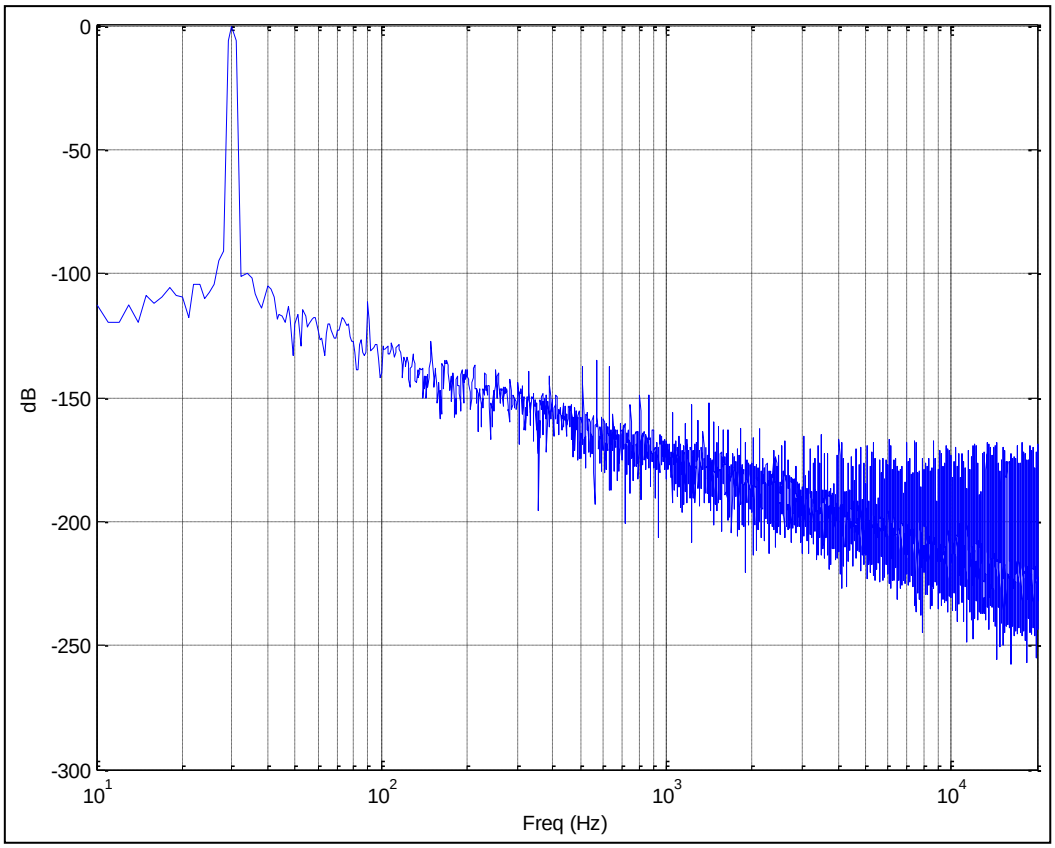

Fig. 46 Ruido estructura biquad con entrada a 24 bits

Estamos utilizando una resolución de partida en multiplicadores y sumadores que no es óptima. El hecho de que estemos obteniendo un ruido alto en bajas frecuencias indica que la resolución no es suficiente. Por otra parte, mediante la simulación sabemos que ninguno de los multiplicadores está saturando, lo cual indica que no son necesarios tantos bits para la parte entera. La siguiente optimización se centrará por lo tanto en adecuar el número y la configuración de bits de multiplicadores y sumadores. En primer lugar, vamos a configurar los sumadores para que se implementen utilizando células DSP en lugar de puertas lógicas. Sabiendo que las células DSP sobre las que se van a implementar los módulos multiplicadores y sumadores, tienen un acumulador de 48 bits, podemos adecuar la configuración para optimizar el número de DSP utilizados. Puesto que necesitamos más resolución, incrementamos la parte decimal y decrementamos la parte entera. Utilizando palabras de 47 bits, con 32 bits para la parte decimal obtenemos:

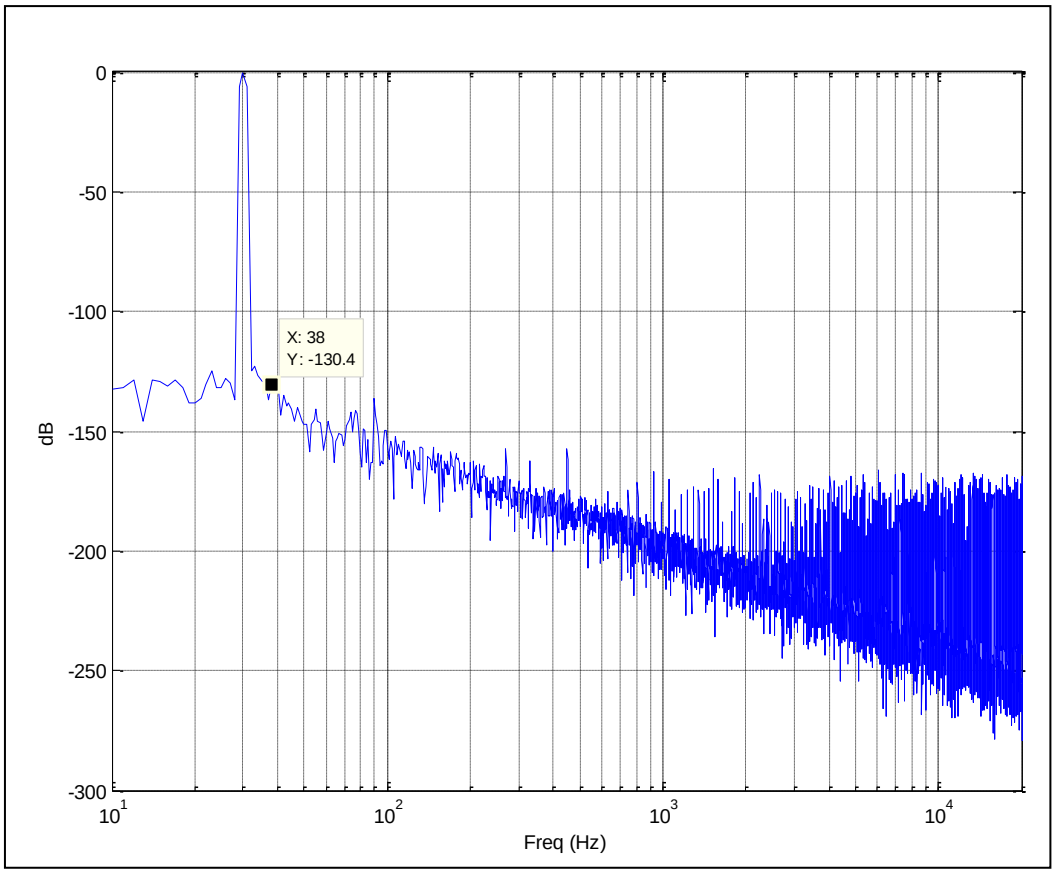

Fig. 47 Ruido estructura biquad para operaciones de 47 bits con 32 bits decimales

El ruido se mantiene por debajo de -130 dB, pero el uso de células DSP se mantiene en 24.

Slices	560
FFs	364
BRAMs	0
LUTs	785
IOBs	71
Mults/DSP48s	24
TBUFs	0

Fig. 48 Recursos FPGA necesarios para operaciones de 47 bits

Disminuyendo el número de bits a 35 y utilizando 31 bits para la parte decimal, obtenemos finalmente una reducción del número de células DSP necesarias a sólo 20, sin embargo el ruido aumenta ligeramente en bajas frecuencias. En cualquier caso, sigue manteniéndose un bajo nivel de ruido, por debajo de -120 dB:

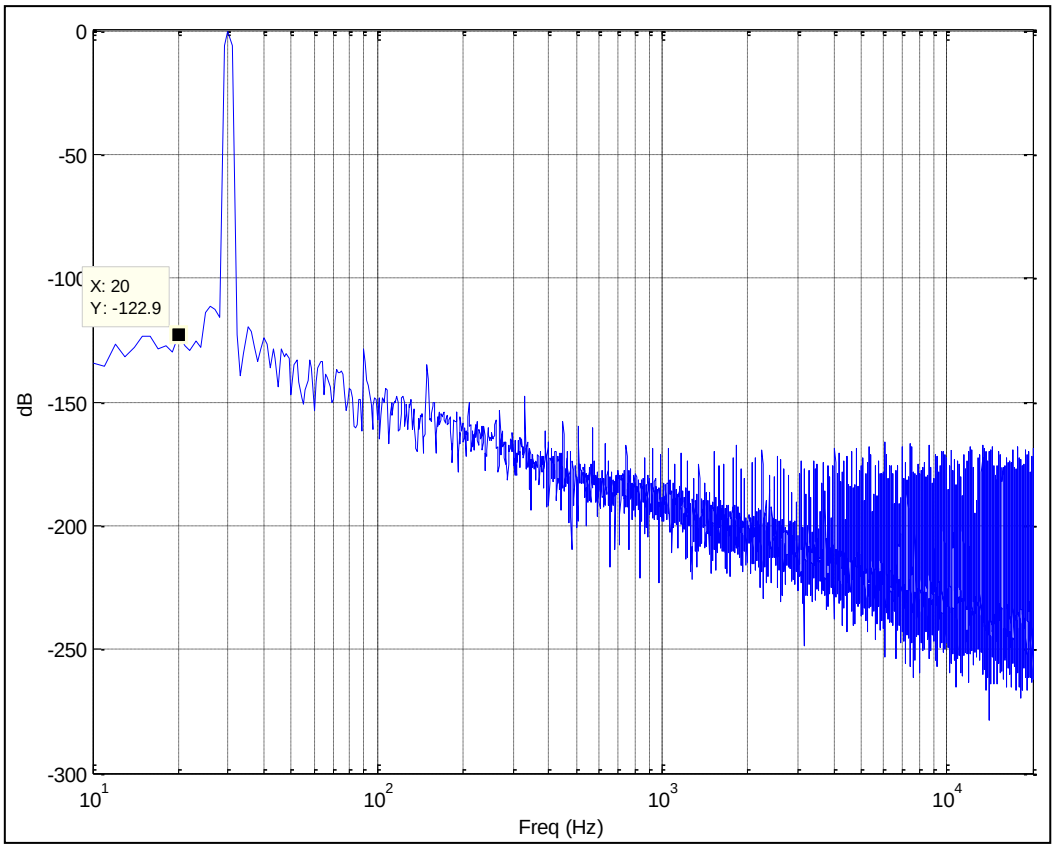

Fig. 49 Ruido estructura biquad para operaciones de 35 bits con 31 bits decimales

Slices	456
FFs	304
BRAMs	0
LUTs	619
IOBs	59
Mults/DSP48s	20
TBUFs	0

Fig. 50 Recursos FPGA necesarios para operaciones de 35 bits

Finalmente la salida del bloque biquad inicial irá encadenada en cascada con la entrada del siguiente bloque biquad. Aunque la resolución de la entrada al primer bloque biquad está definida por el ADC, la entrada de los siguientes bloques biquad puede perfectamente necesitar más rango dinámico, puesto que proviene de la salida de otros bloques biquad, en los cuales es posible que se haya aplicado ganancia a determinadas frecuencias. Podemos por lo tanto utilizar la misma resolución para entradas y salidas que la utilizada para los bloques de suma y multiplicación, es decir 35 bits con 31 bits para la parte decimal. Esta configuración reserva 4 bits para la parte entera, lo que proporciona aproximadamente 24 dB de reserva de ganancia.

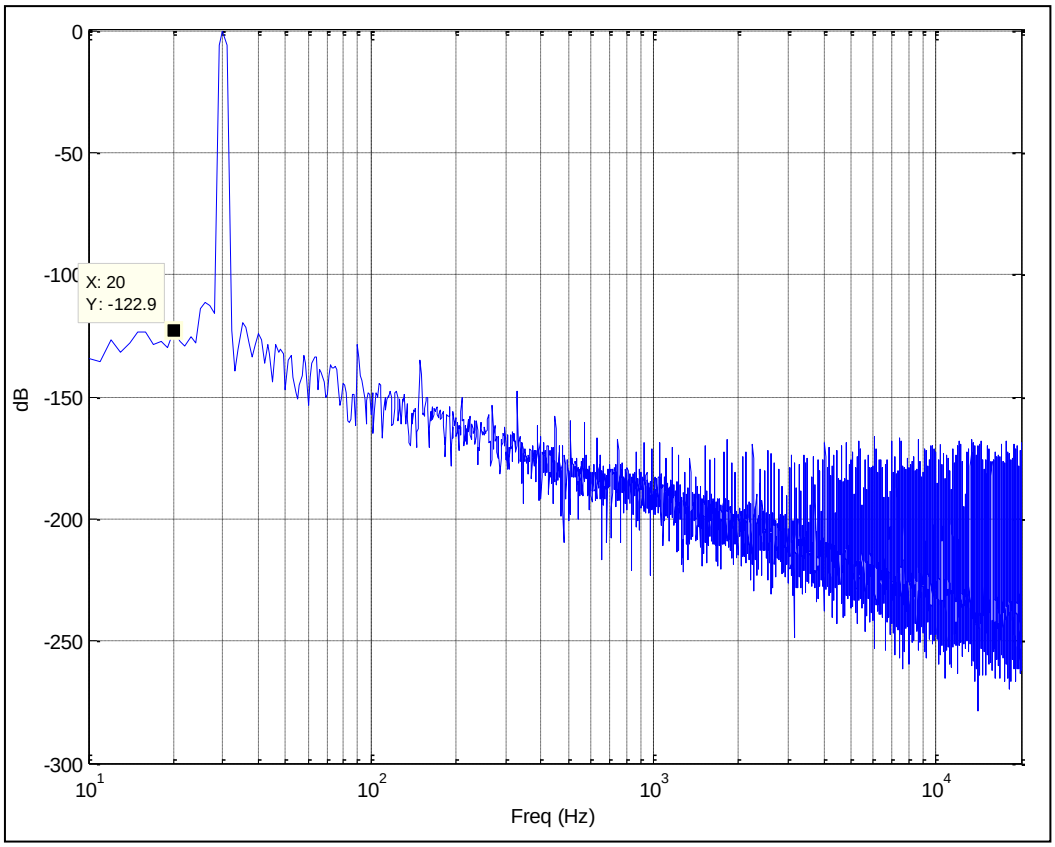

Fig. 51 Ruido estructura biquad para entradas de 35 bits con 31 bits decimales

Y podemos comprobar que los recursos necesarios se mantienen en 20 células DSP por biquad:

Slices	516
FFs	337
BRAMs	0
LUTs	697
IOBs	59
Mults/DSP48s	20
TBUFs	0

Fig. 52 Recursos FPGA necesarios para entradas de 35 bits

Esta será por lo tanto la estructura biquad implementada. Se utilizan 20 células DSP por bloque biquad, con la siguiente configuración de bits:

	Número de bits	N° de bits parte fraccional
Coeficientes	22	19
Datos	35	31
Operaciones internas	35	31

Tabla 1 Formato implementación bloque biquad

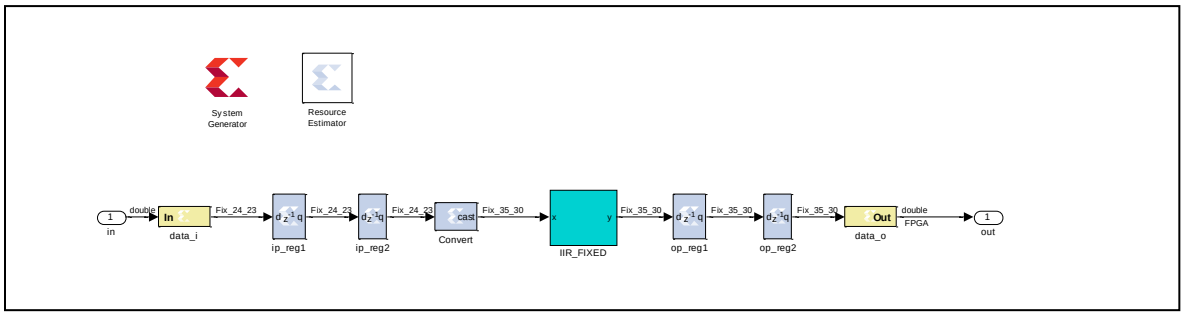

Fig. 53 Implementación final FPGA

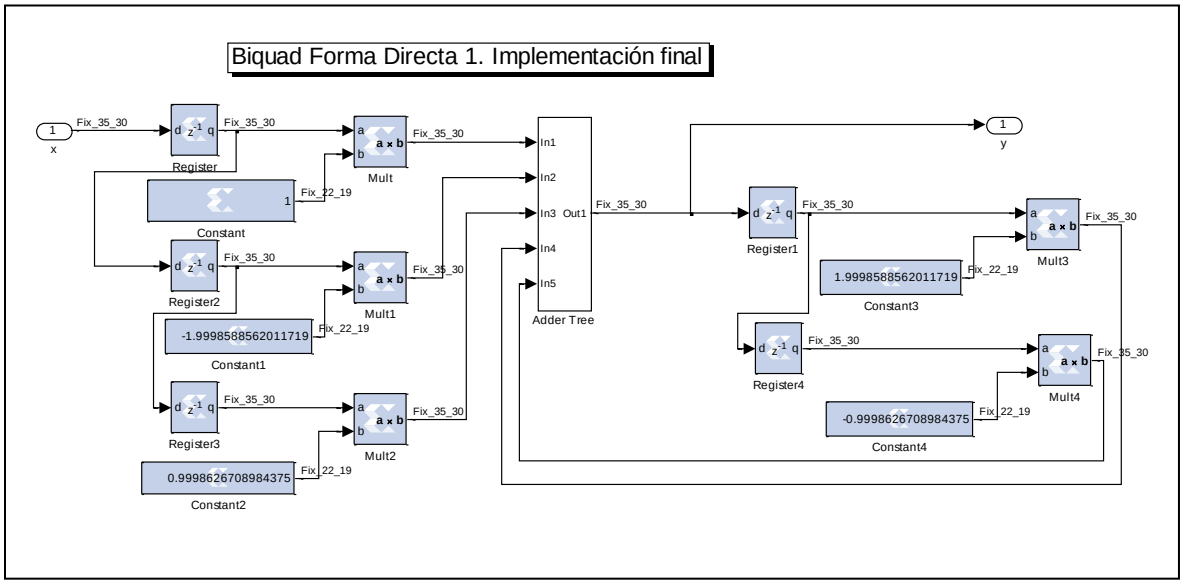

Fig. 54 Implementación final bloque biquad

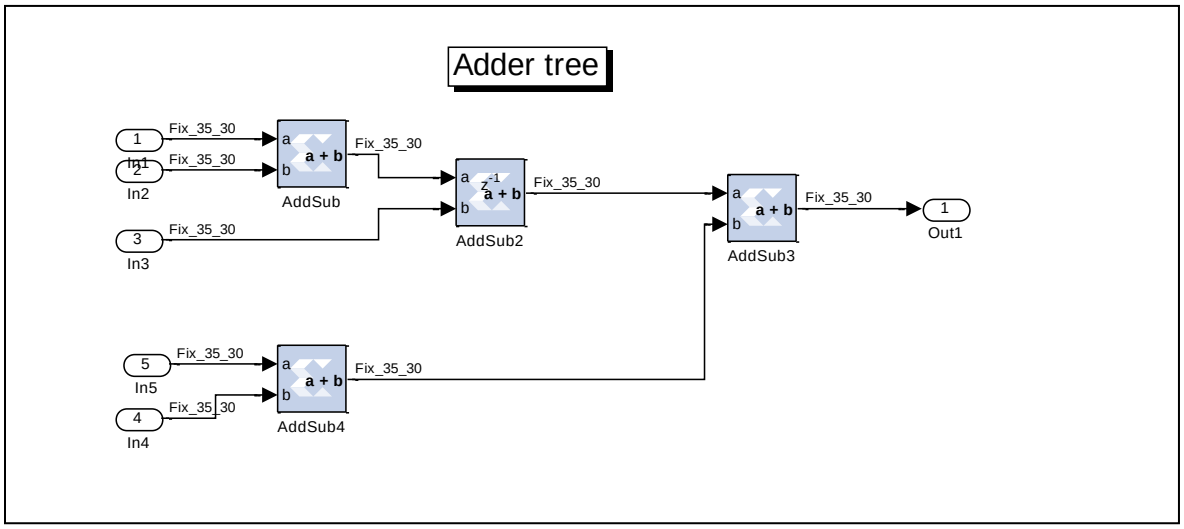

Fig. 55 Implementación final acumulador

Una vez la estructura biquad ha sido optimizada desde el punto de vista del formato del número de bits en cada una de las etapas internas, el siguiente paso en la optimización de recursos de la

FPGA es encontrar la estructura adecuada que implemente un compromiso óptimo entre el número de recursos disponibles en la FPGA, la frecuencia de trabajo de la FPGA, la frecuencia de los datos a procesar y la reutilización de los bloques DSP. En nuestro caso, los datos de audio están muestreados habitualmente a una frecuencia de 48 KHz ó bien 96 KHz. En el caso del dispositivo Zynq-7020, la frecuencia máxima de trabajo teórica de las células DSP es de 741 MHz. Sin embargo, la frecuencia máxima real de cualquier diseño en una FPGA es uno de los parámetros fundamentales, y depende, además de la frecuencia de trabajo del dispositivo, de factores como la complejidad del diseño, el tipo de recursos utilizados, las conexiones entre los distintos bloques, etc… Por ejemplo, a 48 KHz de frecuencia de muestreo de audio, implementando la estructura biquad obtenida en la placa ZedBoard, a una frecuencia de sólo 100 MHz, cada bloque biquad necesita 2 ciclos de reloj, por lo que se podrían reutilizar los bloques DSP mediante la adecuada lógica de control hasta obtener:

$$\text{Num biquads} = \frac{100 \cdot 10^6}{2 \cdot 48000} \approx 1041 \tag{15}$$

Teniendo en cuenta que hemos usado 20 células DSP de las 220 disponibles, el número de biquads que se podría implementar con esta configuración y utilizando todas las células DSP es del orden de 11.000 biquads, sólo con la parte FPGA, quedando aún libre toda la parte de procesador para otros propósitos. A modo de comparación, el procesador Galileo de Meyer Sound [15], de 8 entradas y 16 salidas, e implementado mediante un procesador de propósito general, procesa del orden de 700 biquads.

Como ejemplo de una posible optimización de los recursos de la FPGA teniendo en cuenta la frecuencia de trabajo y el área a utilizar, se puede implementar el mismo bloque biquad pero utilizando un solo multiplicador en lugar de los cinco utilizados en la implementación mostrada. Es decir, se reduce el área utilizada a costa de incrementar la frecuencia necesaria de trabajo. Aunque este tipo de optimización se puede implementar en System Generator, mediante multiplexores y comparadores, tal y como se muestra en la figura Fig. 56, la herramienta adecuada para optimizar los recursos de la FPGA desde el punto de vista del compromiso área utilizada frente a velocidad de proceso, es Vivado HLS de muy reciente aparición. Aunque en este trabajo no se llega a entrar en este tipo de optimización hardware, sí se hace mención en el apartado VII a la herramienta Vivado HLS de forma más detallada.

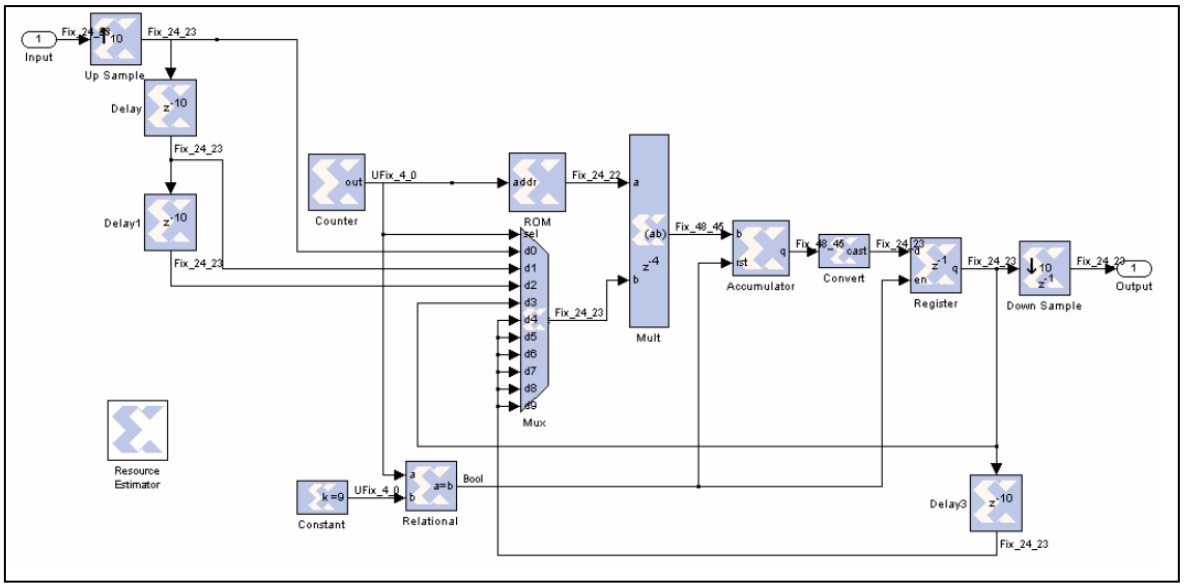

Fig. 56 Implementación del bloque biquad mediante un único multiplicador

VI ESTRUCTURAS DE FILTROS FIR

VI.1 FILTROS FIR EN AUDIO

Los filtros FIR se caracterizan por una función de transferencia en la que no hay polos. La utilización de filtros FIR en audio digital ha sido históricamente minoritaria, debido principalmente a la gran capacidad de proceso necesario para implementar este tipo de filtros, y a la latencia inherente a determinadas implementaciones de este tipo de filtros. Las principales ventajas asociadas tradicionalmente a los filtros FIR, como la posibilidad de diseñar los mismos con fase lineal, no suponen una ventaja importante en audio con respecto a los filtros IIR, teniendo en cuenta la diferencia de proceso necesario que supone implementar un filtro FIR con respecto a un filtro equivalente IIR, por lo que en general han sido poco utilizados.

Sin embargo, en los últimos años, debido en parte al aumento de la capacidad de proceso de los dispositivos hardware, se han empezado a utilizar con más frecuencia en audio digital, principalmente para implementar nuevos algoritmos de procesado con requerimientos especiales.

Es en este tipo de filtros, en los que se requiere una gran capacidad de procesado, donde las FPGA presentan una clara ventaja respecto a los DSP y a los procesadores de propósito general.

VI.2 APLICACIONES FIR PARA AUDIO SOBRE FPGA

Aunque los filtros FIR sí están en general desarrollados e implementados sobre FPGA, y existe amplia literatura al respecto, las especiales características necesarias en audio hacen que no todas las técnicas desarrolladas en otras disciplinas sobre la implementación de filtros FIR sobre FPGA sean válidas en la implementación para filtros de audio. El ejemplo más evidente es la implementación de filtros de fase lineal. Conseguir filtros que no introduzcan distorsión de fase

sólo es posible mediante filtros FIR. El hecho de implementar filtros de fase lineal es muy importante en sistemas de comunicaciones y otras áreas donde se trabaja a alta frecuencia y en bandas estrechas dentro del espectro de radiofrecuencia. En comunicaciones por ejemplo, el hecho de que un filtro introduzca distorsión de fase implica que una señal consistente en una onda cuadrada es deformada de forma importante al pasar por dicho filtro, por lo que se pueden producir errores en la codificación.

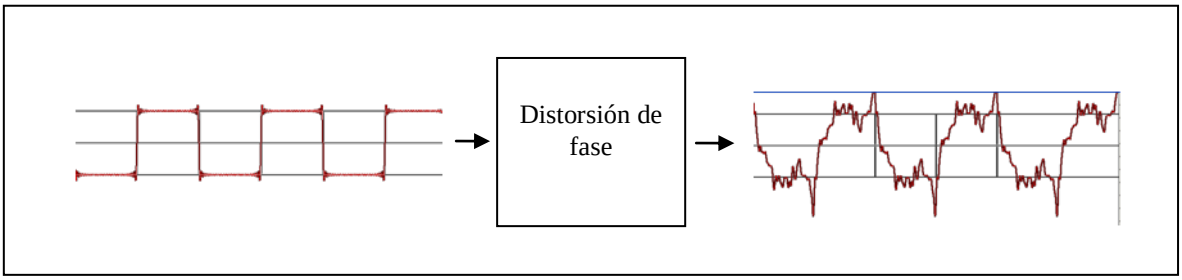

Fig. 57 Deformación de una onda cuadrada por distorsión de fase

En audio, sin embargo, la linealidad de fase es mucho menos importante, entre otras cosas debido a que el procesador digital es sólo un elemento más dentro de una cadena de elementos mucho más amplia, en la cual el principal generador de distorsión es en general el propio altavoz. La audibilidad de la distorsión de fase es también mucho menos importante que la audibilidad de la distorsión de amplitud [16]. Además, el hecho de diseñar un filtro FIR de fase lineal, implica matemáticamente que la respuesta de impulso del filtro debe de ser o bien simétrica o bien antisimétrica [11], lo cual significa que se introducen de forma inevitable latencias importantes en el procesado, generalmente no admisibles en audio, si se quieren diseñar filtros de fase lineal capaces de trabajar a bajas frecuencias.

El hecho de que los filtros de fase lineal no sean tan relevantes en audio como lo son en otras disciplinas no significa que los filtros FIR no tengan cabida en el procesado digital de audio, a pesar de que la característica más notable de los filtros FIR es, precisamente, que pueden ser implementados para que tengan fase lineal. Al contrario, están surgiendo en los últimos tiempos nuevas aplicaciones en las que el uso de filtros FIR es esencial. Un ejemplo claro son los altavoces de columna, en los que es posible controlar digitalmente tanto la inclinación como el ángulo de apertura del haz de sonido para una gran parte del rango de frecuencias de la banda del altavoz. Este control es posible gracias al uso de filtros FIR en los que es posible configurar parámetros de forma muchísimo más flexible que mediante filtros IIR. El precio a pagar es, por supuesto, la altísima capacidad de proceso necesaria para implementar este tipo de algoritmos.

En las siguientes figuras se muestra un ejemplo de aplicación de este tipo de procesado en el altavoz de columna CAL de Meyer Sound Laboratories. En este tipo de altavoces es crítico el control de fase por frecuencia, así como el uso de delays fraccionales, para conseguir control de directividad a cualquier frecuencia. Es en este tipo de requerimientos donde los filtros FIR son capaces de obtener respuestas en fase muy difíciles, o incluso imposibles de conseguir mediante filtros IIR.

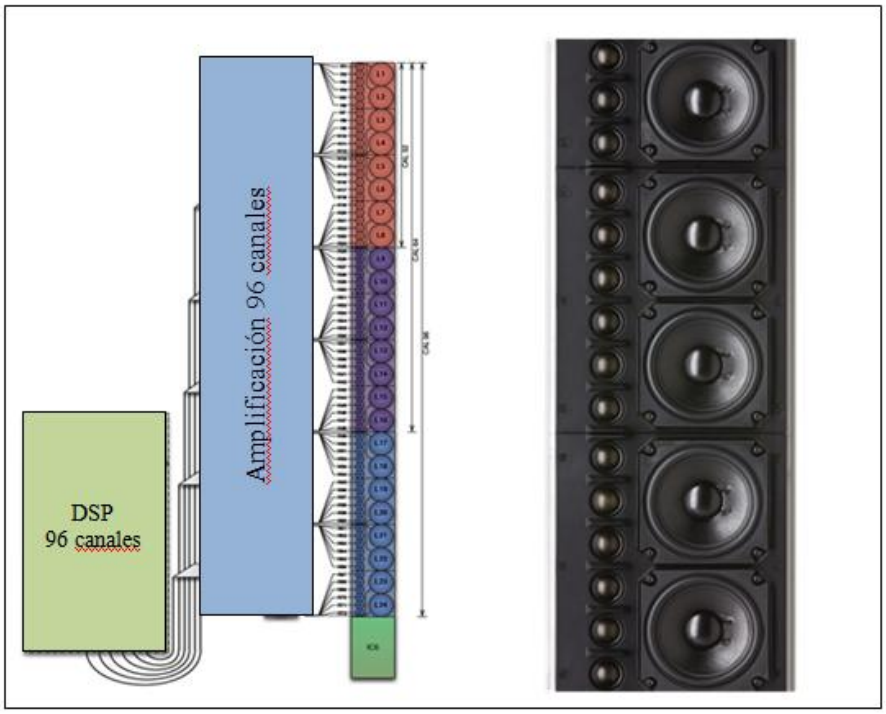

Fig. 58 Esquema de procesado de un altavoz de columna

Fig. 59 Ejemplo de instalación de array de altavoces de columna

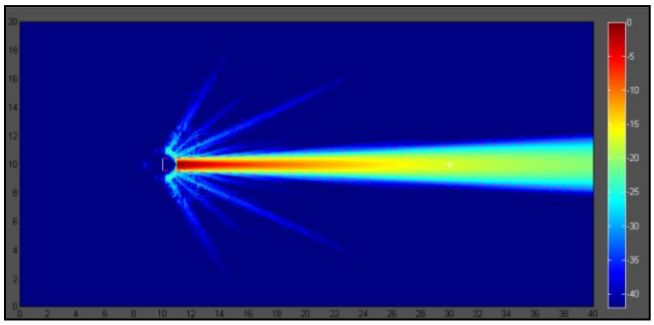

Fig. 60 Array de altavoces de columna. Angulo de apertura 5°. Angulo de inclinación 0°.

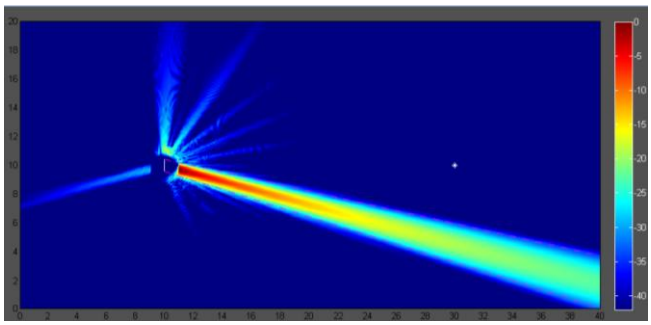

Fig. 61 Array de altavoces de columna. Angulo de apertura 5°. Angulo de inclinación -15°.

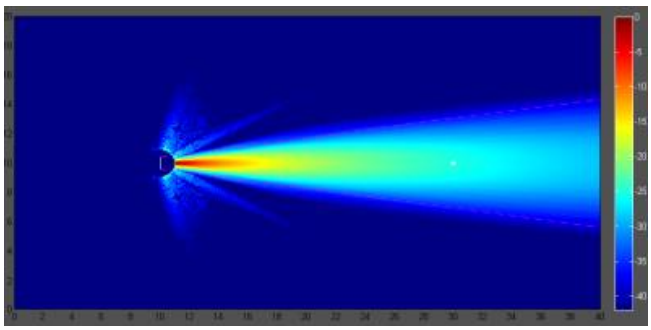

Fig. 62 Array de altavoces de columna. Angulo de apertura 15°. Angulo de inclinación 0°.

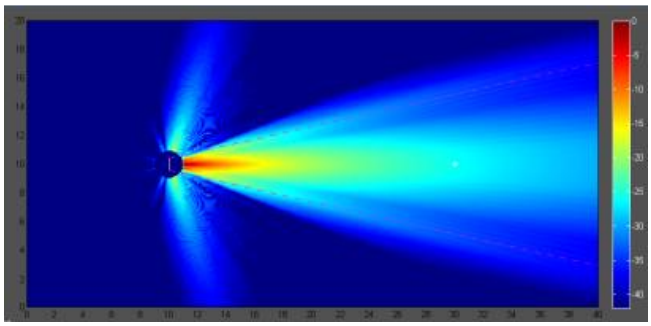

Fig. 63. Array de altavoces de columna. Angulo de apertura 25°. Angulo de inclinación 0°.

VI.3 ESTRUCTURAS DE IMPLEMENTACIÓN FIR SOBRE FPGA

Las estructuras de implementación de filtros FIR sobre FPGA están en general muy estudiadas, ya que se utilizan frecuentemente en otras disciplinas distintas al audio. Las dos estructuras principales en la implementación de filtros FIR sobre FPGA son la forma directa y la forma traspuesta:

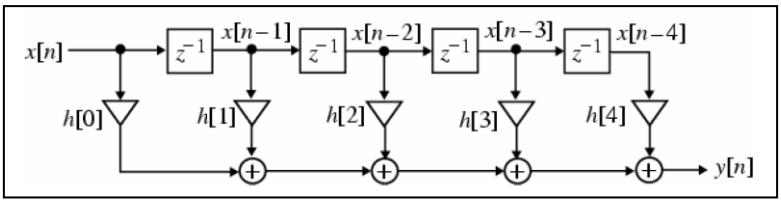

Fig. 64 Estructura FIR en forma directa

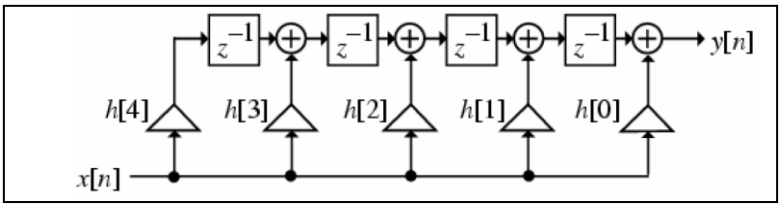

Fig. 65 Estructura FIR en forma traspuesta

En base a estas dos estructuras genéricas, y gracias a la gran flexibilidad que ofrecen las FPGA se pueden realizar multitud de optimizaciones específicas adaptadas a las características de los filtros a implementar. Implementaciones específicas para filtros de fase lineal, implementaciones polifase, descomposiciones en paralelo, estructuras multiplier-less, etc… Debido a que estas estructuras son habitualmente empleadas en FPGA existen disponibles bloques y herramientas de diseño ofrecidos por los propios fabricantes de FPGA, por lo que la implementación de filtros FIR es mucho más sencilla que la implementación de filtros IIR desde el punto de vista de las herramientas de desarrollo. Por otra parte, el hecho de que en los filtros FIR no existan polos, y por tanto no haya realimentación, hace que estos siempre sean estables, facilitando de esta forma también la cuantización de coeficientes.

A modo de ejemplo, la siguiente implementación de un filtro FIR para FPGA, realizada con System Generator, implementa un filtro FIR, en el que además se busca obtener un delay fraccional, es decir, un delay que no es múltiplo entero del período de muestreo. Este es el tipo de filtro que se usa para obtener la inclinación del haz en un array de columna como los presentados anteriormente.

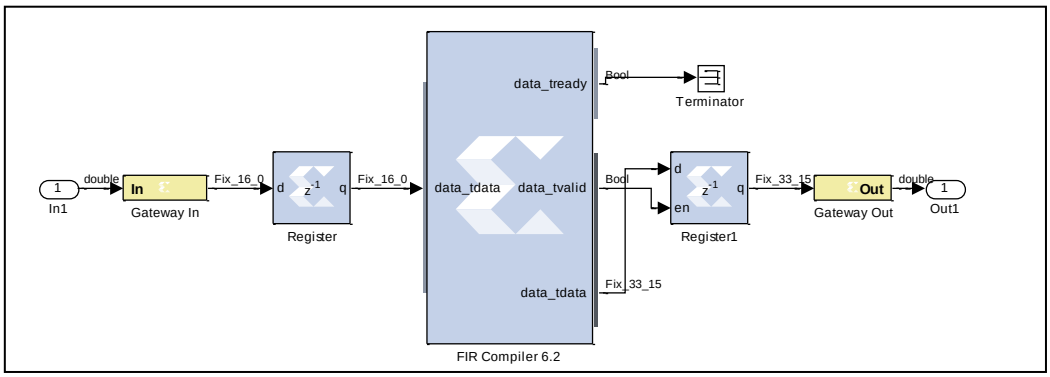

Fig. 66 Modelo de filtro FIR para implementación en FPGA con System Generator

En este tipo de filtro, el objetivo no es obtener una respuesta en amplitud determinada, que debe de ser plana, el objetivo es obtener el retardo de fase y comprobar que no es un número entero de

muestras, si no que es fraccional. Mediante retardos progresivos aplicados a cada altavoz en la columna se obtiene la angulación deseada[17]. Sin embargo estos retardos, son generalmente retardos que no coinciden con un múltiplo entero del periodo de muestreo, por lo que son necesarios delays fraccionales.

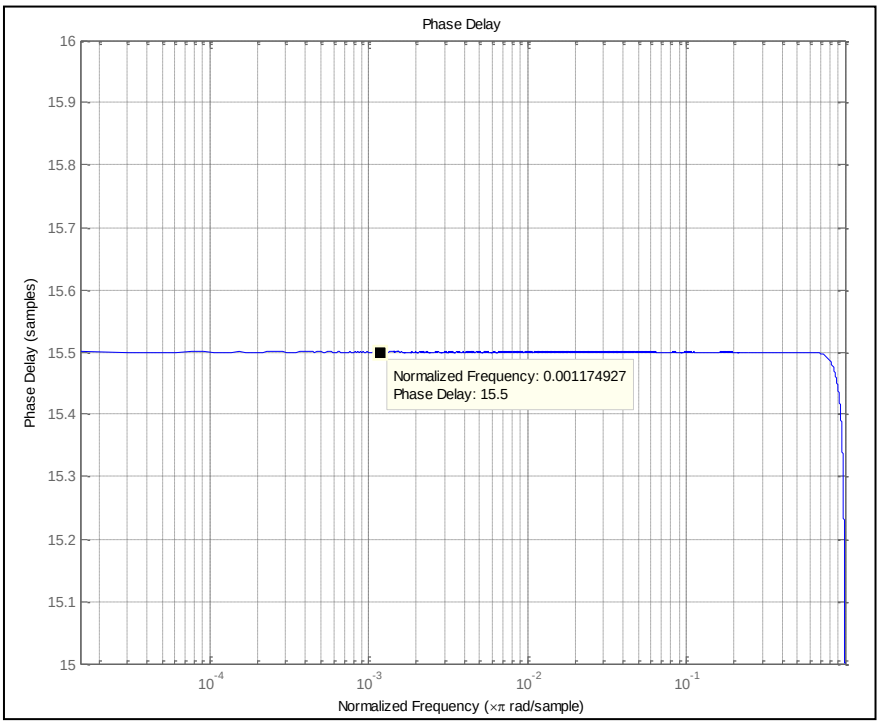

Fig. 67 Retraso de fase en función de la frecuencia para el filtro FIR de delay fraccional

En la figura anterior se observa como el retardo de fase es constante para todas las frecuencias e igual a 15.5 muestras. El filtro FIR, de orden 30, ha sido implementado mediante el bloque FIR proporcionado por Xilinx, y necesita 31 células DSP para esta implementación en concreto, siempre que no se reutilicen recursos, lo cual supone un paso adicional de optimización.

VII IMPLEMENTACIÓN EN FPGA

VII.1 PLATAFORMA DE DESARROLLO

Para implementar los bloques de procesado presentados y poder estudiar las características específicas de cada uno de ellos, así como estudiar los puntos específicos en los que se puede profundizar en investigación de posibles mejoras, se va a utilizar la plataforma ZedBoard, disponible desde hace unos meses en el mercado.

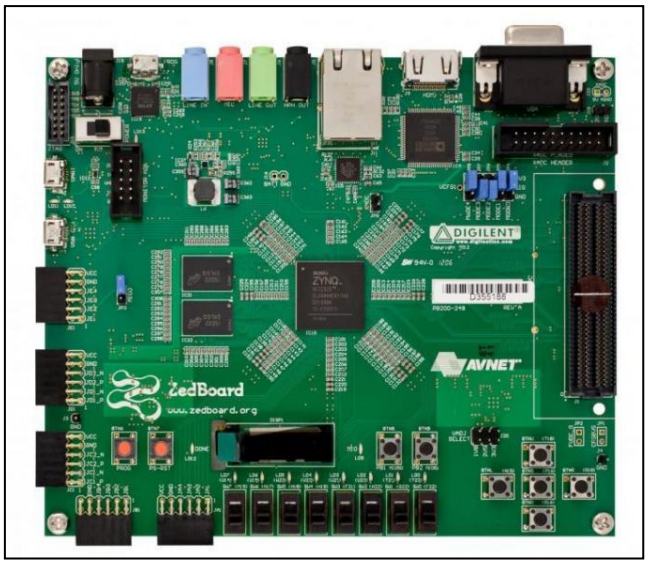

Fig. 68 Plataforma de desarrollo ZedBoard

Esta plataforma de bajo coste está basada en el dispositivo de Xilinx Zynq-7020. Este dispositivo incluye en el mismo chip un procesador ARM Cortex-A9 de deoble núcleo y una FPGA Artix que incluye 220 bloques específicos de procesado DSP. Este tipo de chips está orientado a utilizar la FPGA como un acelerador hardware de la parte del procesador ARM. La parte de procesador se encarga de ejecutar software, y dado el incremento continuo en la capacidad de proceso de los dispositivos ARM, este dispositivo está orientado al uso de sistemas operativos para la parte software. En este tipo de soluciones orientadas al audio, se utilizan generalmente sistemas Linux, sistemas Linux optimizados para funcionar en tiempo real (RTLinux), o si se desea un control total sobre el tiempo de ejecución de determinadas partes del software, se pueden usar también sistemas operativos en tiempo real, específicamente diseñados para este fin (rtOS). Mientras que la parte del procesador se encarga del interface con el usuario, tareas de comunicaciones, cálculo de coeficientes, etc… la parte de FPGA se utiliza como un periférico acelerador de hardware. Es decir, se optimiza el diseño de la FPGA en función del algoritmo que se quiere implementar, y se acopla al sistema procesador como un periférico más.

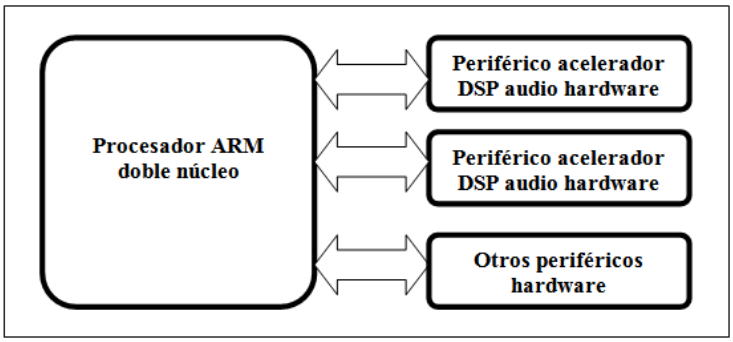

Fig. 69 Orientación del procesado como un periférico hardware

Por ejemplo, se puede implementar en la FPGA un algoritmo que calcule un filtro FIR de 256 coeficientes en unos pocos ciclos de reloj, mediante la paralelización de los cálculos, y realizar

llamadas al periférico implementado desde el procesador. Este trabajo está centrado en la parte FPGA del chip, y en la implementación de periféricos específicos de aceleración hardware, que es el objeto de esta tesis, y no entramos para nada en la parte software y de procesador ARM. La placa de desarrollo incluye, además, entradas y salidas de audio, que se utilizarán para analizar y medir el rendimiento real de cada uno de los bloques.

VII.2 HERRAMIENTAS DE DESARROLLO

Existen diferentes herramientas de desarrollo para implementar algoritmos DSP sobre FPGA. Las más relevantes en el caso de Xilinx son *System Generator* y *Vivado HLS*.

System Generator es la herramienta que se ha usado para generar las implementaciones en este trabajo. Está basada en descripción de bloques en el entorno Matlab y Simulink, y está orientada a la optimización de los algoritmos de procesado en términos de una implementación eficiente, tal y como se ha mostrado en este trabajo.

Vivado HLS es una herramienta de síntesis HDL basada en descripción de algoritmos en lenguaje C. Ofrece herramientas específicas para optimización del procesado sobre FPGA. Vivado HLS es especialmente útil para explorar diferentes implementaciones sobre la FPGA, en cuanto a optimización de recursos. Como se ha comentado anteriormente respecto a la implementación de estructuras biquad, una vez la estructura óptima del algoritmo ha sido definida, hay que buscar un compromiso entre velocidad de procesado y cantidad de recursos utilizados por cada algoritmo, adecuando para ello los loops, los ciclos de reloj utilizados en cada estructura usada para implementar el algoritmo final, el número de operaciones que cada célula DSP debe realizar, la latencia total del algoritmo, etc... La implementación final es un compromiso entre área utilizada y velocidad de proceso, y el resultado final depende de muchos factores. Vivado HLS permite ensayar diferentes implementaciones desde el lenguaje C, sin necesidad de implementar el diseño en la FPGA, lo cual necesita, como ya se ha comentado, tiempos muy largos de compilación de incluso horas. Por lo tanto es posible ensayar diferentes soluciones para un algoritmo dado, de forma que se optimicen los requerimientos de velocidad de proceso, capacidad de cálculo y los recursos disponibles en la FPGA. Aunque en este trabajo sólo se ha desarrollado la parte de System Generator, las dos herramientas son complementarias a la hora de la implementación final de los procesadores sobre una FPGA.

Para la implementación biquad en coma fija, se puede implementar cualquiera de las estructuras presentadas anteriormente, teniendo en cuenta las características de las células DSP de la FPGA. En el caso de Xilinx, la célula DSP incluida en la serie 7, y también en el dispositivo Zynq, es la célula DSP48E1 presentada en el capítulo II.

Como ya se ha comentado, el dispositivo implementado en la placa de evaluación dispone de 220 células DSP48E1, y mediante una estimación aproximada, a una frecuencia relativamente baja, y sin optimizar mediante Vivado HLS la utilización adecuada de recursos, es posible implementar con este dispositivo del orden de 11.000 biquads. Es importante destacar que el dispositivo presentado es de los más pequeños dentro de la familia 7 de Xilinx. Existen dentro de la misma familia dispositivos con hasta 2.020 células DSP, así como dispositivos compatibles, sin la parte de procesador, con hasta 3.600 células DSP. Se puede por lo tanto optimizar el uso de recursos DSP adaptando convenientemente el procesado a realizar por cada célula DSP, así como el dispositivo a utilizar, obteniendo capacidades de procesado muy superiores a los DSP tradicionales.

VIII CONCLUSIONES Y TRABAJOS FUTUROS

VIII.1 CONCLUSIONES

En este trabajo se han presentado varias optimizaciones para implementar procesado de audio sobre dispositivos FPGA. Especialmente se ha presentado una estructura para filtros IIR y una optimización en el número de bits a utilizar, tanto en la cuantización de los datos como en el tamaño de los coeficientes, de forma que el nivel del ruido se mantenga por debajo de -120 dB, manteniendo los recursos a utilizar en la FPGA en los mínimos posibles.

Se ha utilizado también una plataforma de desarrollo orientada a la investigación en la implementación de estructuras de audio para procesado DSP sobre FPGA, que servirá como base para trabajos futuros en la enorme cantidad de estructuras y diferentes implementaciones que aún existen por estudiar relacionadas con el procesado de audio sobre FPGA, no solo en cuanto a filtros, si no también estudios orientados a análisis FFT, mezcladores, procesado temporal, procesadores no lineales, etc…

VIII.2 TRABAJOS FUTUROS

Tal y como se ha comentado a lo largo del presente trabajo, la implementación de procesado digital de audio basado en FPGA, es un campo con un futuro claramente en crecimiento, aunque sigue siendo un campo con escasa investigación aplicada. Las estructuras presentadas en este trabajo, aunque son estructuras básicas ampliamente utilizadas, representan una parte muy pequeña en lo que respecta al procesado de audio, por lo que existe una enorme cantidad de investigaciones a realizar referentes a diferentes campos dentro del procesado de audio. Una plataforma como la presentada en este trabajo es esencial para realizar investigaciones sobre, entre otras:

- Nuevas estructuras de filtros IIR para optimizar el ruido de cuantización.
- Algoritmos de análisis basados en FFT de varios canales.

- Algoritmos de convolución rápida para filtros FIR.
- Procesado no lineal para audio: compresores, puertas de ruido, algoritmos de optimización de respuesta de altavoces, etc…
- Bloques básicos de mezcla, delay y rutado.

Así mismo, la investigación presentada en este trabajo representa sólo una parte del proceso necesario para optimizar e implementar un procesador de audio sobre FPGA. Es necesario desarrollar la optimización de recursos en función de la frecuencia de trabajo disponible y del área a utilizar, así como la parte software a implementar en la parte del procesador, que se realiza generalmente bajo un entorno Linux, y que se encarga de la interface con el usuario, comunicaciones, cálculo de coeficientes, etc…

Agradecimientos

A mis directores por su paciencia, ayuda, ideas y colaboración, que espero no termine con la presentación de este trabajo.

A mi familia que entiende, sufre y soporta mi afición por los filtros, las FPGA, la programación, el audio y tantas otras cosas.

Referencias

[1] http://www.meyersound.com/products/industrialseries/cal/

[2] http://www.meyersound.com/products/constellation/

[3] http://www.cobranet.info/

[4] http://www.ethersound.com/

[5] http://www.avnu.org/

[6] http://www.meyersound.com/products/d-mitri/

[7] http://www.digico.biz/

[8] G. Duckett, T. Pennington, *Fixed-Point vs. Floating-Point DSP for Superior Audio*. Rane Audio Library Note 153.

[9] Ray Miller, *Second-order Digital Filters Done Right.* Rane Audio Library Note 157.

[10] Julius O. Smith III, *Introduction to Digital Filters with Audio Applications*. W3K, 2007

[11] Oppenheim, Schafer, *Discrete Time Signal Processing*. Pearson, 2010.

[12] A. Antoniou, *Digital Filters. Analysis, Design and Applications*. Mc Graw Hill 1993.

[13] W. Chen, *Performance of Cascade and Parallel IIR Filters*. J. Audio Eng. Soc., vol. 44, no. 3, pp. 148-158 (Mar. 1996)

[14] Timo I. Laakso, Jari Ranta and Seppo J.Ovaska: *Design and Implementation of Efficient IIR Notch Filters with Quantization Error Feedback*, IEEE Transactions on instrumentation, Vol. 43,No. 3, June 1994

[15] http://www.meyersound.com/products/processor_drive_systems/galileo/616/

[16] Toole, Floyd E., *Sound Reproduction*. Focal Press, 2009

[17] Harry F. Olson, *Acoustical Engineering*. Professional Audio Journals, Inc. 1991