UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

COMPRESOR BASADO EN TRANSFORMADA WAVELET PARA EL SISTEMA DE ADQUISICIÓN DE DATOS DE PÉTALO

Pedro Antequera Cañadas

Tutor: Vicente Herrero Bosch

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2018-19

Valencia, 2 de Septiembre de 2019

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV- 000030

Resumen

Este trabajo se engloba dentro del proyecto PETALO, el cual se diseña con el objetivo de mejorar las características técnicas de los actuales escáneres PET, muy empleados en medicina nuclear para medir la actividad metabólica de un organismo. En este contexto se presenta la necesidad de enviar la gran cantidad de información generada por el escáner, a lo cual surgen dos opciones: el uso de buses de alta velocidad o la compresión de los datos. Esta segunda opción se presenta como la más económica y sencilla de implementar en el estado actual del escáner, por lo que el presente trabajo tratará de lograr una implementación hardware funcional que disminuya la cantidad de datos que deben ser enviados. Para lograr este fin se usará la Transformada Wavelet Discreta (DWT) tras descartar otras opciones menos idóneas, la cual será implementada primero mediante un prototipo software y después en el diseño hardware final. En ambos casos se verificará el funcionamiento a fin de comprobar que es correcto.

Resum

Aquest treball s'engloba dins del projecte PETALO, el qual es dissenya amb l'objectiu de millorar les característiques tècniques dels actuals escàners PET, molt emprats en medicina nuclear per a mesurar l'activitat metabòlica d'un organisme. En aquest context es presenta la necessitat d'enviar la gran quantitat d'informació generada per l'escàner, a la qual cosa sorgeixen dues opcions: l'ús de busos d'alta velocitat o la compressió de les dades. Aquesta segona opció es presenta com la més econòmica i senzilla d'implementar en l'estat actual de l'escàner, per la qual cosa el present treball tractarà d'aconseguir una implementació maquinari funcional que disminuïsca la quantitat de dades que han de ser enviats. Per a aconseguir aquesta fi s'usarà la Transformada Wavelet Discreta (DWT) després de descartar altres opcions menys idònies, la qual serà implementada primer mitjançant un prototip software i després en el disseny hardware final. En tots dos casos es verificarà el funcionament a fi de comprovar que és correcte.

Abstract

This thesis is included within the PETALO project, which is designed having the aim of improving the technical characteristics of actual PET scanners, very used in nuclear medicine to measure metabolic activity of an living organism. In this context it is a need to send large quantity of the information generated by the scanner, which leads to two options: using high-speed buses of compressing data. This second option is more economical and easy to implement in the project at its actual state, so this thesis will try to reach a hardware functional implementation that decreases the quantity of data to be send. To achieve that, The Discrete Wavelet Transform (DWT) will be used after discarding less suitable options, which will be implemented first in a software prototype and after in the final hardware design. In both cases the operation of the module will be verified to make sure it works properly.

Índice

Capítulo 1. Introducción

El objetivo de este proyecto es el desarrollo de un sistema de compresión de imágenes biomédicas procedentes del escáner PETALO para su envío al sistema de procesamiento de datos. Estas imágenes poseen ciertas propiedades que facilitan su compresión haciendo uso de ciertas herramientas, entre la cual se encuentra la transformada wavelet, la cual será el sistema a emplear, no sin antes justificar el por qué de su uso. Sin embargo, antes de llegar a este paso, se introducirá el contexto en el que se desarrolla el trabajo, comenzando por la introducción a los sistemas PET para la obtención de imágenes biomédicas.

1.1 Introducción a los sistemas PET

Dentro del ámbito de la captura de imágenes médicas para el análisis y el tratamiento de pacientes, el estudio de enfermedades u otras aplicaciones, existen diversos métodos, cada uno de ellos con unas particularidades, ventajas y desventajas [1]. Dentro de este campo existen modalidades de captura basadas en exploraciones transmisivas, reflexivas o emisivas. En el primer grupo el paciente se coloca entre un emisor de fotones de alta energía, algunos de los cuales serán absorbidos por el paciente, y un detector de este tipo de partículas que recibirá aquellas no absorbidas por el paciente. Esto permite obtener una imagen directa sobre un plano del paciente, como es el caso de las radiografías de rayos X, o un conjunto de planos, los cuales mediante técnicas como el backprojection con o sin filtrado o la reconstrucción iterativa permiten obtener imágenes sobre un corte completo del paciente. Por otra parte se encuentran las exploraciones reflexivas, en las cuales el transmisor y el receptor se sitúan en un mismo plano y como su propio nombre indica, no se mide la transmisión de la señal a través del paciente, sino cómo ésta se refleja en él. En esta modalidad las ondas emitidas pueden tener ser de naturaleza mecánica, como en la obtención de imágenes mediante ultrasonidos, o electromagnética como en la obtención de imágenes mediante visión directa o en el caso de pulsioxímetros que utilicen el fenómeno de reflexión en lugar de el fenómeno de transmisión. Por último se encuentra el caso en el que el propio paciente es el transmisor, ya sea debido a la magnetización que provoca la alineación de los spines en la resonancia magnética o en el caso del que vamos a hablar, por la introducción de un material radioactivo en el paciente el cual interactúa con este y emite partículas, las cuales se transforman en señales eléctricas al detectarse en el receptor. A este último apartado, llamado medicina nuclear, es al que pertenece el escáner PETALO.

La medicina nuclear se utiliza para obtener información sobre todo acerca de rutas metabólicas o vasculares, es decir, sobre el funcionamiento del cuerpo y no sobre su estructura física. Esto se debe a que el material radioactivo, inyectado generalmente por vía intravenosa, se distribuye por el organismo siguiendo las rutas de los distintos compuestos que viajan por el interior del cuerpo del paciente de forma natural, emitiendo partículas a su paso. La naturaleza de estas partículas depende del material radioactivo introducido, o mejor dicho, de los radioisótopos que contiene, los cuales varían, no sólo en los tipos de partículas emitidas, sino también en su periodo de semidesintegración, que puede variar desde cientos de años hasta unos pocos minutos. Estos radioisótopos pueden obtenerse mediante aceleradores de partículas, reactores nucleares o generadores de radioisótopos. En el caso que nos involucra, las partículas emitidas serán positrones, partículas cuya masa es igual a la del electrón y su carga es igual a la del electrón en magnitud pero de signo opuesto, de manera que tanto los generadores de radioisótopos como algunos aceleradores de partículas resultan idóneos para este fin.

Dentro de los sistemas de medicina nuclear se encuentran la gammagrafía, la Tomografía Computerizada por Emisión Monofotónica (SPECT) y la Tomografía por Emisión de Positrones (PET), siendo esta última a la que pertenece el escáner PETALO. La Tomografía por Emisión de Positrones, en realidad, no detecta la emisión de los positrones en el detector, sino que detecta los fotones generados cuando el positrón emitido alcanza a algún electrón de la capa externa de un átomo cualquiera y se produce el efecto de aniquilación [2] dibujado en la Fig. 1, donde la masa de ambas partículas se transforma en energía en forma de dos fotones, los cuales viajan en la misma dirección y en sentidos opuestos, teniendo cada ellos una energía de 511 keV.

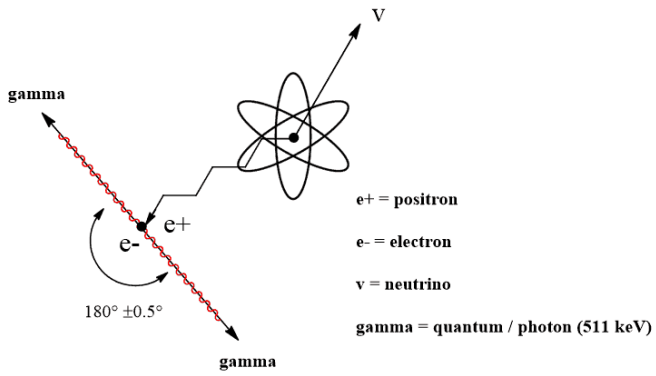

Fig. 1: Esquema del evento de aniquilación positrón-electrón. [3]

Cuando dos de estos fotones llegan al detector en un corto periodo de tiempo se detecta un evento debido al efecto de aniquilación, el cual debería haber ocurrido en algún punto entre la línea imaginaria que une los detectores donde han incidido ambos fotones. Sin embargo, existen algunos artefactos que pueden distorsionar la detección y dar lecturas falsas. Estos artefactos, detallados en la Fig. 2, pueden producirse por la desviación de uno o ambos fotones o debido a que se produzcan dos o más eventos en un breve espacio de tiempo y los fotones de ambos eventos se confundan. Deberá ser pues, el equipo que procese los datos el encargado de tratar de minimizar el impacto que estos artefactos producen en la imagen, para que esta muestre la información realmente útil, es decir, el objeto de estudio en el paciente.

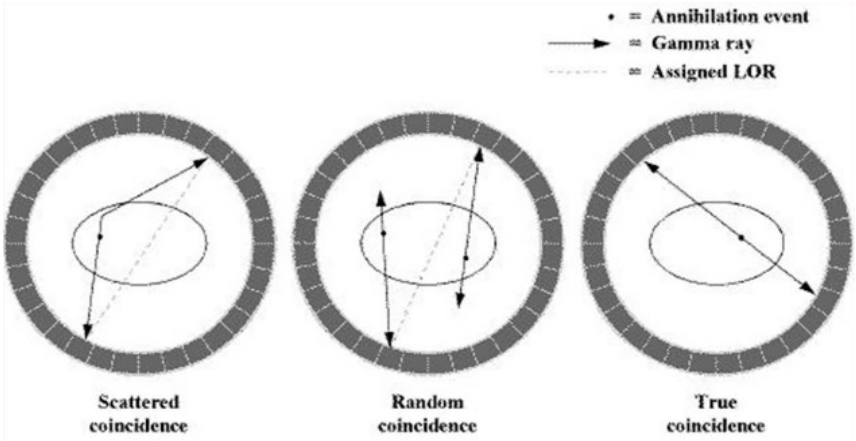

Fig. 2: Posibles artefactos (izquierda y centro) y una detección correcta en el sistema. [1]

Por otra parte, existe un efecto indeseado inherente al propio principio físico en el que se basa el funcionamiento del sistema debido a la distancia que recorre el positrón hasta encontrar un electrón con el que se produzca la aniquilación. Esta distancia recorrida se encuentra en valores comprendidos entre 0,54 mm para el radioisótopo F^{18} y llegando hasta 6.14 mm para el radioisótopo Rb^{82} [4]. Esto supone una incertidumbre añadida a la medida, ya que aunque el sistema de detección de comportase de manera perfecta no bastaría para poder obtener el punto exacto donde se emitió el positrón. Dicha limitación provoca que en la práctica las imágenes obtenidas mediante PET posean unas formas ligeramente difuminadas en lugar de contornos bien definidos como los que se pueden obtener mediante otras técnicas. A pesar de estas limitaciones, los sistemas PET resultan extremadamente útiles para obtener información sobre el metabolismo, la perfusión o comportamientos anómalos entre otros usos, y pueden usarse en conjunto con otras técnicas como la resonancia magnética, tal y como se muestra en la Fig. 3, para obtener información, por ejemplo, sobre el flujo sanguíneo en una zona concreta o la actividad cerebral ante distintos estímulos.

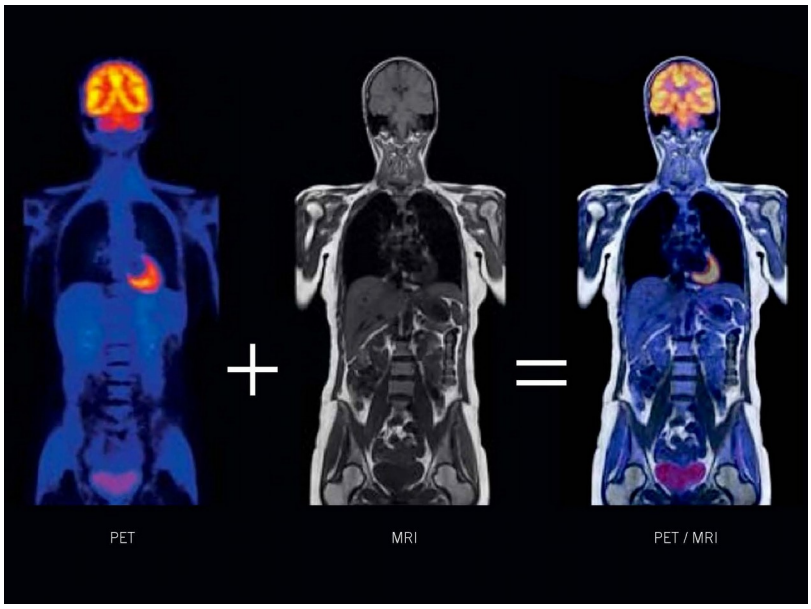

Fig. 3: Ejemplo de imágenes médicas obtenidas mediante PET (izquierda), resonancia magnética (centro) y la superposición de ambas (derecha). [5]

Conocido el principio físico en el que se basan los detectores PET, se puede entrar a profundizar sobre su funcionamiento. Como se ha comentado anteriormente, la aniquilación positrón-electrón emite dos fotones gamma cada uno con 511 keV de energía, aunque éstos no inciden directamente sobre los fotodetectores sino que inciden sobre unos elementos llamados cristales centelleadores. Estos elementos en presencia rayos gamma absorben su energía [6], lo que provoca que se exciten y emitan múltiples fotones de una energía y frecuencia mucho menor, la cual puede encontrarse incluso en el espectro visible, tal y como ocurre en los tubos fluorescentes al exponerse a radiación ultravioleta. Gracias a este efecto se aumenta la sensibilidad del sistema, debido a que los fotodetectores reciben una cantidad de fotones muy superior a una frecuencia inferior a la original de los fotones emitidos por el paciente. Esto permite que la sensibilidad del sistema sea mucho mayor, disminuyendo así la probabilidad de que los fotones generados en otro lugar introduzcan ruido. Sin embargo, para generar suficientes electrones por fotón para que la señal eléctrica pueda procesarse correctamente por la electrónica es necesario otro elemento, el fotomultiplicador, el cual se compone de una serie de elementos llamados dinodos, los cuales multiplican los electrones a su entrada, consiguiendose así la amplificación necesaria para el funcionamiento correcto del sistema. Este conjunto formado por el centelleador, el fotomultiplicador y el fotodetector se muestra en la Fig. 4.

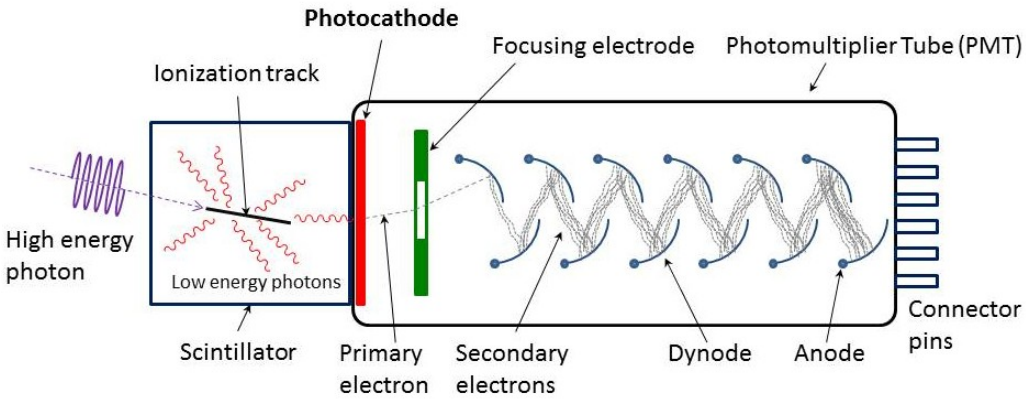

Fig. 4: Detalle del centelleador (izquierda) y fotomultiplicador y fotodetector (derecha). [7]

1.2 Introducción al escáner PETALO

El escáner PETALO nace a partir de una nueva forma de obtener imágenes PET [8] mediante modificaciones en los materiales empleados y en la forma de procesar la información. La mayor diferencia con respecto a los sistemas PET existentes en la actualidad reside en el elemento centelleador, siendo, en la mayoría de diseños actuales, cristales centelleadores agrupados en matrices. Estos cristales centelleadores pueden acoplarse individualmente o formando pequeñas matrices a cada uno de los fotodetectores encargados de transformar las señales lumínicas en señales eléctricas. Gracias a esta disposición se simplifica en gran medida a localización de las interacciones físicas que se dan entre los rayos gamma, emitidos debido a la aniquilación positrón-electrón, y los cristales centelleadores aunque, por contra, disminuye la resolución tanto en tiempo como en cantidad de energía detectada y limita la resolución espacial al tamaño de un píxel, es decir a cada fotodetector. A estos inconvenientes propios del funcionamiento hace falta añadir otros debidos a la fabricación del propio escáner entre los que destacan el

elevado coste de algunos materiales, las limitaciones de forma y tamaño en cuanto a la fabricación de los cristales centelleadores y la dificultad de fabricación del propio escáner, sobre todo en los escáneres PET de gran tamaño como los escáneres de cuerpo completo. Con el fin de paliar algunos de estos inconvenientes, en concreto los relativos al funcionamiento, se han investigado distintas soluciones, destacando entre ellas la solución que emplea cristales centelleadores de mayor tamaño, cada uno de los cuales tiene asociado un conjunto de fotodetectores, lo que permite obtener y analizar las características del evento gamma con mayor detalle a cambio de una pérdida en resolución espacial.

Debido a todos los problemas que presenta el uso de cristales centelleadores se han realizado distintas investigaciones enfocadas a sustituir estos elementos por otros con mejores características tanto a nivel de comportamiento como a nivel de fabricación. Algunas de estas investigaciones se han enfocado en el uso de Xenón Líquido como elemento centelleador, como es el caso del escáner PETALO. El Xenón Líquido presenta propiedades que lo hacen una buena opción frente a los tradicionales cristales centelleadores, como el alto rendimiento de centelleo, la rapidez de respuesta y la posibilidad de construir detectores con mayor flexibilidad en cuanto a tamaños para una gran cantidad de aplicaciones. A todo esto hay que tener en cuenta que, obviamente, este medio es líquido, lo que conlleva que sea un medio continuo y homogéneo en lugar de un medio con discontinuidades típico de los cristales centelleadores, que a su vez implica una mayor resolución, un menor efecto de bordes en la distribución de la luz detectada y la ausencia de distorsión geométrica al no existir discontinuades. Esto supone que los datos obtenidos tendrán un aspecto distinto, aunque la información que se obtendrá de ellos será equivalente a la obtenida con otros escáneres, añadiendo eso sí las mejoras que presenta el Xenón Líquido. Debido a las diferencias en las imágenes de los eventos obtenidas será necesario rediseñar la electrónica encargada de obtener la información para lograr un funcionamiento correcto y sacar partido a las ventajas de este nuevo sistema.

Sin embargo, no todo son ventajas, ya que el hecho de utilizar Xenón Líquido conlleva dificultades añadidas por las bajas temperaturas a las que debe operar. El Xenón se mantiene en estado líquido entre 161 K y 165 K [9], siendo la temperatura de funcionamiento elegida para el criostato 163 K con el fin de lograr un margen simétrico lo mayor posible ante fluctuaciones de la temperatura y mantener el Xenón en estado líquido. Esto hace necesario, por una parte un sistema de refrigeración capaz de alcanzar y mantener esa temperatura por un periodo de tiempo relativamente largo, y por otra parte que ciertos elementos sean capaces de trabajar a bajas temperaturas. Sin embargo, con objeto de evitar el desarrollo de electrónica capaz de trabajar a temperaturas tan bajas, se puede situar la electrónica del detector en una cámara de vacío a muy baja presión, lo que provocará que la energía térmica que ceda la electrónica al criostato por convección a través del aire sea prácticamente nula y por tanto ésta se pueda mantener a una temperatura ambiente. Esto hace posible el uso de electrónica convencional, abaratando los costes de fabricación y disminuyendo el tiempo necesario para el desarrollo del prototipo del escáner. Además, el uso de temperaturas bajas tiene un efecto positivo respecto al ruido generado por los fotodetectores, debido a que el ruido térmico es la mayor contribución. Por lo tanto, disminuyendo la temperatura, el ruido en la imagen disminuirá significativamente, hasta

el punto de que experimentalmente se ha comprobado que el ruido en los fotodetectores es despreciable.

Vistas las diferencias entre el escáner PETALO y los escáneres PET actuales es evidente que la electrónica encargada de generar los datos que se llevarán al sistema de procesado para generar las imágenes biomédicas será distinta a la existente. Para ello se va a diseñar la electrónica teniendo en cuenta el funcionamiento anteriormente comentado para el primer prototipo de escáner PETALO, el prototipo PETIT, de 17 cm de altura y 36 cm de diámetro, que en versiones sucesivas permitirá aumentar la altura con el fin de reducir el efecto de bordes en el área útil del escáner. Este aumento debe tenerse en cuenta a la hora de diseñar la electrónica, ya que ésta debe ser fácilmente escalable y no degradar el comportamiento temporal del sistema debido a retardos excesivos, lo que permitirá poder sacar partido a todas las ventajas que aporta este nuevo sistema. Para poder desarrollar este sistema de manera más sencilla se ha desarrollado un banco de pruebas escrito en Python, lenguaje muy usado en algunas herramientas de simulación física, como las usadas en este banco de pruebas, a las que se suma la librería “SymPy”, desarrollada para realizar análisis de estímulos basados en eventos tal y como se realiza físicamente en el escáner. Una de las desventajas del escáner PETALO es la ingente cantidad de información generada por unidad de tiempo a la entrada de los elementos L1 (Fig. 5) puede llegar a ser de hasta 25 Gbit/s, información que tras procesarse en las placas L1 disminuye hasta necesitar un ancho de banda de unos 2.2 Gbit/s. Esta cantidad de información sigue siento excesiva, por lo que se plantea el bloque L2, el cual se dedicará a la compresión de los datos para bajar el ancho de banda lo máximo posible sin introducir errores demasiado elevados para los valores con los que trabaja el escáner.

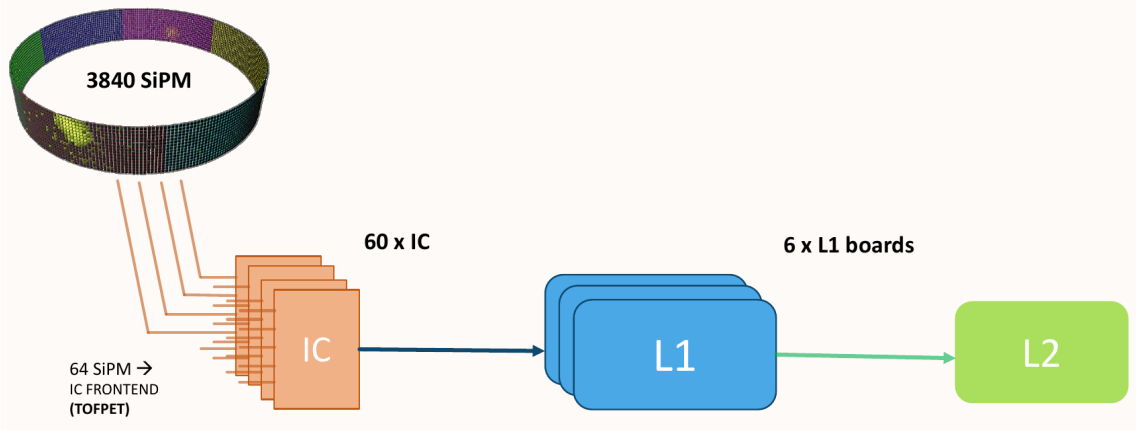

Fig. 5: Esquema de los elementos básicos del sistema de adquisición de datos.

Para poder disminuir la cantidad de datos que envía el sistema pueden plantearse múltiples soluciones, las cuales deben ser rápidas, compatibles con la fragmentación espacial, lo más sencillas posibles y capaces de trabajar con cambios bruscos en la imagen.. La compatibilidad con la fragmentación espacial es necesaria debido a la estructura de los bloques L1, la cual hace posible que los eventos puedan darse entre dos elementos L1 tal y como se muestra en la Fig. 6. Por otra parte, la sencillez viene impuesta por la necesidad de usar el menor número de recursos de la FPGA posible para que sea posible la implementación del sistema completo en la

electrónica programable. La capacidad para trabajar con cambios bruscos por otra parte, viene dada por la propia naturaleza de las imágenes recogidas por el escáner.

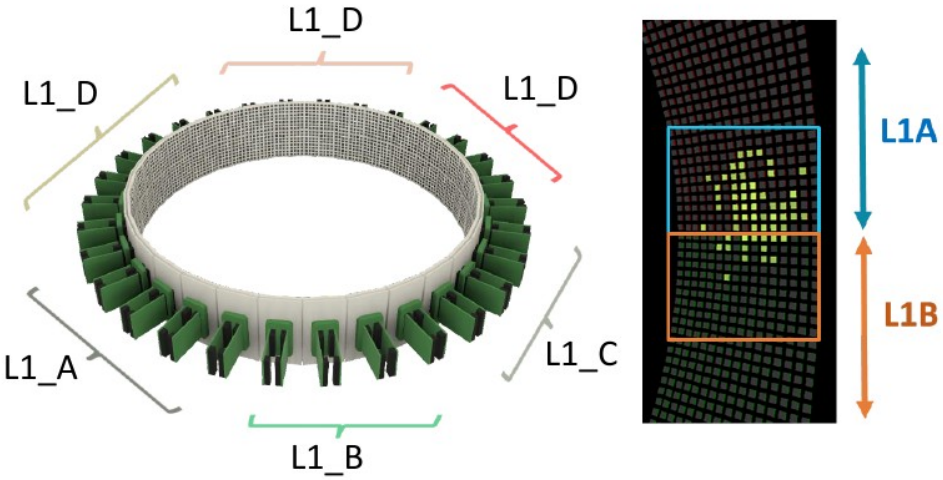

Fig. 6: Detalle de la disposición física de los elementos L1 y de un caso donde el evento se produce entre dos elementos L1 dando lugar a una fragmentación espacial.

Estas características hacen que los codificadores de entropía queden descartados por su lentitud y complejidad, quedando como posibles soluciones la codificación de información mediante redes neuronales y la compresión mediante transformada wavelet. La primera de ellas requiere el uso de redes neuronales entrenadas, las cuales se encargarán de elegir los datos más significativos de las imágenes y descartar aquellos con poca o ninguna información sobre los eventos. Sobre la teoría este sistema puede llegar a ofrecer ratios de compresión elevados con una complejidad no demasiado elevada a la hora de implementarlo sobre una FPGA, siendo el desarrollo de ésta complejo al necesitar estimar el tamaño y el número de capas de la red neuronal y entrenar dicha red. A pesar de estas mejoras, durante el desarrollo los sistemas del escáner el prototipo de red neuronal ha presentado una gran degradación de la información sobre los eventos como puede observarse en la Fig. 7, por lo que se ha decidido dejar de lado esta opción, que requeriría un desarrollo mucho mayor para ser operativa, y optar por la compresión mediante transformada wavelet.

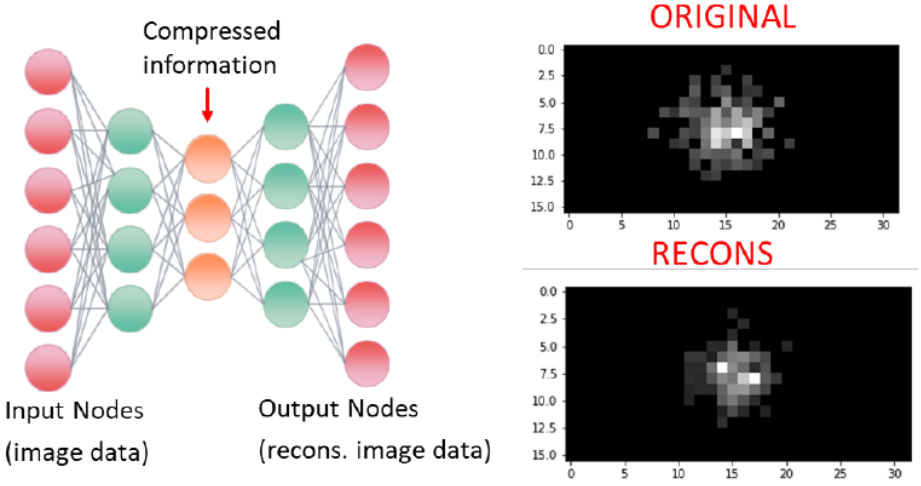

Fig. 7: Método de compresión empleando una red neuronal entrenada.

La técnica de compresión mediante transformada wavelet se basa en el uso de esta herramienta, similar en algunas características a la transformada de Fourier, para cambiar de base los datos y concentrarlos en algunos coeficientes. Esta transformada es reversible y no produce pérdida alguna, aunque tampoco comprime los datos, por lo que, para aprovechar sus características se deben aplicar unos umbrales con el fin de enviar sólo los datos que superen éstos, lo que elimina algunos datos menos significativos. Tras esto los datos se codifican con un número diferente de bits en función de la zona en la que se encuentren, usando un mayor número para aquellos coeficientes que concentren la mayor cantidad de información y un menor número para aquellos coeficientes que tengan un menor peso en los datos de los eventos registrados. Gracias a esto, se consigue una compresión de datos que dependerá del tipo de wavelet, los umbrales y la codificación de bits elegidos, sobre los cuales se hablará más en profundidad en la introducción a la transformada wavelet. El ejemplo de codificación con la que partirá el desarrollo del proyecto se muestra en la Fig. 8, donde se puede apreciar la concentración de los datos en los coeficientes de baja frecuencia y las diferencias entre la imagen del evento original y la reconstrucción tras atravesar el compresor.

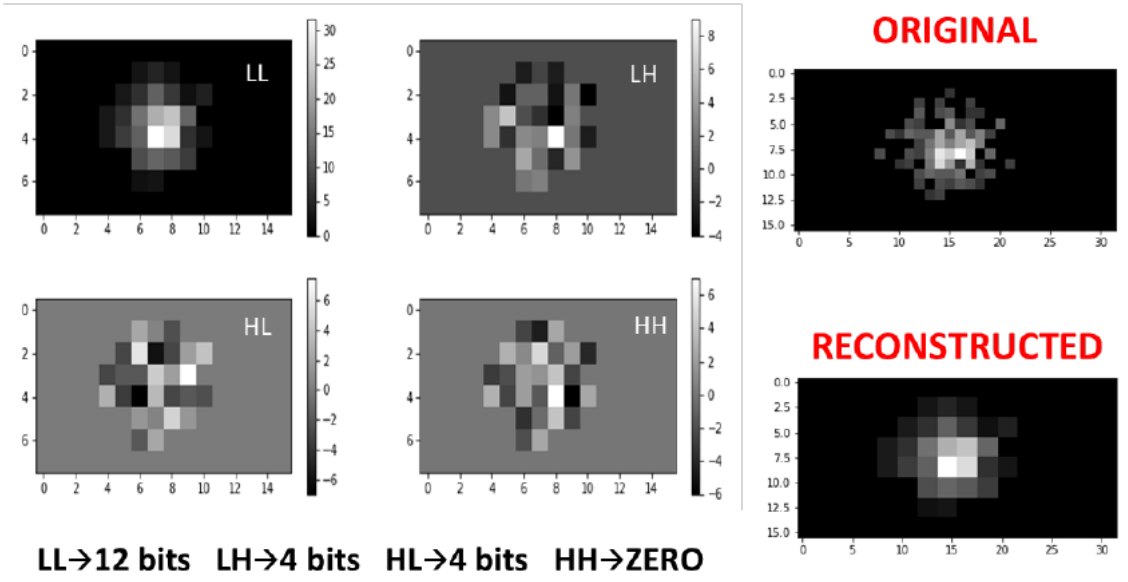

Fig. 8: Método de compresión empleando DWT (Transformada Wavelet Discreta).

Como se ha podido observar en las Fig. 7 y 8, las imágenes obtenidas por el escáner PETALO contienen los eventos detectados, los cuales previamente han sido ordenados temporalmente en los elementos L1, los cuales contienen a su vez la información sobre las coordenadas en las que ocurrió el evento. Estas coordenadas serán extraídas por el sistema de procesamiento de datos al que se envían los datos, aunque conviene destacar que para que esto sea posible deben poder extraerse a partir de la imagen su centro (media) y su anchura o dispersión (desviación típica). Por lo tanto, el sistema de compresión de imágenes deberá realizarse para conseguir los menores cambios posibles en estos dos datos, los cuales se usarán para comprobar la degradación de la imagen tras atravesar el compresor y evaluar su comportamiento en función de los umbrales, codificadores de bits y wavelets usadas.

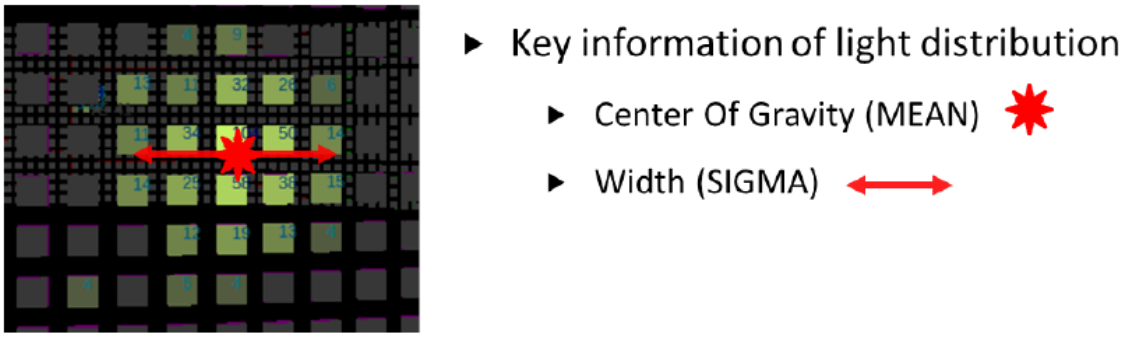

Fig. 9: Información clave a extraer de las imágenes generadas por los fotodetectores.

Por último queda añadir que el escáner se encuentra actualmente en fase de desarrollo y construcción, lo que motivará más adelante a que el desarrollo del proyecto trate, en la medida de lo posible, de realizarse de forma paramétrica. Esto permitirá que los posibles cambios que deban realizarse en el prototipo no impliquen el rediseño completo del sistema de compresión, sino que con pequeñas modificaciones se pueda reutilizar en la placa del sistema, mostrada en la Fig. 10. También es necesario añadir, en relación con el prototipo, que la compresión mínima que debe lograr el proyecto se encuentra en un valor alrededor de 0.6, de acuerdo con los experimentos realizados.

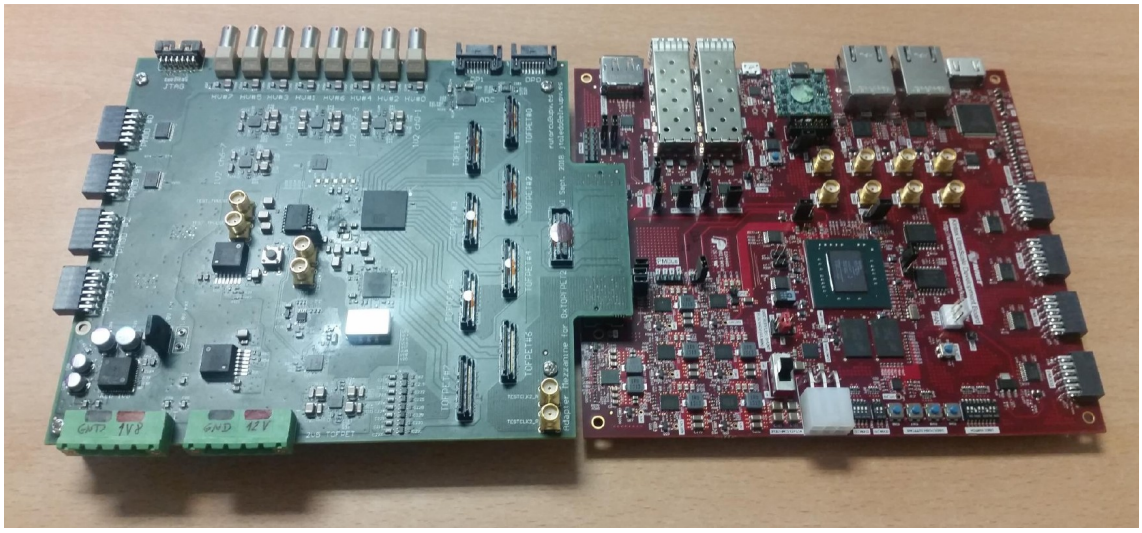

Fig. 10: Prototipo de la placa de adquisición de datos.

1.3 Introducción a la transformada wavelet

Para comprender por qué se ha elegido usar la transformada wavelet para realizar la compresión de imágenes es necesario primero introducirla y nombrar sus características más importantes. La transformada wavelet, debido a algunas características que se nombrarán más adelante, es hoy en día una herramienta muy útil para el estudio espectral de señales haciendo uso de distintas escalas o resoluciones, reconstrucción de señales con un alto nivel de ruido o para la compresión de datos e imágenes, usado en algunos estándares muy populares como JPEG2000. El origen de esta herramienta se encuentra a principios del siglo XX [10], cuando Alfréd Haar,

un matemático de origen húngaro que vivió entre finales del siglo XIX y principios del siglo XX, propone en un apéndice de su tesis la primera función wavelet, la cual lleva el nombre de su descubridor. La transformada wavelet tipo Haar es no nula sólo en el intervalo [0,1] tal y como se muestra en la Fig. 11, y es, de todas las transformadas wavelet, la más sencilla.

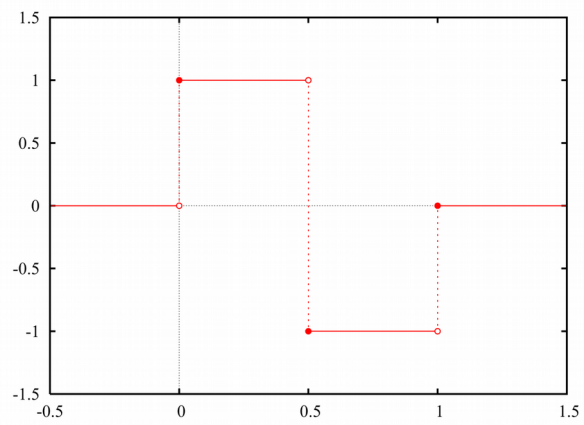

Fig. 11: Wavelet tipo Haar. [11]

Esta transformada compartía algunas características con otra más antigua y conocida, la transformada de Fourier. Ambas trabajan convirtiendo señales expresadas en dominio del tiempo a señales expresadas en dominio de la frecuencia, lo que resulta extremadamente útil en comunicaciones con aplicaciones simples como la representación espectral de señales hasta otras más complejas como la caracterización del comportamiento de componentes en frecuencia o el diseño de filtros. Este cambio de dominio se consigue expresando la señal a la entrada en función de la de la base de cada transformada y sus combinaciones lineales, seno y coseno o exponenciales complejas en el caso de la transformada de Fourier, y funciones wavelet en el caso de la transformada homónima. Estas funciones wavelet, a diferencia del seno y del coseno, están localizadas, no sólo en frecuencia, sino también en el espacio, lo que provoca que ciertas señales introducidas a la entrada logren que a la salida la información se concentre en ciertos coeficientes, dejando otros con una relevancia mucho menor. Gracias a este efecto, los coeficientes con menor relevancia pueden comprimirse y disminuir la cantidad de datos necesarios para transmitir una determinada información introduciendo un error relativamente pequeño. También puede usarse este efecto para suavizar la imagen y eliminar ruido aunque es posible que las frecuencias más altas se vean atenuadas.

Otra propiedad interesante de las transformadas wavelet, descubierta por Paul Levy, es la capacidad de estudiar tanto pequeños y complicados detalles de una señal como las formas generales que tiene ésta. Esto por ejemplo puede ser de utilidad en los casos en que la señal tenga anchos de banda muy elevados como transmisiones en banda base ya que permitiría estudiar con el mismo detalle señales de alta y baja frecuencia gracias al cambio de escala o resolución. Debido al cambio de escala, la transformada wavelet puede aplicarse a fragmentos de la señal de distinto tamaño, por lo que se obtienen datos interesantes para cada escala de manera similar a como se puede llevar a cabo el estudio de una muestra bajo el microscopio

usando distintos aumentos para ver ciertos detalles con cada tipo de aumento. Esta propiedad diferenciadora con respecto a la transformada de Fourier es la que motiva su empleo en ciertos ámbitos hoy en día, ya que hace posible el estudio en escala de señales y además permite emplearse con señales con cambios pronunciados, donde la transformada de Fourier no es demasiado útil debido a que puede introducir algunos artefactos ante cambios muy bruscos.

De la misma forma que existe la Transformada Discreta de Fourier (DFT), también existe la Transformada Discreta Wavelet (DWT) [12], las cuales se usan el el ámbito digital para tratar señales discretizadas. La DWT comparte con su análoga analógica las mismas propiedades, por lo tanto la base en la que se representan las señales a la salida de la DWT está formada por funciones wavelet discretas, las cuales se agrupan en familias en función de sus características, que dependiendo de la aplicación concreta en la que se deseen utilizar serán más o menos ventajosas. Cada wavelet dentro de estas familias posee un número de coeficientes y forma y se implementa mediante cuatro filtros, dos usados para realizar la DWT y otros dos para reconstruir la señal a partir de su DWT, siendo uno de los filtros usados tanto en la implementación de la DWT como de su inversa paso-bajo y el otro de ellos paso-alto tal y como se muestra en la Fig. 12. Se dice entonces que las funciones usadas para implementar la DWT son complementarias a las funciones usadas para implementar su inversa ya que si se aplican ambas a una señal, ésta permanece inmutable salvo por pequeños errores cometidos inevitablemente durante los cálculos en punto flotante. Siguiendo con lo anterior, a priori puede parecer que la transformada no aplica ninguna ventaja, ya que el número de coeficientes al aplicar la DWT es incluso ligeramente mayor que el número de coeficientes de la señal original, pero sin embargo, una de las propiedades más interesantes de la DWT es que la información se concentra en ciertos coeficientes, mientras que otros pueden sufrir cambios relativamente importantes sin que tenga un efecto apreciable en la reconstrucción de la señal original. Gracias a esto se puede aplicar una codificación con un alto número de bits a la zona donde se agrupan los coeficientes más significativos y emplear una codificación con un número de bits menor para aquellos coeficientes cuyo peso en el valor de los datos sea menor, consiguiendo así comprimir los datos sin sufrir pérdidas significativas. Esta última característica es la que motivará principalmente el uso de la DWT en el sistema de adquisición de datos PETALO para comprimir los datos obtenidos del sistema antes de proceder a su envío al sistema que procesará los eventos físicos detectados.

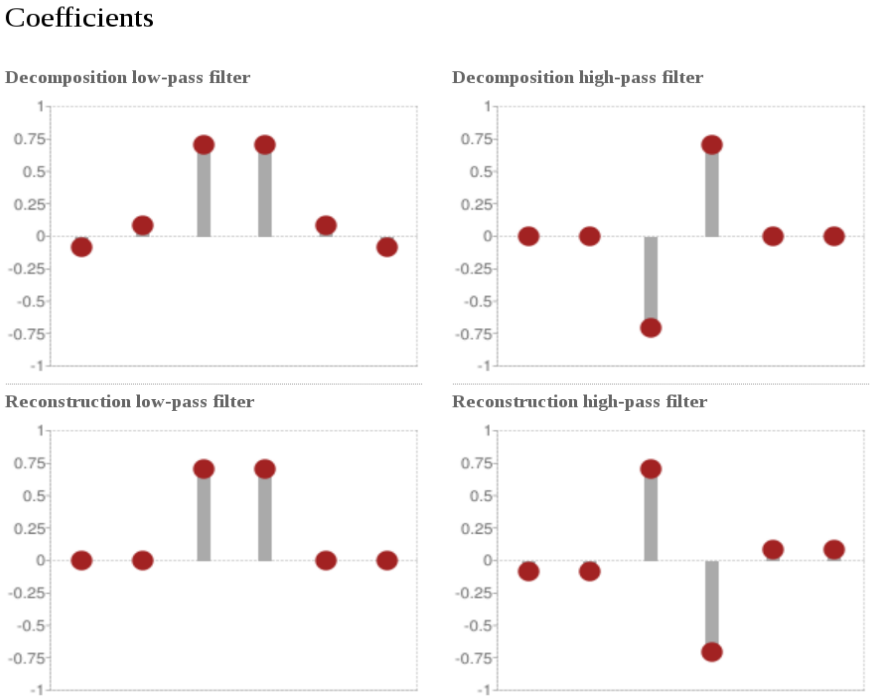

Fig. 12: Ejemplo de los filtros de cada DWT. En este caso se muestra la DWT biortogonal 1.3. [13]

En el caso de la transformada wavelet a implementar, dada la naturaleza de los datos a procesar, deberá ser de dos dimensiones. Esto abre la puerta a dos posibles implementaciones, por una parte puede implementarse directamente la transformada wavelet en dos dimensiones mediante el empleo de un filtrado bidimensional haciendo uso de una matriz de coeficientes, y por otra parte puede aprovecharse la propiedad de ortogonalidad de las funciones wavelet para realizar un filtrado en cada eje, lo que permitiría reducir la cantidad de memoria y elementos CLB a usar en la implementación hardware. La forma de implementación se discutirá más adelante, durante la fase de diseño del prototipo en lenguaje de programación de alto nivel.

Capítulo 2. Objetivos

Este proyecto nace como solución al problema que presenta el escáner PETALO en cuanto al ancho de banda necesario para poder enviar toda la información correctamente al sistema de procesamiento de datos. Como se ha comentado durante la introducción, de las dos posibilidades elegidas inicialmente para la compresión de datos son: redes neuronales y transformada wavelet, siendo esta última la única que parece lograr los resultados esperados, y por tanto, sobre la que tratará el diseño. Para llevar a cabo este proyecto se han definido los siguientes objetivos a lograr:

- Obtención de la wavelet óptima para los datos de simulación, así como los umbrales y número de bits necesarios para lograr un ratio de compresión lo mayor posible manteniendo los errores de medición dentro de unos márgenes aceptables.
- Desarrollo de un modelo de implementación de la transformada wavelet en un lenguaje de alto nivel que permita lograr el mismo cometido que las funciones de la librería empleadas en el banco de pruebas usado en el punto anterior.
- Adaptación del modelo anterior para sustituir el funcionamiento en punto flotante por un funcionamiento en punto fijo, el cual permita una aproximación más cercana al funcionamiento del hardware.
- Desarrollo de un sistema de verificación simple, capaz de comparar los resultados de la implementación realizada, tanto en punto flotante como en punto fijo, con los resultados de la librería para detectar errores de diseño y facilitar la depuración.
- Implementación del modelo, tanto en punto flotante como en punto fijo, en el banco de pruebas del escáner PETALO para verificar el funcionamiento de estas implementaciones con datos reales.
- Implementación del modelo desarrollado anteriormente mediante un lenguaje HDL, lo que implica investigar distintas formas de implementación de los elementos internos.
- Comprobación del código HDL del modelo mediante un sintetizador RTL para verificar que el código es sintetizable y, por tanto, puede implementarse sobre una FPGA.
- Desarrollo de un banco de pruebas para comprobar el correcto funcionamiento del modelo desarrollado, primero con datos de prueba y seguidamente con datos reales.

Capítulo 3. Metodología

3.1 Herramientas y lenguajes usados

Las herramientas empleadas en el diseño, depuración y verificación del proyecto han sido bien distintas, ya que para poder lograr desarrollar este sistema debe partirse desde una idea general de cuál deberá ser el funcionamiento y algunas especificaciones básicas sobre cómo deberá comportarse. Para este fin resulta idóneo el uso de lenguajes de programación de alto o muy alto nivel donde el desarrollador puede abstraerse del funcionamiento a bajo nivel como la gestión de memoria, y únicamente preocuparse de recrear el funcionamiento del sistema y un pequeño banco de pruebas para comprobar este funcionamiento. Estos lenguajes, como Python, Matlab, Java o C++ poseen además la característica de poder realizar prototipos de manera rápida y sencilla, aunque con una optimización, en general, bastante pobre debido a su propia naturaleza. Es por ello, que durante la fase de desarrollo del prototipo a alto nivel se han empleado python funcionando sobre jupyter-notebooks para la verificación del diseño y tanto python como matlab en jupyter-notebooks y MatlabIDE respctivamente, para el desarrollo del prototipo que se usará como referencia para crear el dispositivo hardware a implementar sobre un dispositivo de microelectrónica programable FPGA.

Sin embargo, usar lenguajes de alto nivel no es suficiente para lograr el diseño hardware, ya que no es posible, en general, alcanzar circuitos digitales óptimos a partir de estas descripciones que puedan ser implementados usando una FPGA. gPara ello, una vez terminado el prototipado en alto nivel se comenzará con el desarrollo del hardware haciendo uso de un lenguaje de descripción hardware, habiéndose valorado para ello tres posibles opciones, VHDL, Verilog y SystemVerilog. De estos tres candidatos se ha terminado seleccionando el tercero de ellos, SystemVerilog, debido a la sintaxis similar a Verilog, las herramientas que añade a éste lenguaje para descripción pero sobre todo para verificación y la mayor extensión, hablando en términos de uso, que posee comparado con VHDL. A esto se ha añadido el hecho de que ModelSim y Questasim, usadas para verificación, son compatibles con este lenguaje, así como los sintetizadores de Xilinx y Altera entre otros, usados para comprobar que el hardware descrito es capaz de sintetizarse correctamente y lograr un correcto funcionamiento en ambas plataformas.

3.2 Metodología de diseño

La metodología de diseño del proyecto se puede dividir en tres partes bien diferenciadas: la elección de la transformada wavelet a emplear, el diseño del sistema de compresión wavelet con punto fijo en software, y por último la implementación de dicho diseño en un lenguaje de descripción de hardware. Estas dos últimas tareas pueden dividirse, a su vez, en dos subtareas bien diferenciadas, por una parte la creación del diseño y por otra parte la verificación de éste junto a la evaluación de sus características técnicas.

Siguiendo con este guión previamente establecido, en primer lugar, se ha partido del banco de pruebas usado para comprobar el funcionamiento del proyecto PETALO, el cual está diseñado en Python y que hace uso de la librería pywt para implementar la transformada wavelet. De este banco de pruebas se usará la parte destinada a la comprobación del compresor de datos, formado por tres módulos: la DWT de dos dimensiones, un umbral para cada salida del filtro y por útimo un diezmador en cada salida, el cual se encarga de obtener el número de bits más significativos que se especifiquen en la simulación, logrando de esta manera la compresión deseada. Tomando esta parte del banco de pruebas se ha realizado un barrido modificando el tipo de transformada wavelet que utiliza el sistema y posteriormente se han ajustado tanto los umbrales como el diezmando de bits para lograr que la dispersión, tanto de las medias como de las desviaciones típicas de los errores fueran lo menores posibles. Un efecto deseable para lograr un mejor funcionamiento sería la concentración de errores alrededor del cero con una forma lo más simétrica posible con respecto a éste a la vez que la cantidad de datos enviados se minimiza para lograr un ratio de compresión lo mayor posible sin introducir demasiados errores. Cabe destacar que para esta tarea se han empleado datos reales recogidos por el prototipo del sistema, por lo que el comportamiento real debería ser bastante similar al comportamiento de la simulación, siempre y cuando no se realicen cambios significativos en el sistema.

Gracias a esta primera fase se obtienen tanto el tipo de wavelet como los umbrales y el diezmado ótimos, los cuales servirán para realizar las pruebas del diseño que se desea implementar, por lo que con estos datos puede procederse a realizar el diseño de un prototipo del sistema en un lenguaje de programación en alto nivel, el cual permitirá llevar a cabo un prototipado rápido y hacer pruebas cercanas a lo que será el diseño final. Para llevar a cabo esta tarea se ha recreado el funcionamiento de la librería que permite realizar la transformada wavelet mediante el uso de funciones que realizan pequeñas y concretas tareas, aunque con una diferencia, en este caso la transformada wavelet bidimensional se va a implementar con transformadas wavelet unidimensionales, donde se realizará un barrido en cada eje aprovechando las propiedades de ortogonalidad de la transformada wavelet. Una vez realizada la implementación, se creará un pequeño banco de pruebas para comprobar si la implementación es correcta, calculando el máximo error que se produce entre la función implementada y la función de la librería, error que debería ser muy bajo aunque probablemente no nulo, dado que pequeñas diferencias en los cálculos en punto flotante provocarán que se introduzcan errores de una magnitud muy inferior al valor de los coeficientes. Tras comprobar esto se sustituirá la llamada a la librería en el banco de pruebas del sistema por la llamada a la función creada y se comenzará a evaluar la cuantificación para coeficientes del filtro, datos de

entrada y datos de salida en caso de trabajar en punto fijo con una resolución determinada. Esto permitirá evaluar el efecto que tiene trabajar con este tipo de aritmética antes de comenzar la implementación hardware y permitirá tener una idea bastante aproximada al funcionamiento de éste, aunque podrán existir ciertas diferencias dado que los cálculos en el prototipo de alto nivel se han realizado en punto flotante con entradas y salidas en punto fijo debido a la dificultad añadida que suponía realizar los cálculos en punto fijo y manejar a la vez eventos como el overflow en los cálculos. Por último en lo que respecta a la metodología de diseño del prototipo en alto nivel, es necesario destacar que se han usado dos lenguajes, Matlab y Python. El uso del primero viene dado por la sencillez a la hora de depurar el programa gracias a las herramientas de depuración incluidas en el IDE de Matlab, mientras que el uso del segundo viene motivado por la necesidad de usar un sólo lenguaje para la verificación del prototipo, lo que implica menor complejidad de diseño y depuración que el uso de dos lenguajes y mayor portabilidad al eliminar la necesidad de disponer de una licencia para el uso de Matlab. Por lo tanto se ha realizado el prototipo de las funciones y su verificación en Matlab y posteriormente se ha portado a Python, verificando de nuevo su comportamiento tras realizar las adaptaciones oportunas debido a las diferencias entre ambos lenguajes. Asimismo hace falta indicar que gracias a esta implementación, bastante similar en términos de funcionamiento al hardware, se ha profundizado en las características de las transformadas wavelet, lo que resultará especialmente útil para la implementación en un dispositivo de electrónica programable.

Tras completar el diseño y verificación del prototipo del diseño en un lenguaje de programación de alto nivel es posible partir de éste para realizar el diseño en un lenguaje de descripción hardware, el cual será procesado por el sitetizador y será transformado en un bitstream, el cual se introducirá en el dispositivo programable para realizar la función para la que está diseñado. En este caso, al tener un prototipo de referencia con una estructura muy similar a la que tendrá el hardware, se va a aprovechar para crear cada uno de los módulos con los que se conformará el diseño. Además, se va a diseñar el hardware de tal manera que sea parametrizable dentro de lo posible, para que en caso de querer cambiar la resolución en número de bits usada para los datos, el tamaño de las imágenes, el número de filtros, etc. sea posible reutilizar el diseño realizando cambios mínimos. Sin embargo, es en el banco de verificación donde se ha puesto el mayor esfuerzo, ya que éste se ha realizado de tal manera que exista un programa en Python que permita crear los archivos de memoria en el formato utilizado por Modelsim para introducir en el diseño los datos y los coeficientes del filtro. En este momento podrá lanzarse la simulación en Modelsim, la cual cargará los datos de estos archivos y realizará la simulación, donde se obtendrán los datos y se guardarán en otro archivo de resultados. Por último, otro programa diseñado en Python se encargará de cargar los datos del fichero, traducirlos a datos numéricos, guardarlos en matrices y evaluar las diferencias de los datos procesados por la librería wavelet original en Python y la procesada por el diseño hardware simulado en Modelsim. En este último programa se verificará el valor del residuo o error cuadrático, el cual no es demasiado preciso a la hora de estimar el error, y posteriormente se verificará el error haciendo uso de las herramientas del banco de pruebas del que se partió para evaluar el tipo de transformada wavelet óptima. A todo esto es necesario añadir que además de la verificación a nivel RTL llevada a cabo es igual de importante comprobar que el código creado es sintetizable, ya que de no ser así el diseño no podría implementarse en la electrónica programable y sería necesario

realizar las modificaciones oportunas para lograr que el código sea sintetizable y funcione correctamente en el banco de pruebas. En la Fig. 13 se muestra el diagrama de flujo con los distintos pasos en el desarrollo del módulo.

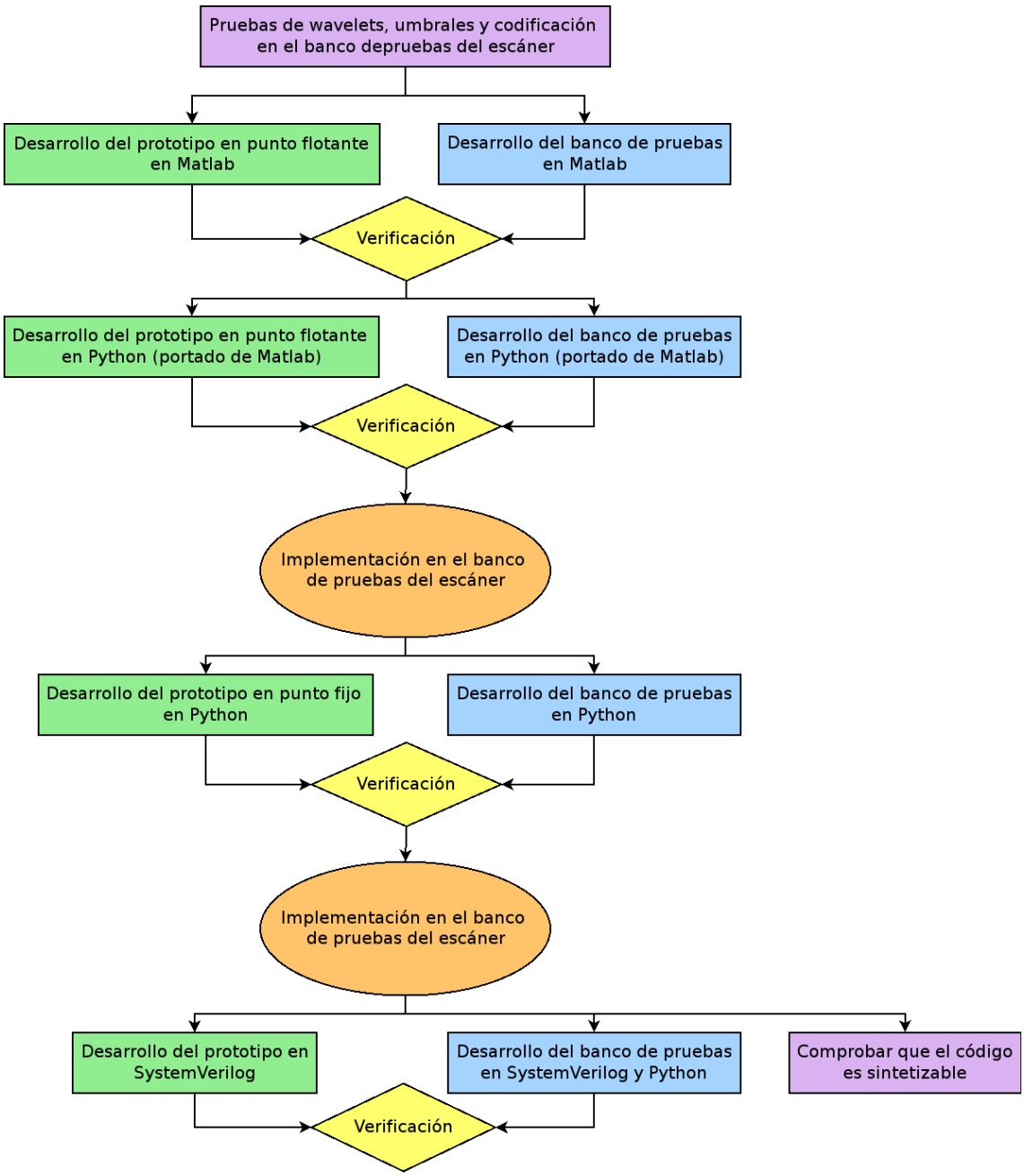

Fig. 13: Diagrama de flujo con los distintos pasos del desarrollo del proyecto

Por último, es necesario aclarar que no se ha llevado una simulación del diseño ya sintetizado debido a que la simulación podría tardar demasiado tiempo, hasta varios días. Es por ello que se ha supuesto que el sintetizador no ha introducido modificaciones, siendo su comportamiento muy similar al comportamiento a nivel RTL.

Capítulo 4. Búsqueda de la wavelet, umbrales y codificación a usar

Como se ha comentado anteriormente, el primer paso para comenzar el diseño consiste en partir del banco de pruebas usado para implementar la verificación del sistema y realizar pruebas cambiando el tipo de wavelet usada, los umbrales y el diezmado de bits hasta lograr una combinación que satisfaga los resultados deseados. Para lograr este fin, y dado que existen múltiples parámetros variables y por ende una gran cantidad de combinaciones, va a definirse un método que permita realizar menos simulaciones y lograr unas prestaciones cercanas a las óptimas. Este método consiste en experimentar en primer lugar con el tipo de wavelet que se implementará, pudiendo ser distintas para cada uno de los dos ejes, por lo que se realizará un barrido primero haciendo uso de la misma wavelet en ambos ejes, en segundo lugar variando únicamente la wavelet de un eje y en tercer lugar haciendo lo propio con el otro eje. Además de lo anterior, es necesario tener en cuenta que el número de coeficientes será de importancia para la implementación hardware, por lo que es conveniente elegir una solución con pocos coeficientes. En segundo lugar, se modificarán tanto los umbrales como el número de bits para tratar de reducir el uso de datos enviados manteniendo el error cometido en valores reducidos.

Para realizar la búsqueda de la wavelet inicialmente se utilizarán más bits para los coeficientes de baja frecuencia (cA), en concreto 12 bits, ya que son los que más información almacenan sobre la imagen del evento. Los coeficientes de frecuencias medias (cH y cV) se codificarán con 4 bits inicialmente, por aportar menor cantidad de información sobre la imágen y por último los coeficientes de alta frecuencia (cV) se codificarán también con 4 bits, al contener menos información sobre la imagen. Gracias a esto será posible identificar aquellas wavelets que son capaces de condensar la mayor cantidad de información sobre las imágenes de los eventos en baja frecuencia, permitiendo así lograr ratios de compresión más elevados.

Antes de comenzar a hablar sobre los resultados obtenidos, es conveniente aclarar que los resultados, a pesar de la similitud que poseen, no son distribuciones gaussianas o normales (mostradas en línea roja discontinua), por lo que las propiedades de éstas no son aplicables. Es por este motivo que para comparar los resultados de las distintas wavelet usadas se va a emplear la media (μ) y la anchura a media altura (FWHM), los cuales permitirán comparar la bondad de las wavelets para esta aplicación en concreto. Estos dos datos se estudiarán sobre la media de

errores y la media de desviación típica tanto del coordenada Z, referido a la altura del evento, como del coordenada φ, referido al ángulo o coordenada acimutal, siendo el radio constante e igual a 160mm y estando los errores expresados en coordenadas cilíndricas.

4.1 Barrido de wavelet en ambos ejes

Como punto de partida para comenzar la búsqueda de esta wavelet óptima se usará una wavelet tipo Haar, la cual posee únicamente dos coeficientes y es, tal y como se describió en la introducción, la más sencilla de todas las wavelets existentes. El resultado de ésta simulación se puede observar en la Fig. 14, donde se puede apreciar que la media de errores tanto de la altura, Z, como del azimut φ se encuentran centradas en 0, presentando una dispersión FWHM en el coordenada Z de 0.17 mm y 0.12 mm en el coordenada φ, presentando por tanto buenas características tanto en este apartado como en lo relativo a la compresión donde arroja un ratio de 0.5487. Sin embargo, en la desviación típica el error se encuentra centrado en -0.234 mm en el caso del coordenada Z y en -0.362 mm en el caso del coordenada φ, con una dispersión FWHM de 0.828 mm en el caso del coordenada Z y 0.617 mm en el caso del coordenada φ. Esto significa que existe una gran variabilidad en la dispersión de los eventos reconstruidos, lo que motiva la búsqueda de una wavelet con mejores características en este apartado.

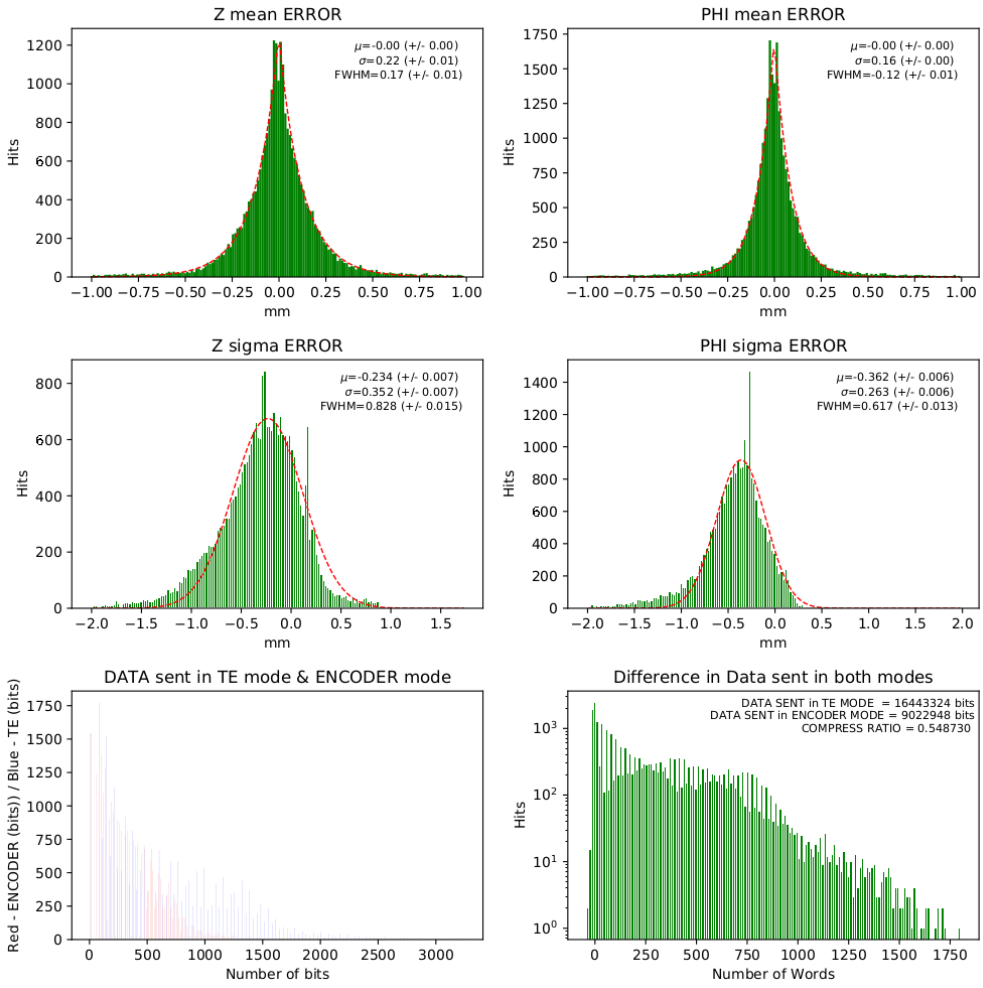

Fig. 14: Resultados del sistema al usar la wavelet haar

Una vez se ha comprobado el comportamiento de la wavelet Haar para las imágenes de los eventos, se puede comenzar a buscar una posible solución alternativa que mejore algunos parámetros, en especial, la desviación típica de los errores. Para ello se van a analizar las wavelets agrupadas por familias, en concreto las familias: Daubechies (db), Symlets (sym), Coiflets (coif), Biortogonal (bior) y Biortogonal Inversa (rbio) descartando la familia Meyer Discreta debido al elevado número de coeficientes, 62 en este caso, lo que provocaría que, a pesar de sus características, no sea ideal para la implementación debido al retardo introducido por el filtro y la cantidad de recursos necesarios para su implementación. El resto de familias, por otra parte, deberán cumplir con ciertas características para considerarse válidas:

- La media y la desviación típica tanto de la altura como del ázimut deben estar centrados lo más próximos al cero para evitar que los eventos reconstruidos tengan desplazamientos sistemáticos, y deben tener una variación lo menor posible, lo que implica que el FWHM sea lo más cercano a cero posible.
- Deben tener un ratio de compresión, final/inicial, no superior a 0.7, de otra forma, la compresión sería insuficiente para satisfacer las demandas del escáner.
- No debe tener un número demasiado elevado de coeficientes, siendo lo ideal que se encuentre por debajo de 12 coeficientes.

La primera de las familias que se va a comprobar es la Daubechies comenzando por la wavelet db2, ya que db1 es exactamente igual a haar y, por lo tanto, posee sus mismas características. En el caso de la wavelet db2, tal y como se observa en la Tabla 1, la dispersión de la media es claramente mayor a la que se produce en la wavelet Haar, así como su ratio de compresión, mejorando únicamente en la menor dispersión en la desviación típica. Por otro lado, las wavelets db3 y db4 empeoran claramente en todos los aspectos, sobre todo en el caso de la última, donde la distribución de los errores posee formas bastante irregulares, lo que implica que el error cometido en cada caso es bastante variable e impredecible. Por estos motivos se ha decidido descartar el uso de esta familia de wavelets para el escáner.

La siguiente familia a examinar es la Symlets, cuyos miembros sym2 y sym3 tienen los mismos coeficientes que db2 y db3, por lo que a priori no parece una opción prometedora. Los coeficientes de las wavelets sym4 en adelante son diferentes, por lo que parece una opción razonable comprobar si esta opción es válida. Sin embargo, a pesar de estas diferencias, la wavelet sym4 muestra un comportamiento bastante deficiente, empeorando en todos los aspectos respecto a una wavelet Haar, llegando incluso a presentar un ratio de compresión superior a la unidad, lo que implica enviar una cantidad de datos mayor a la inicial, lo que sería totalmente absurdo. La siguiente en la familia, sym5, obtiene unos resultados incluso peores, con formas irregulares similares a las observadas en el caso de la familia Daubechies, con lo que queda completamente descartado el uso de este tipo de wavelets.

Otra familia que es necesario valorar es la Coiflets, comenzando en este caso por la wavelet con menor número de coeficientes, coif1, con 6. Esta wavelet, de la misma manera que las anteriormente examinadas, empeora las características iniciales, en este caso, en todos los aspectos, incluyendo el ratio de compresión. La siguiente wavelet de la familia coif2, empeora aún más los resultados, lo que motiva a descartar también el uso de esta familia de wavelets para el escáner.

Por último, se llega al grupo de las wavelets Biortogonales, las cuales están divididas en dos familias, Biortogonales Directas o simplemente Biortogonales, y Biortogonales Inversas. Ambas familias son idénticas con la excepción del uso de los filtros, ya que los filtros usados para la descomposición en la familia Biortogonal, se utilizan en reconstrucción en la Biortogonal Inversa y viceversa, característica que puede cambiar en gran medida el comportamiento de la wavelet, por lo que deberán estudiarse ambas familias por separado, comenzando por la Biortogonal. En este caso, la wavelet bior1.1 posee los mismos coeficientes que haar, por lo que no resulta de interés comenzar por esta wavelet, sino por bior1.3, la cual tiene un comportamiento poco adecuado al diseño, deformando la distribución de errores de la media del coordenada Z y mejorando únicamente en la dispersión de la desviación típica del coordenada φ ligeramente, lo que no compensa el resto de sus inconvenientes. La wavelet bior1.5, siguiendo la misma tendencia, incrementa los inconvenientes de bior1.3, por lo que se decide comprobar si bior2.2 posee unas propiedades más acordes al uso que se le dará. En este caso, de la misma manera que en los anteriores, empeoran todos los parámetros, al igual que en los casos bior2.4 y bior3.1, lo que motiva el descarte de la familia Biortogonal para este diseño.

La familia Biortogonal Inversa por otro lado podría tener un comportamiento bastante distinto al comportamiento de la familia Biortogonal, por lo que merece la pena estudiarla, comenzando con la wavelet rbio1.3, ya que rbio1.1 tiene los mismos coeficientes que haar. En este caso se aprecia una clara mejora en la desviación típica centrada en -0.071 mm y 0.089 mm para los ejes Z y φ respectivamente y cuya dispersión es de 0.522 mm y 0.455 mm respectivamente, bajando además el ratio de compresión hasta 0.5261 a cambio de un ligero aumento de la dispersión en la media del coordenada Z que llega hasta 0.22 mm. Dentro de la familia no es la única con un buen comportamiento, sino que las wavelets rbio1.5 y rbio2.2 parecen mostrar buenas características para esta aplicación. El resto de wavelets de la familia no parecen poder lograr buenos ratios de compresión a pesar de que el resto de características siguen siendo bastante aceptables, como se puede apreciar en el caso de rbio2.6 y rbio3.1, wavelets que se han descartado debido al ratio de compresión logrado, el cual es de 0.7161 y 0.8677 respectivamente, unos valores demasiado elevados para permitir su uso en el proyecto. En el caso de rbio2.2 por otra parte, el ratio de compresión no es tan elevado, lo que permitirá que se tenga en cuenta, aunque, eso sí, bajando el ratio de compresión todo lo posible mediante el uso de umbrales y codificadores.

Tipo de wavelet	Media				Desviación típica				Ratio de compresión
	Z		φ		Z		φ		
	μ	FWHM	μ	FWHM	μ	FWHM	μ	FWHM	
haar	0.00	0.17	0.00	0.12	-0.234	0.828	-0.362	0.617	0.5487
db2	0.07	0.63	0.07	0.25	-0.313	0.816	-0.417	0.532	0.6904
db3	0.13	1.21	0.18	0.46	-0.541	1.292	-0.809	0.900	0.8750
db4	0.37	0.67	0.27	0.62	-0.787	1.801	-1.246	1.340	1.0474
sym4	-0.01	1.18	0.01	0.28	-0.669	1.382	-1.007	0.817	1.0065
sym5	-0.05	1.49	-0.20	0.66	-0.967	1.980	-1.661	1.163	1.3368
coif1	0.02	0.79	0.02	0.17	-0.447	1.101	-0.606	0.670	0.8037
coif2	0.06	1.13	0.13	0.41	-1.149	2.181	-1.850	1.480	1.3337
bior1.3	0.00	1.19	-0.01	0.17	-0.499	0.967	-0.696	0.603	0.7852
bior1.5	0.00	1.48	-0.01	0.38	-0.984	1.876	-1.492	1.003	1.1973
bior2.2	0.01	1.36	0.00	0.23	-0.631	1.525	-0.840	0.972	0.6744
bior2.6	-0.02	2.01	-0.01	0.48	-1.629	2.842	-2.509	1.506	1.4454
bior3.1	0.00	0.82	-0.02	0.54	-2.086	3.665	-1.411	1.375	0.5642
rbio1.3	0.00	0.22	0.00	0.12	-0.071	0.522	-0.089	0.455	0.5261
rbio1.5	0.00	0.22	0.00	0.12	-0.056	0.515	-0.082	0.457	0.5198
rbio2.2	0.00	0.12	0.00	0.02	-0.087	0.576	-0.153	0.373	0.6926
rbio2.6	0.00	0.15	0.00	0.05	-0.093	0.499	-0.154	0.341	0.7161
rbio3.1	0.00	0.60	0.00	0.02	-0.137	0.371	-0.169	0.295	0.8677

Tabla 4.1: Comparativa de distintos tipos de wavelet

De las tres wavelets seleccionadas para verificar su comportamiento, rbio1.3, rbio1.5 y rbio2.2, la más prometedora a nivel de especificaciones es la última, que tiene por único inconveniente de peso, un ratio de compresión algo elevado. Para tratar de atajar este problema se va a reducir el número de datos a la salida bajando los umbrales todo lo posible sin aumentar la cantidad de errores hasta el punto de que la imagen reconstruida no contenga la información deseada. El resultado para un diezmado de 11 bits en baja frecuencia (cA) sin tener en cuenta ni las frecuencias medias (cH y cV) ni las altas frecuencias (cV) y sin usar umbrales se muestra en la Fig. 15. En ésta se puede apreciar que los errores cometidos en el cálculo de la media del

coordenada Z presentan una dispersión exagerada comparada con la wavelet tipo haar y el ratio de compresión apenas baja hasta 0.6674, por lo que esta opción, que inicialmente parecía prometedora, no permite lograr la compresión deseada sin introducir una gran cantidad de errores.

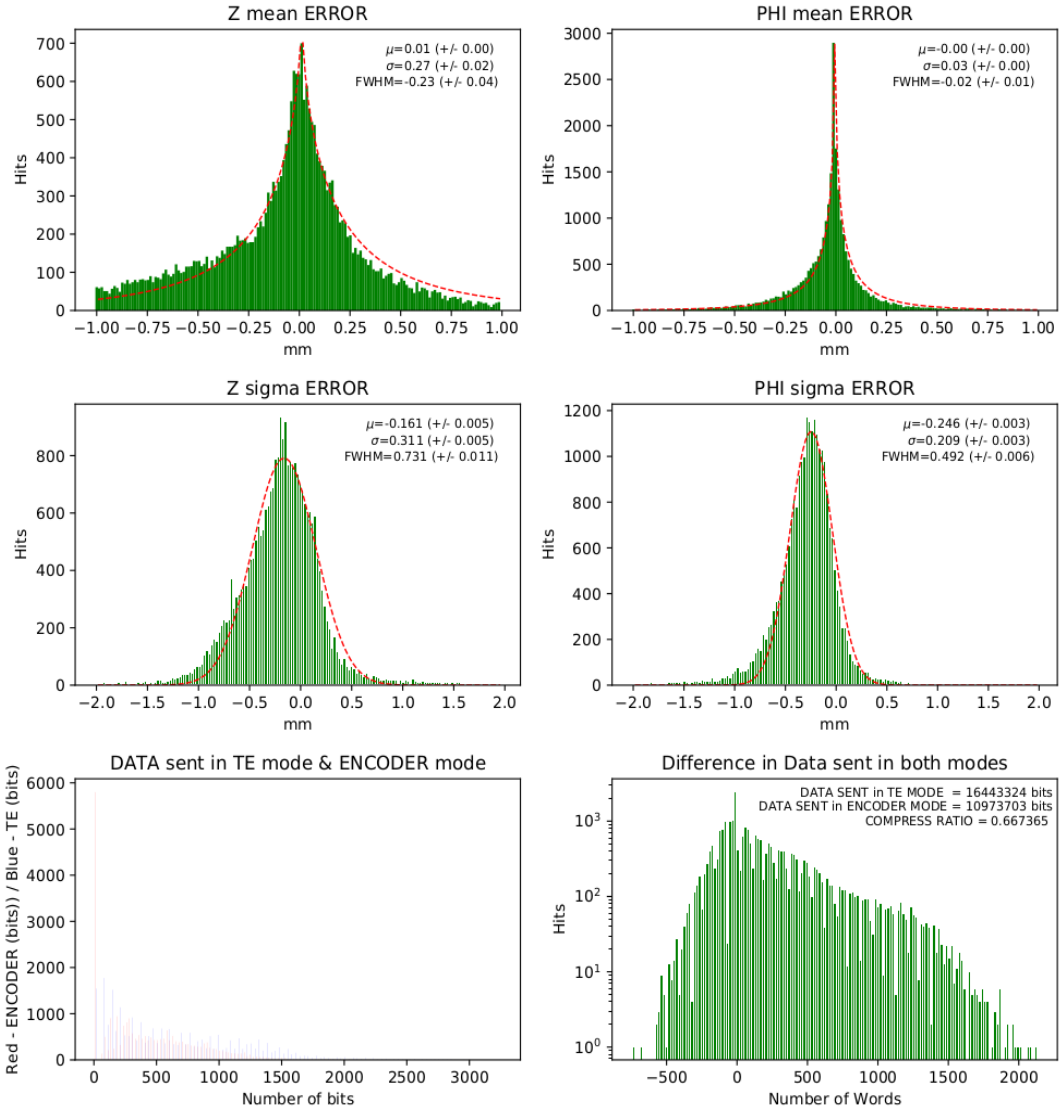

Fig. 15: Resultados del sistema al usar la wavelet rbio2.2 con codificación de 11 bits en baja frecuencia (cA)

Como consecuencia directa del comportamiento de rbio2.2 cuando se trata de reducir la cantidad de datos, y con ello el ratio de compresión, las opciones a valorar son rbio1.3 y rbio1.5, siendo esta última la que obtiene iguales o mejores características en todos los aspectos salvo en la dispersión de la desviación típica del coordenada φ y en el número de coeficientes. El primero de los aspectos es anecdótico y despreciable debido a la diferencia nimia entre ambas wavelets, aunque el número de coeficientes sí resulta más interesante de estudiar. La wavelet rbio1.3 hace uso de 6 coeficientes, mientras que la wavelet rbio1.5 hace uso de 10 coeficientes, por lo que ésta última introducirá algo más de retardo. Para calcular la diferencia entre estos retardos se usará la expresión (4.1), la cual permite calcular el número total de ciclos necesarios para

realizar el filtrado desde que entra la primera muestra hasta que sale la última. Esta expresión se puede deducir atendiendo al hecho de que el filtrado wavelet se va a realizar con extensión simétrica, es decir, al inicio y al final se añaden valores obtenidos al espejar los datos, es decir, añadiendo los valores que aparecen en los datos pero en sentido opuesto. Esto permite evitar los efectos de las discontinuidades en el filtro aunque añade un pequeño retardo, ya que esta extensión ha de realizarse anterior y posteriormente a los datos. Para que esta extensión sea realmente efectiva es necesario que en el instante en el que se introduce la primera muestra de los datos se encuentren en el filtro los valores espejados, lo que añade un retardo igual al número de coeficientes del filtro menos la unidad tanto el la parte anterior como en la parte posterior, lo que justifica la expresión (4.1)

$$nCLK = lData + 2(lH - 1) \tag{4.1}$$

En la ecuación (4.1) “lData” se refiere a la longitud de los datos, “lH” a la longitud del filtro, es decir, al número de coeficientes que posee, y “nCLK” es el número de ciclos que tarda el filtro en obtener la señal a su salida. De esta manera se puede calcular el retardo en cada caso, teniendo en cuenta que la longitud de los datos es de 60 muestras en el caso del eje X y de 16 muestras en el caso del eje Y de la imagen. Los retardos calculados se muestran en la Tabla 2.

Wavelet	Longitud del filtro	Longitud de los datos	Número de ciclos necesario
haar	2	60	62
		16	18
rbio1.3	6	60	70
		16	26
rbio1.5	10	60	78
		16	34

Tabla 4.2: Comparativa de retardos (en ciclos de reloj) producidos por las wavelets haar, rbio1.3 y rbio1.5

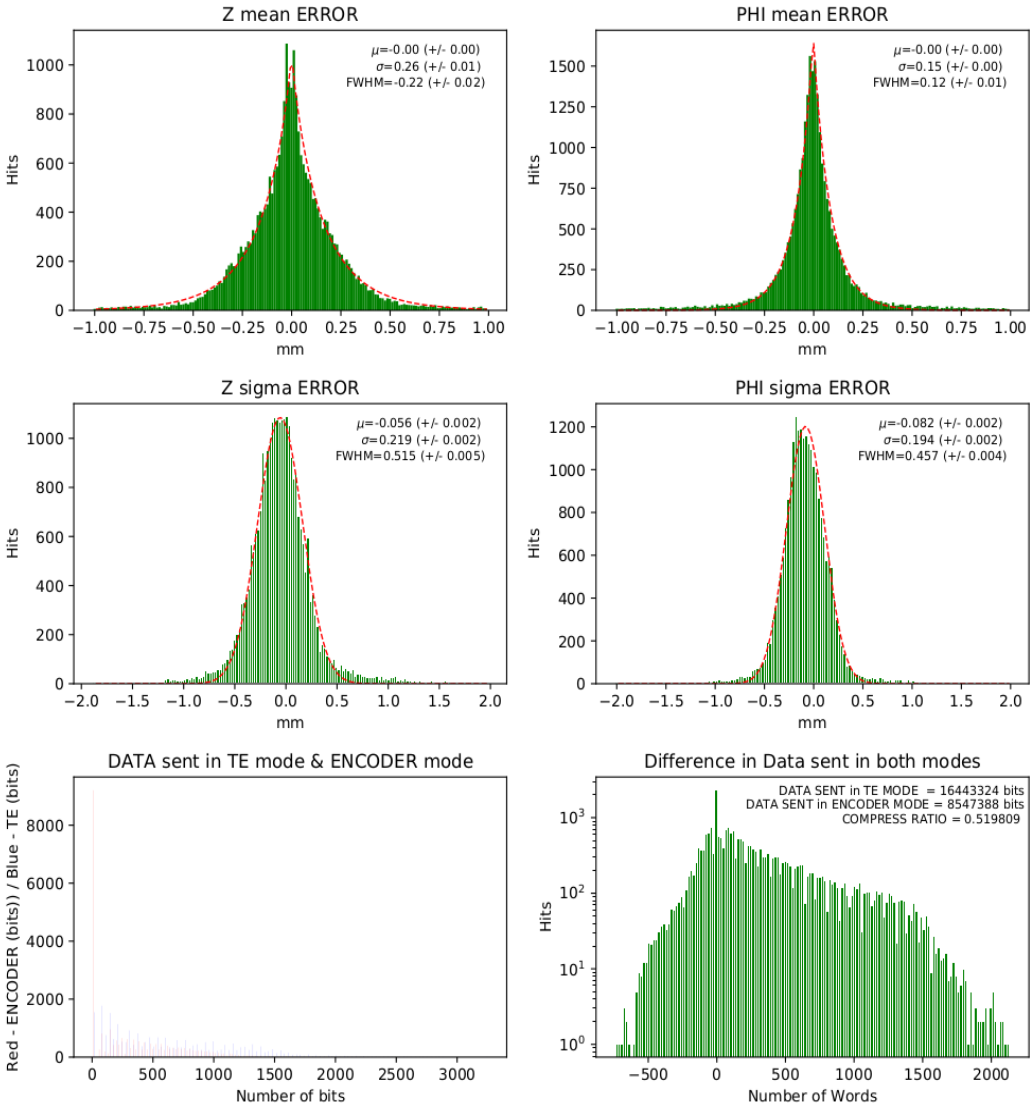

Fig. 16: Resultados del sistema al usar la wavelet rbio1.5

Como se puede apreciar a partir de los datos de la Tabla 2, la diferencia de retardos para ambos ejes es de 8 ciclos más para rbio1.5, lo cual, teniendo en cuenta el número de ciclos necesarios para realizar el filtrado, no supone un incremento exagerado en el retardo. Es debido a esto que la wavelet elegida para implementar el compresor es rbio1.5, cuyos resultados se muestran en la Fig. 16, donde se puede observar las características citadas en la Tabla 1. Con ello se comprueba cómo las distribuciones de errores se encuentran centradas en valores cercanos a cero, lo que significa que la posición de los eventos no se ve alterada de forma sistemática. Además se puede comprobar como, si bien la dispersión de la media de errores del coordenada Z es ligeramente superior a la dispersión que logra una wavelet Haar, la dispersión lograda para la desviación típica tanto en el coordenada Z como en el coordenada φ son bastante más contenidas. A todo lo anterior se añade también el reducido ratio de compresión, el cual es incluso inferior que en el caso de la wavelet Haar, la cual poseía un ratio de 0.5487, mientras que en el caso de rbio1.5, este valor se sitúa en 0.5198. Gracias a todas estas características la wavelet rbio1.5 se postula

como un candidato ideal para su uso en este proyecto, aunque aún es necesario evaluar si puede mejorarse aún más mediante el uso de distintas wavelets para cada eje.

4.2 Barrido de wavelet en ejes distintos

El barrido usando la misma wavelet en ambos ejes de la imagen ha arrojado unos resultados prometedores a nivel de especificaciones, mejorando en la mayoría de aspectos el comportamiento de la wavelet Haar usada inicialmente. Sin embargo, existe la posibilidad de usar wavelets distintas en cada eje, lo que abre la opción de encontrar alguna combinación con mejores características. Para proceder a la búsqueda de esta combinación de wavelets se van a emplear las únicas que han arrojado buenos resultados, es decir, rbio1.3 y rbio2.2 en uno de los ejes combinadas con rbio1.5 como wavelet fija en el otro eje. Estas combinaciones se probarán primero variando la wavelet usada en el eje X manteniendo rbio1.5 en el eje Y y después manteniendo rbio1.5 en el eje X y variando la wavelet usada en el eje Y. El resto de combinaciones no se tendrán en cuenta debido a los malos resultados que han arrojado durante el barrido usando la misma wavelet en ambos ejes, lo que simplifica la búsqueda de la combinación óptima.

4.2.1 Barrido de wavelet en el eje X

En primer lugar, se realizará un barrido de distintas wavelets en el eje X, manteniendo fija la wavelet rbio1.5 en el eje Y. La primera de estas wavelets a emplear será rbio2.2, la cual arrojaba buenos datos en los errores, aunque lograba compresiones demasiado bajas. En este caso, tal y como se puede apreciar en la Fig. 17, este ratio ha bajado hasta 0.5988, aunque sigue siendo alto, lo que unido al empeoramiento de las características de la wavelet para el filtrado en la mayoría de casos no parecen justificar el uso de esta combinación. El único caso en el que mejoran las propiedades de esta combinación respecto a rbio1.5, es en la dispersión de la media del coordenada Z donde los errores de valores cercanos a cero se reducen, aunque aumentan los errores más significativos, lo que motiva a descartar su uso.

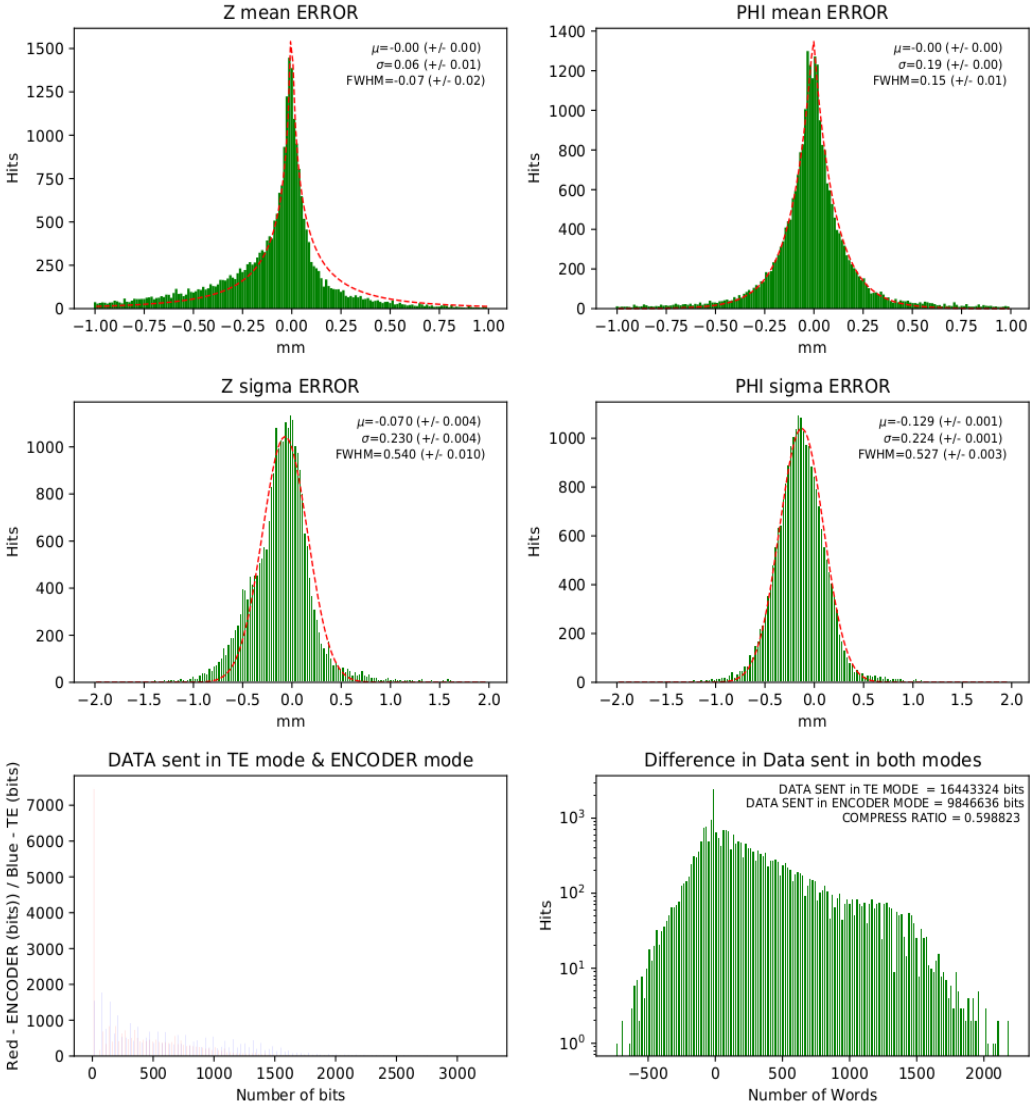

Fig. 17: Resultados del sistema al usar la wavelet rbio2.2 en el eje X y la wavelet rbio1.5 en el eje Y

La otra wavelet que puede emplearse en el eje X por las buenas características demostradas en el apartado anterior es rbio1.3 tal y como se muestra en la Fig. 18. En este caso, el comportamiento es muy similar al mostrado al usar rbio1.5 para ambos ejes, siendo ligeramente mejor en el ratio de compresión, 0.5176 frente al logrado por rbio1.5, 0.5198, y ligeramente peor en la distribución de errores de la desviación típica del coordenada Z. Este empeoramiento se traduce en un centrado en -0.071 mm en lugar de -0.056 mm y una dispersión ligeramente superior, de 0.522 mm en lugar de 0.515 mm. Estas diferencias, casi despreciables tanto en el ratio de compresión como en la distribución de errores de la coordenada Z, provocan que no resulte demasiado interesante el uso de esta combinación con los problemas de implementación que puede acarrear. Es por ello que se ha decidido mantener la wavelet rbio1.5 en el eje X, aunque se investigará si existe una mejor opción en el caso del eje Y.

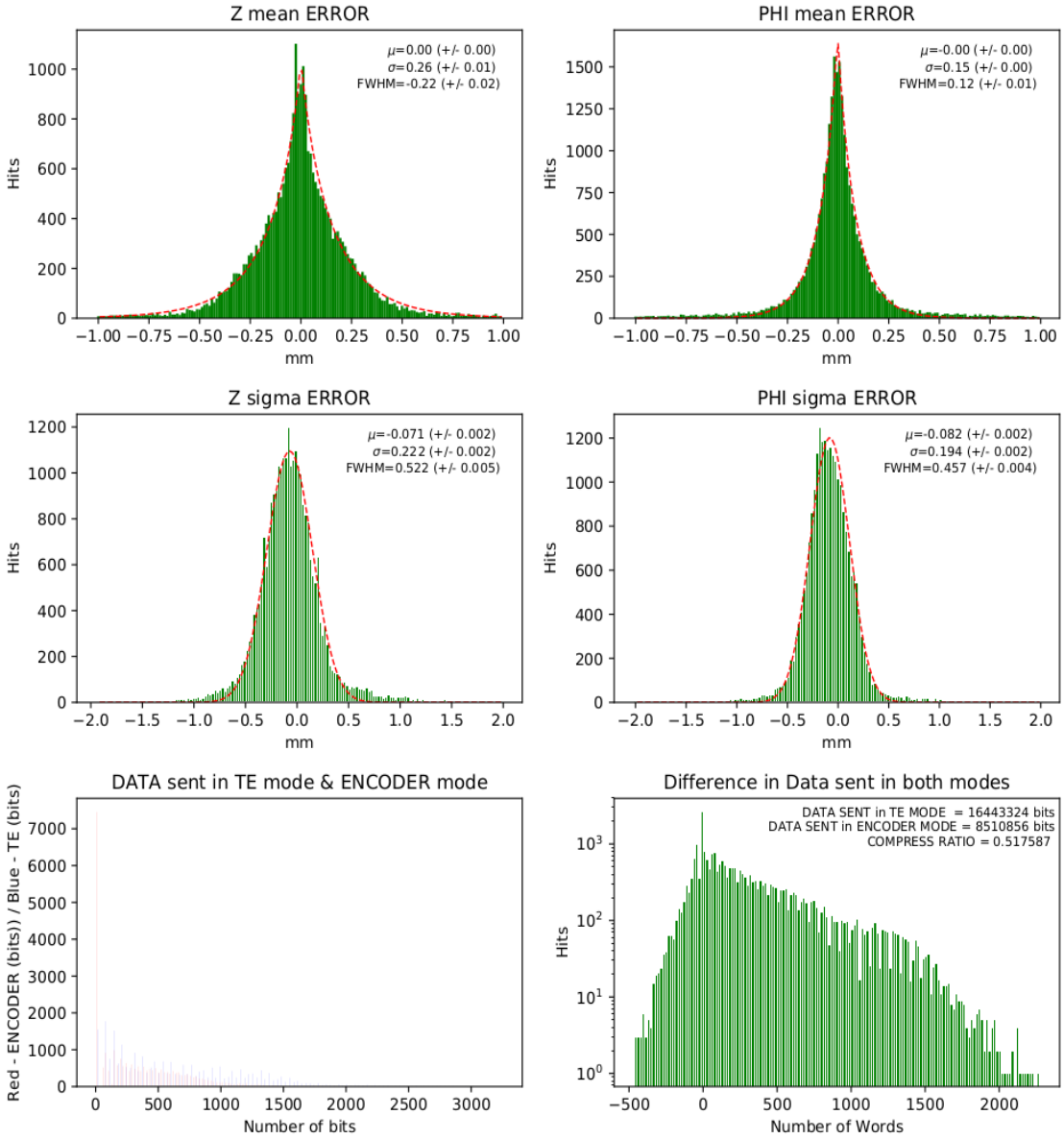

Fig. 18: Resultados del sistema al usar la wavelet rbio1.3 en el eje X y la wavelet rbio1.5 en el eje Y

4.2.2 Barrido de wavelet en el eje Y

Tras fijar la wavelet que se usará para el diseño en el eje X, se va a realizar un barrido de manera similar en el eje Y con la finalidad de estudiar si existe una opción mejor para implementar la transformada wavelet. De la misma manera que para el eje X, se va a comenzar usando la wavelet rbio2.2, mostrada en la Fig. 19 usada junto a la wavelet rbio1.5. En esta imagen se puede apreciar una disminución muy significativa de la dispersión de los errores tanto en la media como en la desviación típica de la coordenada φ, aunque en el caso de la desviación típica aumenta el descentrado, pasando de -0.082 mm a -0.129 mm. Como aspectos negativos aumenta ligeramente la dispersión tanto de la media como de la desviación típica de la coordenada Z, aunque es el gran aumento en el ratio de compresión la característica más preocupante. Este aumento provoca que la compresión corra riesgo de no satisfacer las necesidades del escáner, que requieren un ratio cercano a 0.6 como mínimo. El ratio de

compresión puede verse alterado además, por la implementación hardware, por lo que esta opción no resulta demasiado interesante en este aspecto y se descarta en favor de otras opciones.

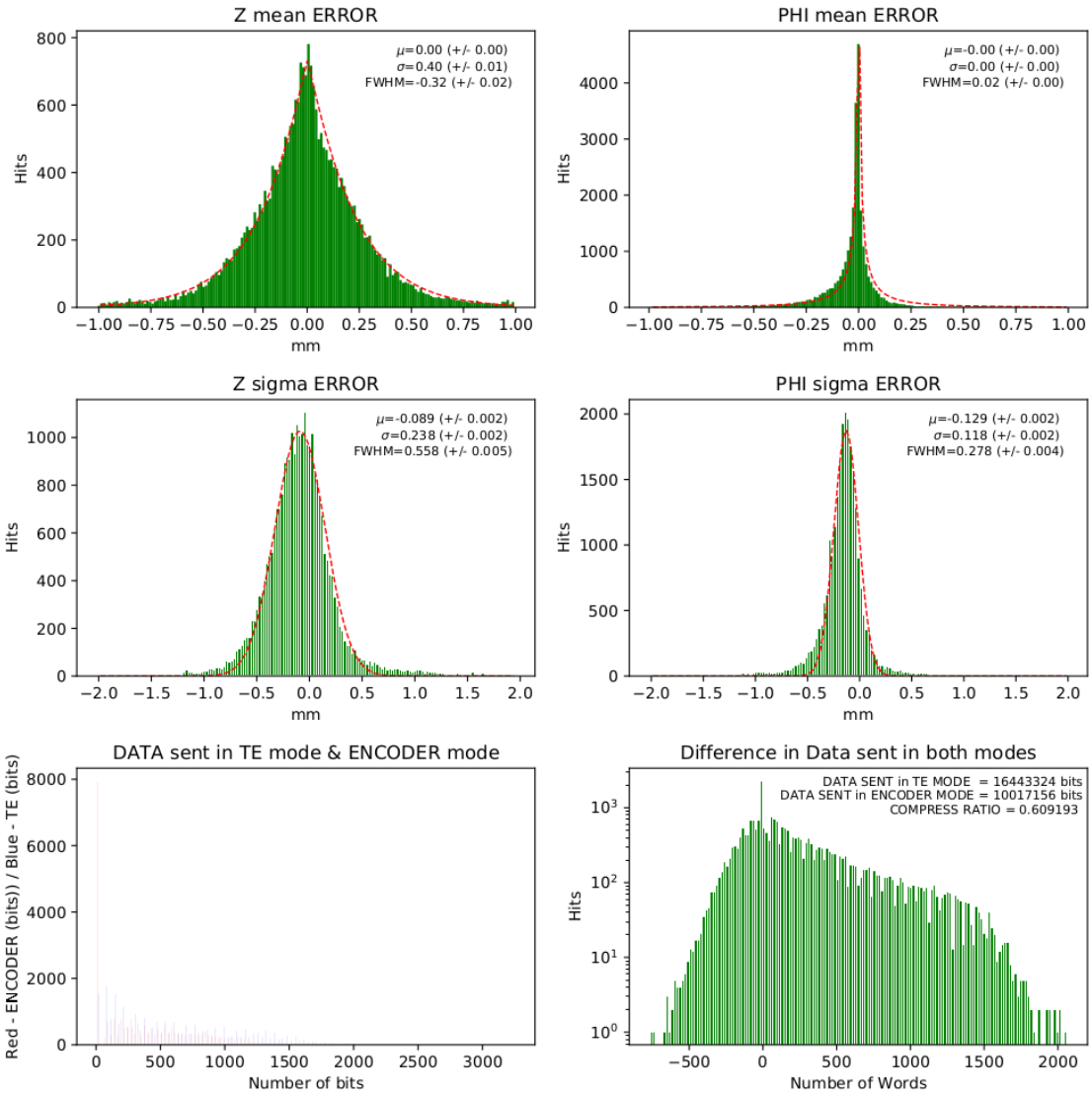

Fig. 19: Resultados del sistema al usar la wavelet rbio1.5 en el eje X y la wavelet rbio2.2 en el eje Y

La última opción es el uso de rbio1.3 en el eje Y, ya que haar, a pesar de contar con sólo dos coeficientes y resultaría ideal para el retardo, no logra unos resultados aceptables, y el resto de wavelets han demostrado un comportamiento nefasto en las pruebas anteriores. Por ello se ha realizado la simulación, mostrada en la Fig. 20, donde es posible comprobar que las diferencias respecto a usar rbio1.5 en ambos ejes se reducen a un ratio de compresión ligeramente más elevado y pequeños cambios sin apenas importancia en la distribución de errores de la coordenada φ.

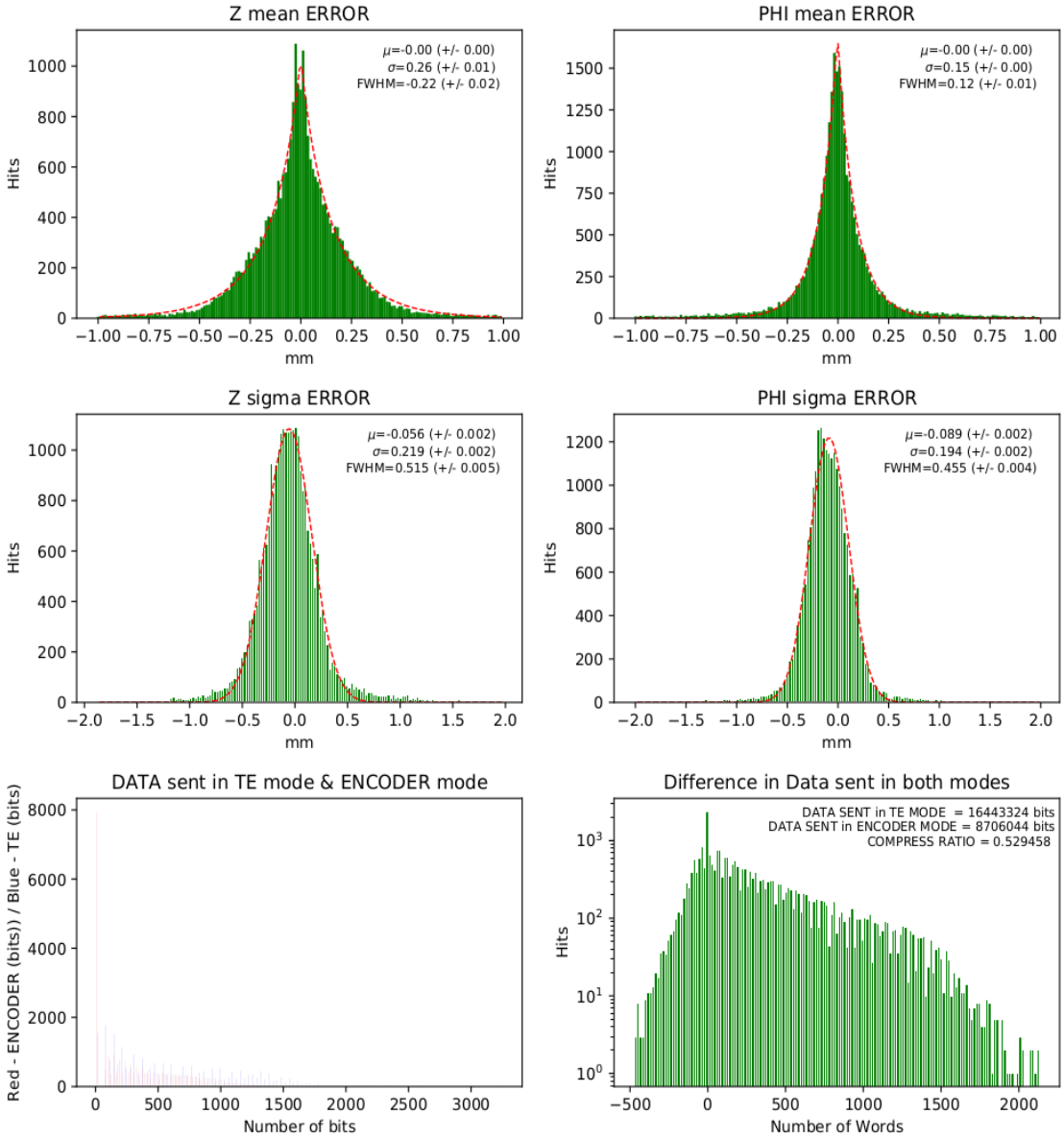

Fig. 20: Resultados del sistema al usar la wavelet rbio1.5 en el eje X y la wavelet rbio1.3 en el eje Y

A la vista de los resultados con distintas wavelets en cada eje se puede concluir que no aportan grandes mejoras apreciables respecto al uso de una para ambos ejes, rbio1.5. Esto por lo tanto, no justifica la complejidad añadida a la hora de diseñar los filtros en el prototipo hardware, los cuales requerirían poder trabajar con una cantidad de coeficientes distintos en cada eje. Es por ello que se ha decidido usar la wavelet rbio1.5 para ambos ejes, dado que se han comprobado sus buenas prestaciones y permite una implementación más simple.

4.3 Búsqueda de umbrales y codificaciones óptimos

Junto a la elección de la wavelet, los umbrales y las codificaciones elegidas para cada uno de los grupos de coeficientes, determinarán el comportamiento y la cantidad de datos enviados. La elección de los umbrales y las codificaciones permitirá realizar pequeños ajustes, los cuales no

serán tan visibles como la wavelet, que es la que más peso tiene en el comportamiento del compresor. Para poder ajustar correctamente umbrales y coeficientes, en primer lugar se ajustarán los umbrales, siempre en exponenciales de 2 para tratar de simplificar la síntesis del hardware, y posteriormente se ajustará el número de bits con el que se codifique cada zona de coeficientes. Esto permitirá tener una primera idea de la importancia de los datos menos significativos, es decir, por debajo del umbral, en la reconstrucción de la imagen para poder ajustar de forma más rápida el número de bits de los coeficientes. Como punto de partida para realizar las simulaciones se van a usar unos umbrales de 0, 256, 256 y 256 y un número de bits de 12, 4, 4 y 4 bits para los coeficientes cA, cH, cV y cD respectivamente.

El objetivo es tratar de concentrar todos los datos enviados en los coeficientes de baja frecuencia, lo que permitiría simplificar el hardware y la codificación de trama empleada. Debido a la codificación de trama actual, es posible que se produzca un ligero aumento del ratio de compresión, aunque esto se solucionará durante la verificación hardware cuando se introduzca una nueva codificación de trama que permita exprimir las características propias del diseño. De lograr un diseño que pueda prescindir de las componentes de media (cH y cV) y alta frecuencia (cD) y usar sólo aquellas en baja frecuencia (cA), se lograría un diseño mucho más simple, por lo que los esfuerzos que se realizarán durante esta fase se enfocarán en este apartado.

Primeramente, para estudiar el efecto de los umbrales sobre los datos, se va a comenzar por la componente de alta frecuencia, cD, la cual representa los detalles diagonales de la imagen. En este caso, el umbral va a incrementar progresivamente, comenzando en 256 tal y como se muestra en la Tabla 3. Como se puede observar, para un umbral de 512 no se produce cambio alguno, de la misma manera que con umbrales de 2048 y 8192, lo que confirma la suposición de que los coeficientes de alta frecuencia tienen un impacto despreciable en los parámetros obtenidos a partir de la imagen.

Tras comprobar que las altas frecuencias tienen influencia despreciable en los datos extraídos a partir de las imágenes de los eventos se va a proceder a estudiar el efecto que tienen las frecuencias medias sobre estos datos. Para ello se va a realizar el mismo procedimiento que con las altas frecuencias, es decir, partiendo de un umbral de 256, se va a aumentar, primero hasta 512 y luego hasta 2048 con el objetivo de comprobar el efecto sobre los datos. Tras realizar las pertinentes simulaciones se han incluido los datos en la Tabla 3, donde se puede observar pequeños cambios entre el uso de los umbrales 256 y 512, siendo los resultados de éste último iguales a los resultados del umbral 2048. Estos resultados parecen indicar que las frecuencias medias poseen poca relevancia sobre los resultados de la simulación, aunque será durante la modificación de los bits empleados para codificar los resultados cuando se puedan sacar conclusiones definitivas sobre estas suposiciones.

Tras ajustar los umbrales y comprobar que, tanto frecuencias medias como altas poseen poca relevancia sobre los datos se va a proceder a ajustar la cantidad de bits empleadas para codificar cada grupo de coeficientes. De la misma manera que en el caso de los umbrales, se va a comenzar por las frecuencias altas, es decir, cA, reduciendo el número de bits con el que se codifican hasta 2. Como se puede observar en la Tabla 3, este cambio no tiene efecto alguno sobre los errores cometidos al recomponer la imagen. Seguidamente se reducirá a 2 bits el grupo de coeficientes cV, dado que en el eje vertical existe una cantidad de información menor, tras lo cual se hará lo propio con el grupo de coeficientes cH, comprobándose en ambos casos que no existe diferencia alguna en los errores cometidos.

Por último se van a reducir los bits de codificación a cero, es decir, se van a eliminar por completo los coeficientes, primero de las altas frecuencias y después de las frecuencias medias. En sintonía con las simulaciones previas, los resultados arrojados demuestran que si se desprecian los coeficientes de alta frecuencia (cA) y de frecuencias medias (cH y cV) y se recompone la imagen a partir de la información de baja frecuencia el error cometido es muy similar al que se obtuvo cuando no se despreciaron por completo estos coeficientes.

Gracias a esta información es posible afirmar que pueden obtenerse los datos a partir de los coeficientes de baja frecuencia de la imagen, cometiendo un error relativamente pequeño, tal y como demuestran los datos de la Tabla 3. Sin embargo, aunque estos resultados son bastante buenos, de hecho, permiten simplificar el diseño en gran medida, puede comprobarse también el efecto que tienen los umbrales sobre los datos de baja frecuencia, así como la posibilidad de usar una cantidad de bits menor para codificar estos datos, lo que permitiría usar una cantidad menor de elementos de la FPGA. Para llevar esta tarea a cabo, primero se va a aumentar el umbral desde 0, hasta 2, lo que permitirá saber si los datos más pequeños tienen importancia en la imagen.

Con este nuevo umbral los errores permanecen inalterables, lo que permite proseguir en la búsqueda de los umbrales y codificación de bits óptimos. En este caso, a diferencia de los anteriores, se va a proceder a reducir la cantidad de bits antes de aumentar el umbral de nuevo para comprobar si es posible usar una cantidad de bits menor. Desgraciadamente, al reducir el número de bits empleados de 12 a 11 se incrementan los errores significativamente, aumentando sobretodo la dispersión de los errores en ambas coordenadas tanto en la media como en la desviación típica. Queda por tanto definido el límite en el número de bits a emplear en los coeficientes de baja frecuencia como 12 bits.

Umbrales Número de bits				Media				Desviación típica			
				Z		φ		Z		φ	
cA	cH	cV	cD	μ	FWHM	μ	FWHM	μ	FWHM	μ	FWHM
0 12	256 4	256 4	256 4	0.00	0.22	0.00	0.12	-0.056	0.515	-0.082	0.457
0 12	256 4	256 4	512 4	0.00	0.22	0.00	0.12	-0.056	0.515	-0.082	0.457
0 12	256 4	256 4	2048 4	0.00	0.22	0.00	0.12	-0.056	0.515	-0.082	0.457
0 12	256 4	256 4	8192 4	0.00	0.22	0.00	0.12	-0.056	0.515	-0.082	0.457
0 12	512 4	512 4	8192 4	0.00	0.22	0.00	0.12	-0.057	0.516	-0.082	0.456
0 12	2048 4	2048 4	8192 4	0.00	0.22	0.00	0.12	-0.057	0.516	-0.082	0.456
0 12	2048 4	2048 4	8192 2	0.00	0.22	0.00	0.12	-0.057	0.516	-0.082	0.456
0 12	2048 4	2048 2	8192 2	0.00	0.22	0.00	0.12	-0.057	0.516	-0.082	0.456
0 12	2048 2	2048 2	8192 2	0.00	0.22	0.00	0.12	-0.057	0.516	-0.082	0.456
0 12	2048 2	2048 2	8192 0	0.00	0.22	0.00	0.12	-0.057	0.516	-0.082	0.456
0 12	2048 0	2048 0	8192 0	0.00	0.22	0.00	0.12	-0.057	0.516	-0.082	0.456
2 12	2048 0	2048 0	8192 0	0.00	0.22	0.00	0.12	-0.057	0.516	-0.082	0.456
2 11	2048 0	2048 0	8192 0	0.00	0.66	0.00	0.17	-0.147	0.651	-0.212	0.555
4 12	2048 0	2048 0	8192 0	0.00	-1.22	0.01	0.45	0.363	1.654	0.702	1.800

Tabla 4.3: Comparativa de distintos umbrales y codificación en número de bits

Tras comprobar que no es posible usar una cantidad de bits menor, se va a intentar reducir la cantidad de datos mediante el uso de un umbral de 4, en lugar 2. Los resultados que se pueden apreciar tanto en la Tabla 3 como en la Fig. 21 son incluso peores que en el caso anterior, lo que implica que se han alcanzado los límites para esta wavelet en cuanto a umbrales y codificación.

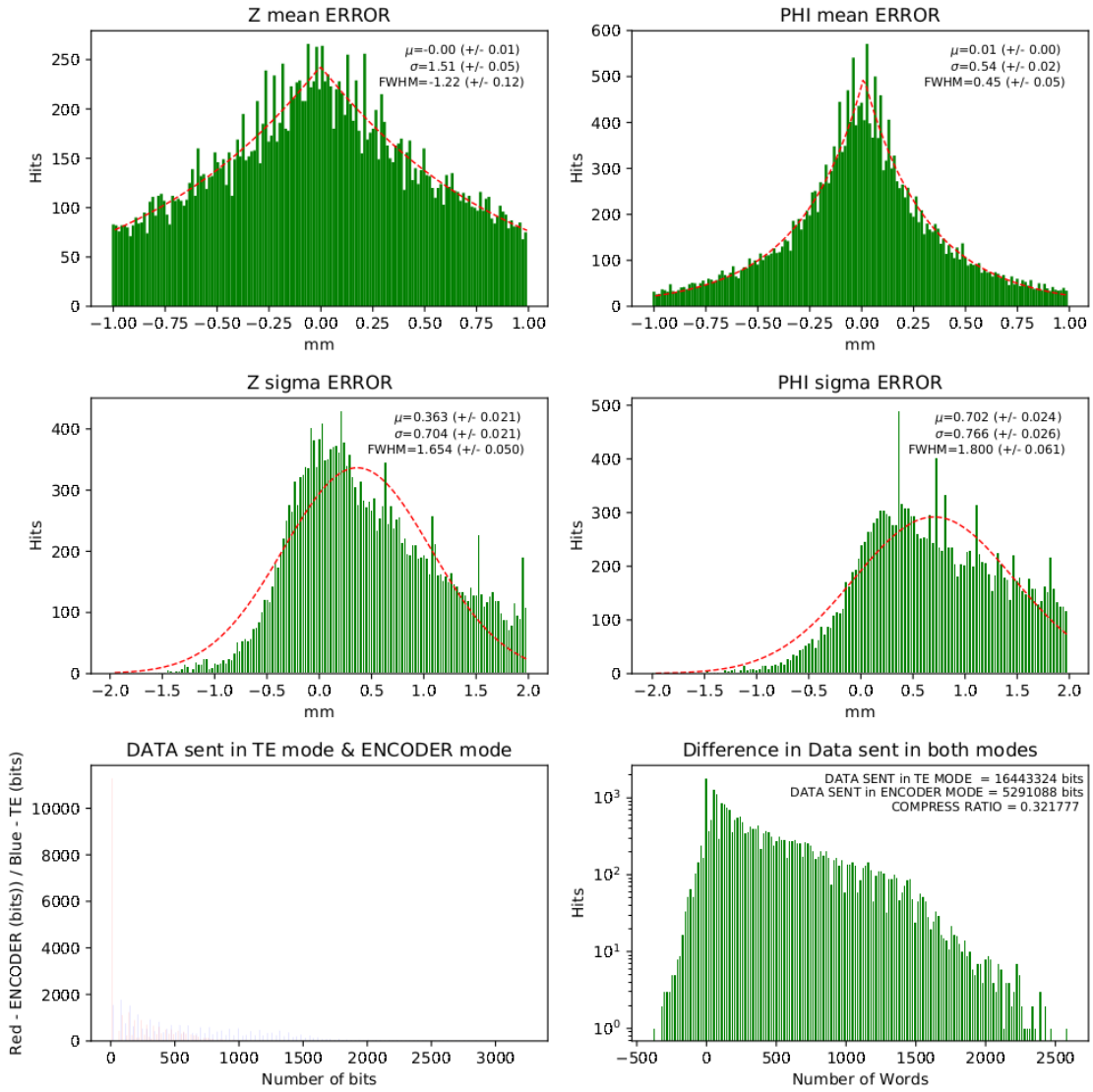

Fig. 21: Wavelet rbio1.5 aplicando un umbral de 4 a baja frecuencia y despreciando los coeficientes de media y alta frecuencia

Conocidos por tanto los límites para esta wavelet, se ha decidido suprimir el umbral aplicado a los coeficientes de baja frecuencia con el objetivo de poder simplificar el hardware. Esto se fundamenta en que los cambios entre el uso de un umbral 2 o un umbral 0 (sin umbral) son despreciables, y sin embargo, el uso de un umbral implica la síntesis de comparadores de bits, los cuales pueden llegar a suponer una cantidad importante de elementos.

Finalmente, tras aplicar todos los cambios especificados en este apartado los resultados quedan tal y como se aprecian en la Fig. 22. Como se puede observar, apenas y hay diferencias con respecto a los umbrales y codificación de partida, aunque en este caso, al emplear solo coeficientes de baja frecuencia se pueden llevar a cabo tres optimizaciones:

- Por una parte se puede emplear una codificación de trama distinta a la actual, lo que permitiría reducir la cantidad de datos enviada y hacer un uso más eficiente de la línea de transmisión. Esta nueva codificación se detallará durante la verificación del hardware.
- Por otra parte el hecho de usar únicamente los coeficientes de baja frecuencia permite simplificar de sobremanera el hardware, pudiendo eliminar los filtros paso-alto del diseño, además de memoria y otros elementos.
- Por último pueden eliminarse los umbrales, lo que se traduce en un ahorro mayor si cabe, de elementos de la FPGA.

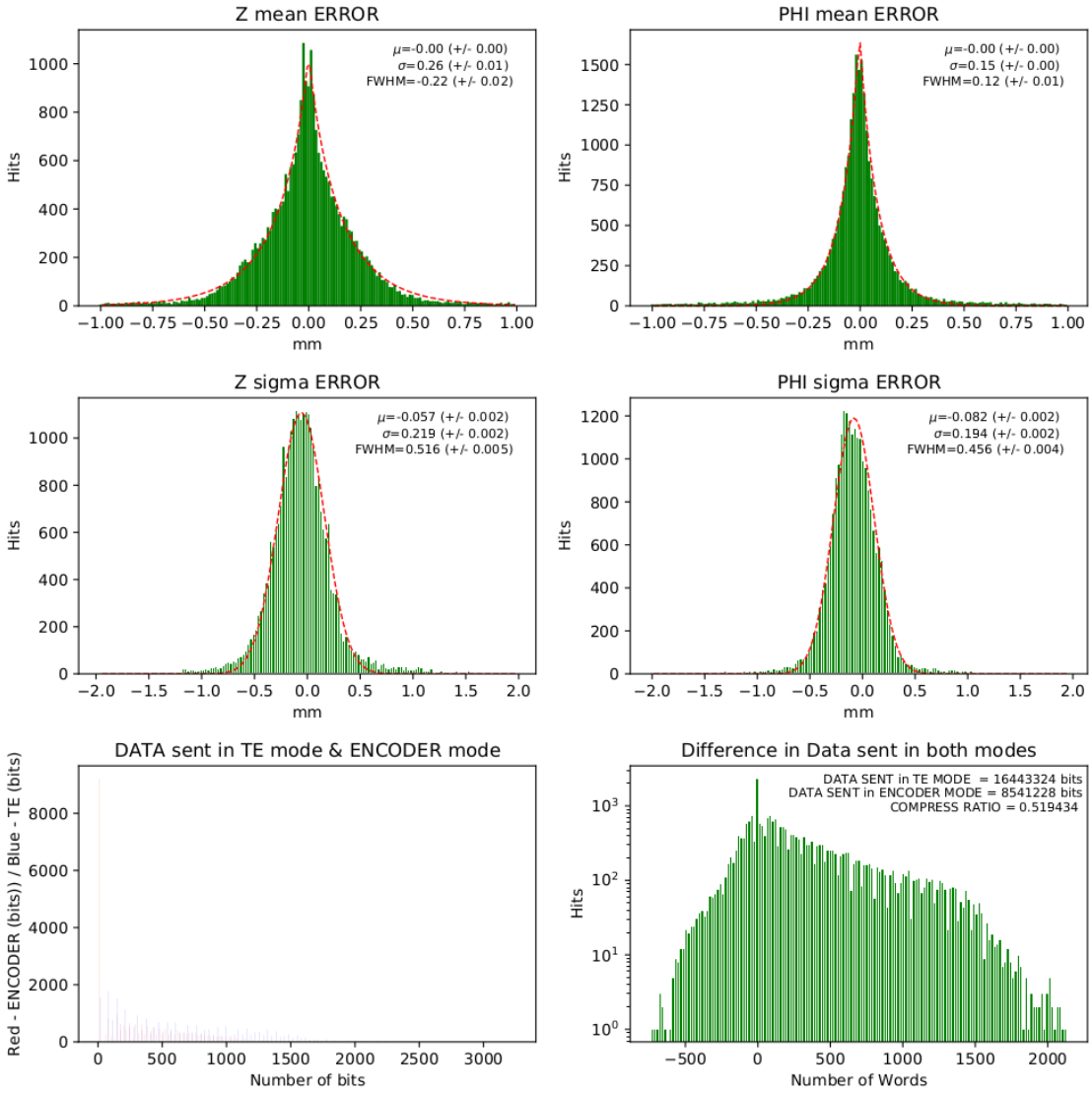

Fig. 22: Wavelet rbio1.5 con los umbrales y diezmado de bits elegidos

Capítulo 5. Diseño del prototipo software

Elegida la wavelet, los umbrales y los bits que codificarán, el siguiente paso en el desarrollo es comenzar el diseño del prototipo software. Este prototipo será el encargado de arrojar los primeros datos sobre la implementación, ya que la implementación hardware tomará ésta como base para su desarrollo, cambiando, eso sí, la forma de trabajo. Para diseñar el prototipo software se realizará en primer lugar una implementación de la DWT mediante funciones, primero en Matlab, para depurar más fácilmente, y después se portará a Python, lenguaje en el que se ha desarrollado el banco de pruebas usado en el apartado anterior. Una vez se haya completado el desarrollo de este modelo y se haya comprobado su funcionamiento mediante pruebas sencillas, se pasará a la implementación en punto fijo, la cual sólo se realizará en Python mediante la librería “spfpm”. Esta implementación tiene como objetivo aproximarse al comportamiento futuro del hardware, ya que éste trabajará en punto fijo debido a la enorme complejidad que supondría implementar las operaciones de la DWT en punto flotante. Por ello, gracias a esta librería se podrá realizar una simulación lo más parecida posible, aunque esto no implicará que no se deba realizar una verificación al hardware, ya que, además de posibles errores cometidos durante el diseño, la librería no permite realizar cálculos en punto fijo, los cuales deberán realizarse en punto flotante y convertirse posteriormente. Debido a esta diferencia, es bastante posible que los resultados de la simulación software difieran de los resultados de la simulación hardware, sobretodo en las operaciones intermedias, aunque darán una primera idea de las dimensiones que deberán tener los componentes del hardware y de los resultados obtenidos.

5.1 Diseño del prototipo software en punto flotante

Antes de comenzar a diseñar el prototipo software es necesario realizar una elección entre las dos posibles implementaciones que tendrá en función de cómo se realice el filtrado:

- El filtrado matricial es la opción más sencilla a nivel estructural, dado que simplemente posee una matriz de coeficientes y unos filtros que permiten realizar el filtrado en columnas y filas simultáneamente. Este sistema sin embargo, presenta la dificultad de implementación, ya que deben implementarse operaciones matriciales sobre la FPGA, algo que de por sí no presenta dificultad, la cual se da en el cableado de datos y la creación de las máquinas de estados finitos. Esta dificultad aumenta el riesgo de error y,

por tanto, el tiempo de diseño, además de precisar más elementos de la FPGA para poder implementarse.

- El filtrado fila/columna es una alternativa al filtrado por matrices, que a cambio de una mayor dificultad en la estructuración, permite que los elementos que la componen sean más simples. Esta implementación se basa en realizar dos barridos, el primero de ellos para filtrar las filas y el segundo para filtrar las columnas. Cada uno de estos barridos aplica un filtrado en una dimensión, lo que reduce la complejidad a transponer correctamente los datos entre ambos barridos para funcionar correctamente. En la Fig. 23 se muestra detalladamente el funcionamiento de este sistema, donde se pueden observar los bancos de filtros (Lo_D y Hi_D) y los diezmadores (1|2) para cada conjunto de coeficientes.

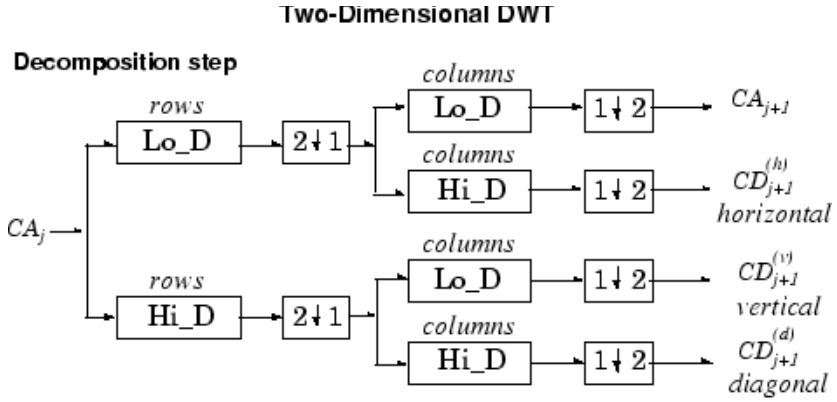

Fig. 23: Detalle del funcionamiento del filtrado fila/columna

De estas dos opciones posibles se ha elegido la segunda, ya que, por una parte no requiere experiencia del diseñador en filtros bidimensionales, más complejos que los filtros unidimensionales, y porque existe una posibilidad de fallo en el diseño menor, lo que reducirá los tiempos de desarrollo del hardware. A todo lo anterior hay que sumar que en casos de imágenes de poco tamaño, como las recogidas por el escáner, que son de 16x60 píxeles, no existe demasiada diferencia de retardos entre ambas implementaciones, siendo ligeramente más rápida la opción de filtrado matricial. También es necesario añadir, que esta implementación es posible gracias a la propiedad de ortogonalidad de la DWT, ya que de otra forma no sería posible esta implementación.

Aclarado este punto, se puede comenzar la implementación, la cual se dividirá en

- El extensor de muestras, es el encargado de extender las muestras a la entrada de forma simétrica para eliminar el efecto de los bordes durante el filtrado.
- El sistema de filtrado, implementado mediante convolución.
- El diezmador, encargado de reducir la cantidad de muestras del filtrado a la mitad.
- El módulo de la Transformada Wavelet discreta (DWT) en una dimensión, formado por los elementos anteriores.

- El módulo de la Transformada Wavelet discreta en dos dimensiones (DWT2), formado por los elementos anteriores.

El primero de los elementos, siguiendo el mismo orden en el que se han diseñado, es el extensor de muestras. Este elemento es el encargado de generar la secuencia de muestras que entrará al filtro a partir de la secuencia de muestras inicial con el fin de eliminar los efectos generados por los bordes, los cuales podrían provocar que la DWT no se comportase de manera correcta. Para lograr esto, se añadirán una serie de muestras tanto en la parte inicial como en la parte final de los datos iniciales de manera simétrica, es decir, colocando los h-1 primeros elementos en orden inverso al principio, y los h-1 últimos elementos en orden inverso al final. Con esto se consigue una continuidad en los datos que evitará que los bordes de éstos afecten al resultado final, lo que permitirá reproducir el mismo comportamiento que en el caso de las librerías usadas durante el apartado anterior. Este elemento, como se puede observar en la Fig. 23, no aparece explícitamente, sino que se sitúa dentro del propio filtro, aunque en este caso se ha decidido separarlo para simplificar el desarrollo.

Una vez extendidas las muestras iniciales, se puede proceder a realizar el filtrado, que será donde se realice propiamente la DWT. El módulo de filtrado, que es el corazón de la implementación y el más importante, será implementado como un filtro FIR, donde los coeficientes que se emplearán son conocidos [13]. La implementación de este filtro en software puede llevarse a cabo de dos formas: por una parte puede implementarse realizando la convolución de las muestras de entrada y los coeficientes del filtro, y por otra parte puede implementarse mediante el uso de la Transformada Discreta de Fourier (DFT). En este segundo caso la operación consistiría en realizar la DFT, multiplicar el resultado por los coeficientes del filtro, y aplicar la DFT inversa (DFT^{-1}) tal y como se ilustra en la Fig. 24. Esta opción es, por tanto, algo más compleja de implementar, y dado que la forma elegida para implementar el filtro en hardware será la convolución, se ha decidido optar por la primera opción para mantener la mayor similitud posible entre el diseño software y el diseño hardware, además de la simplicidad que conlleva este método a costa de la mayor cantidad de operaciones a realizar.

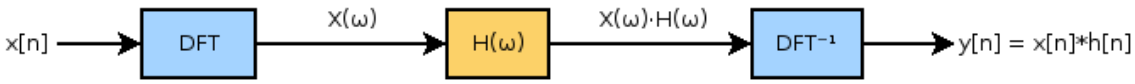

Fig. 24: Implementación del filtrado mediante Transformada Discreta de Fourier (DFT)

Tras atravesar el filtro se realizará un diezmado, el cual eliminará la mitad de los coeficientes. Este diezmado sin embargo, no tiene efecto alguno sobre los datos, más allá de los pequeños errores cometidos por aproximaciones en los cálculos, debido a que existe solapamiento en los datos introducidos al filtro. De esta manera, se reduce a la mitad los coeficientes obtenidos en el filtro sin perder información.

Estos tres módulos son los elementos básicos de la DWT de una dimensión, los cuales se agruparán para poder realizar correctamente la transformada. En concreto, los elementos se agruparán de la manera que se observa en la Fig. 25, usando un único extensor de muestras, que será común a ambos filtros para ahorrar cálculos (software) y elementos (hardware), dos filtros, uno de baja frecuencia y otro de alta frecuencia y dos diezmadores, uno por cada filtro.

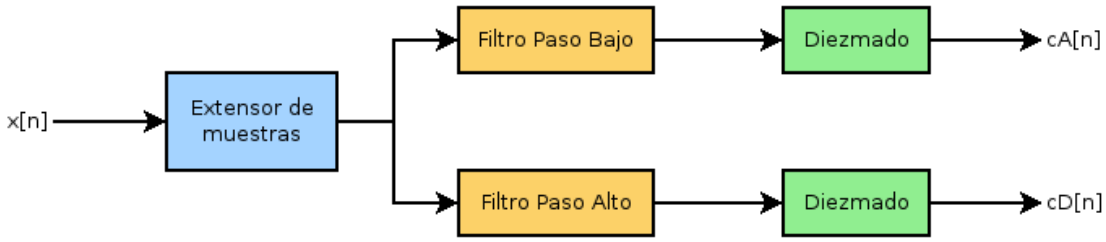

Fig. 25: Implementación del módulo DWT en el prototipo software

Por último, el módulo DWT2 está formado varias llamadas al módulo DWT, tal y como se puede observar en la Fig. 26. Estas llamadas se pueden agrupar en dos partes diferenciadas, por un lado el filtrado en el eje horizontal a la izquierda, y por otra parte están las dos llamadas a la función DWT que componen el filtrado en el eje vertical a la derecha. En ambos casos, los datos introducidos se filtran en un sólo eje y dan como resultado dos grupos de coeficientes: por un lado los coeficientes de baja frecuencia que contienen la forma general de la imagen, y por otro lado se encuentran los coeficientes de alta frecuencia, los cuales contienen información sobre los detalles de contorno de la imagen. Al atravesar varios módulos, la información queda distribuida en coeficientes de baja frecuencia (cA), coeficientes de media frecuencia (cH y cV) y coeficientes de alta frecuencia (cV), siendo los primeros los que contienen la mayor cantidad de información sobre la imagen.

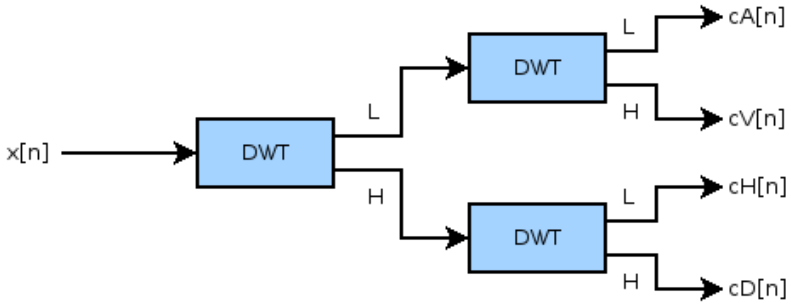

Fig. 26: Implementación del módulo DWT2 en el prototipo software en Matlab

Con estas funciones definidas, se ha llevado a cabo la implementación sobre Matlab, con el objetivo de hacer uso del IDE Matlab para ayudar en la depuración, ya que este entorno es más conocido que cualquiera que permita trabajar con Python como lenguaje de programación. De esta manera, se consigue un proceso de depuración más sencillo a cambio de un esfuerzo algo mayor a la hora de portar las funciones que implementan la DWT de Matlab a Python. Este esfuerzo añadido se traduce básicamente en realizar las funciones nativas en Matlab que puedan

ser necesarias en Python y en adaptar la forma de tratar los vectores, ya que Matlab, a diferencia del resto de lenguajes empleados en el proyecto, utiliza vectores que comienzan en 1 en lugar de comenzar en 0. La primera de las modificaciones necesarias para portar las funciones consistirá en crear una función que permita realizar la transposición de los datos para poder direccionar las direcciones de memoria de la misma manera en ambos barridos y obtener después el resultado final con el formato correcto. Por otra parte, la segunda modificación necesaria se puede solventar de manera sencilla, ya que las funciones desarrolladas anteriormente son válidas con apenas unos pocos cambios en las lecturas y escrituras a la memoria. De esta manera, en la Fig. 27 se muestran los cambios introducidos para portar las funciones de Matlab a Python.

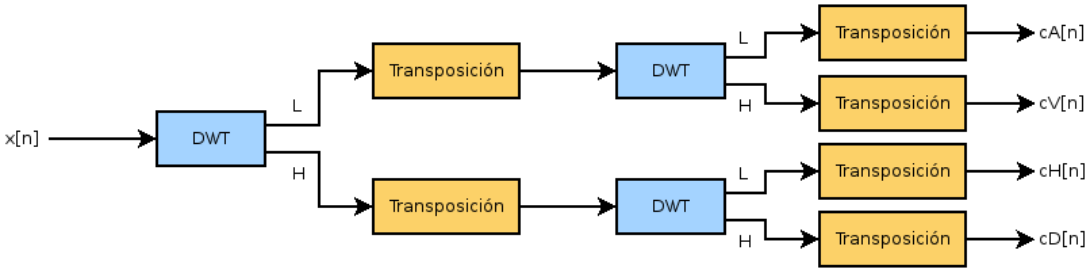

Fig. 27: Implementación del módulo DWT2 en el prototipo software en Python

5.2 Verificación del prototipo software en punto flotante

Terminada la implementación de las funciones para realizar la DWT sin necesidad de emplear la librería nativa, se va a proceder a evaluar el comportamiento de éstas para verificar si cumplen con su cometido correctamente. Para ello, en lugar de emplear el banco de pruebas del proyecto pétalo, que dificultaría en gran medida la tarea de depuración, además de no permitir verificar el funcionamiento en Matlab, se va a emplear un test basado en muestras aleatorias y errores máximos. De esta forma, se introducirán una serie de datos aleatorios a la entrada de la función DWT2, se calculará la IDWT2 mediante la librería nativa en cada lenguaje, y se obtendrá el mayor error de cada imagen al reconstruirla, considerando que el error es aceptable si posee un valor varios órdenes de magnitud inferior al valor de los coeficientes, siendo este error debido a las aproximaciones efectuadas durante los cálculos en punto flotante.

Tras realizar esta sencilla, pero útil comprobación en varias ocasiones para comprobar el funcionamiento y depurar el código, se han conseguido unos errores máximos situados en valores en torno a 1e-13, lo que confirma que la implementación ha sido correcta. De esta manera, se puede proceder, en primer lugar a portar el código de Matlab a Python, y tras realizar las modificaciones pertinentes y depurar el código, se procederá en segundo lugar a sustituir la implementación de la DWT2 en el banco de pruebas del escáner PETALO. Una vez realizado este segundo paso, se comprobará nuevamente el funcionamiento del banco de pruebas para verificar que, efectivamente, el comportamiento no ha variado, tal y como se muestra en la Fig. 28, donde se puede observar que los resultados son exactamente iguales a los resultados de la Fig. 22, la cual hace uso de la librería nativa.

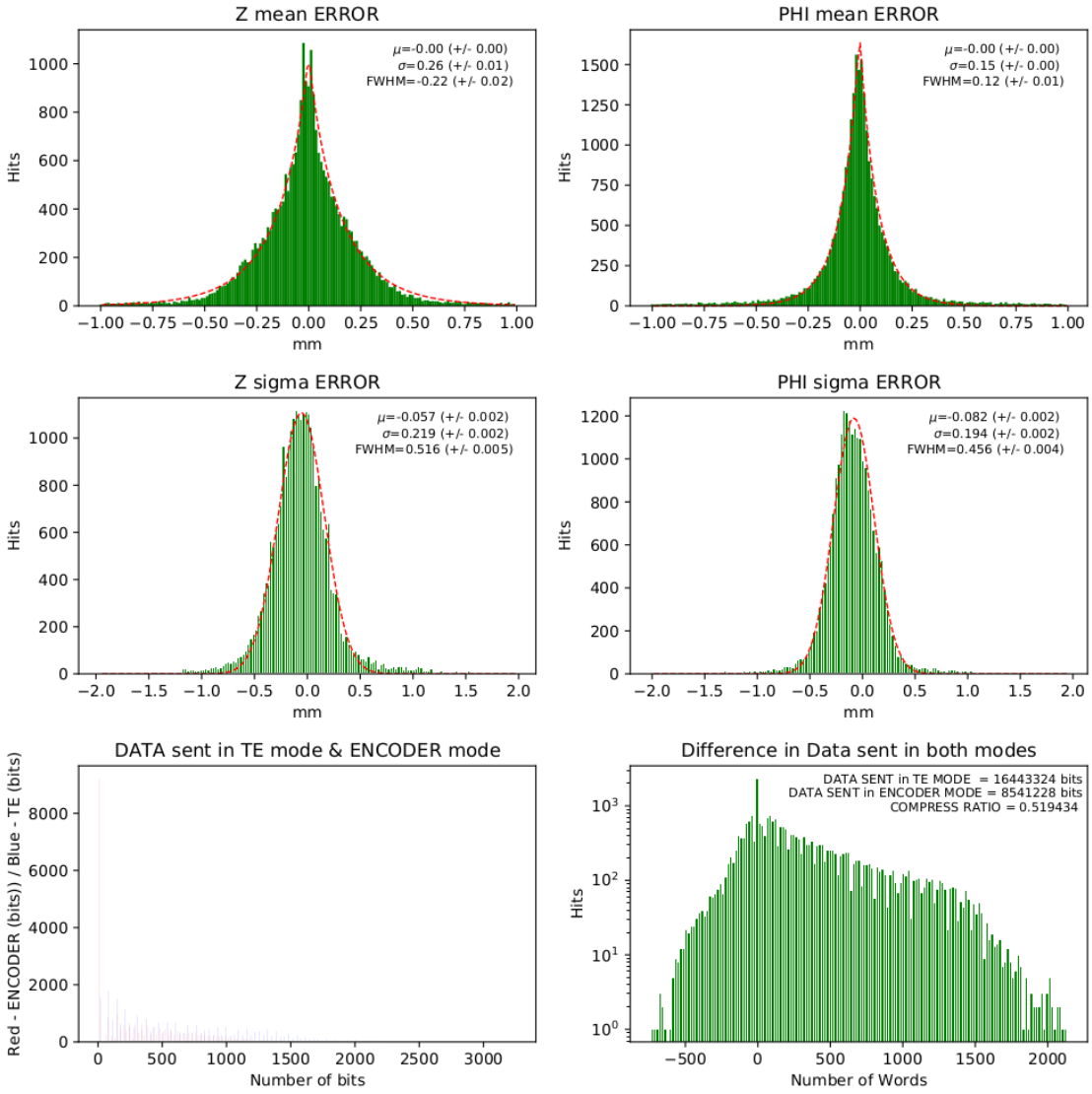

Fig. 28: Resultados del banco de pruebas con la implementación de la DWT2 en funciones.

5.3 Diseño del prototipo software en punto fijo

Una vez se ha realizado la implementación de la Transformada Wavelet Discreta de dos dimensiones (DWT2) en punto flotante y se ha comprobado su correcto funcionamiento, se va a proceder a modificar dicha implementación para añadir la capacidad de trabajar con números en coma fija, en este caso sólo en Python, ya que esta implementación se comprobará directamente sobre el banco de verificación del escáner. Dicha capacidad servirá para comprobar los efectos que tiene la cuantificación de los datos sobre éstos y verificar la cantidad de bits necesaria en cada operando para que el diseño pueda funcionar correctamente. Para llevar a cabo esta tarea se va a emplear la librería “spfpm” en Python, la cual permite trabajar con este tipo de números, eliminando la dificultad de una posible implementación de este tipo de aritmética a cambio de no poder realizar operaciones con distintos tipos de números en punto fijo. Esta desventaja se

traduce en la imposibilidad de estimar el tamaño necesario de los sumadores y multiplicadores que constituyen el filtro hasta la realización de la implementación hardware.

Para llevar a cabo una implementación que permita configurar los distintos tamaños de cada operador se han definido tres familias de número de bits:

- xBits: es el tamaño de los datos de entrada del filtro.
- hBits: es el tamaño de los coeficientes del filtro.
- cBits: son los tamaños de los resultados intermedios (tras realizar multiplicación y suma) y el resultado final. En este caso, se puede configurar un tamaño inicial y un incremento en cada etapa para tener distinto número de bits y evitar desbordamientos sin necesidad de añadir más bits de los necesarios.

Todos estos tamaños son configurables desde el archivo JSON que permite configurar la simulación, siendo el propio banco de pruebas el encargado de cargar y procesar este archivo. De esta forma se puede realizar la configuración de todos los parámetros de la simulación desde un único archivo y se mantiene la coherencia con el resto del banco de pruebas.

5.4 Verificación del prototipo software en punto fijo

Por último, en relación al prototipo software, se ha realizado la verificación de funcionamiento de la implementación con cálculos en punto fijo para comprobar el funcionamiento y obtener una primera estimación del tamaño necesario para cada operando. Como punto de partida se ha decidido usar 14 bits para todos los datos, repartidos en 6 bits para la parte entera (incluyendo el signo) y 8 bits para la parte decimal, usando un incremento de 1 bit por etapa para los datos intermedios, lo que se traduce en una salida de 24 bits de resolución. Estas resoluciones serán previsiblemente menores tras realizar una serie de simulaciones para comprobar el comportamiento del compresor, comenzando por los coeficientes del filtro, los cuales podrían prescindir de todos los bits enteros salvo uno para indicar el signo. Esta suposición se verifica para todos los resultados obtenidos en las simulaciones, cuyos datos más significativos se representan en la Tabla 5.1, donde se puede observar que un único bit en la parte entera de los coeficientes es suficiente para realizar los cálculos correctamente, quedando por especificar el número de bits de la parte decimal.

Además de modificar el número de bits de los coeficientes del filtro, será necesario ajustar también el número de bits usados por los datos de entrada y cálculos. En el caso de los datos de entrada, dado que son de 12 bits, previsiblemente bastará con esta cantidad de bits, correctamente distribuidos entre enteros y decimales, aunque se ha preferido disminuir esta cifra progresivamente por si existe algún caso en el que los datos ocupen más bits. De esta forma, primero se va a comprobar si 13 bits, 5 en la parte entera y 8 en la parte decimal, son suficientes

como para poder realizar el filtrado, tal y como se muestra en la Tabla 1. En este caso se puede comprobar cómo los resultados son relativamente buenos, aunque atendiendo a la Fig. 29 se puede observar cómo se produce una dispersión de los errores algo mayor, fruto de un desbordamiento producido por el truncamiento de los datos de entrada. Es por ello que se estima oportuno continuar con 5 bits en las pruebas para comprobar si esta suposición es correcta, la cual se termina confirmando en todas las simulaciones en las que se utilizan 5 bits en la parte decimal, por lo que se estima oportuno aumentar este número a 6 bits para reducir estos problemas y obtener un mejor comportamiento.

Tamaño (bits) de los datos			Media				Desviación típica			
enteros-decimales (aumento)			Z		φ		Z		φ	
x	h	y	μ	FWHM	μ	FWHM	μ	FWHM	μ	FWHM
6-8	6-8	6-8 (0-1)	0.00	0.16	0.00	0.12	-0.042	0.507	-0.33	0.470
5-8	1-8	5-8 (0-1)	0.00	0.12	0.00	0.13	-0.038	0.577	-0.002	0.540
5-7	1-8	6-8 (0-1)	0.00	0.16	0.00	0.12	-0.042	0.507	-0.033	0.470
5-7	1-8	6-8 (0.5-0.5)	0.00	0.17	0.00	0.12	-0.035	0.489	-0.044	0.450
5-7	1-8	5-8 (0.5-0.5)	0.00	0.15	0.00	0.12	-0.038	0.485	-0.041	0.450
5-7	1-7	5-7 (0.5-0.5)	0.00	0.15	0.00	0.12	-0.038	0.485	-0.041	0.450
5-7	1-7	5-7 (0-0)	0.00	0.15	0.00	0.13	-0.036	0.577	-0.001	0.540
5-6	1-6	5-6 (0.5-0.5)	0.00	0.14	0.00	0.12	-0.034	0.507	-0.020	0.473
6-6	1-6	6-6 (0-0)	0.00	0.16	0.00	0.12	-0.064	0.518	-0.063	0.476

Tabla 5.1: Comparativa de distintos tamaños para xBits, hBits y cBits

El número de bits en la parte decimal, sin embargo, ha demostrado no ser tan decisivo a la hora de cuantificar los datos de entrada, pudiendo bajar el número de bits dedicados a ésta hasta 6 bits sin introducir errores significativos, emplear menos, sin embargo, causa errores demasiado elevados como para que esa solución pueda tomarse como óptima.

Los datos intermedios del filtro en principio deberían necesitar una cantidad mayor o al menos igual a los datos de entrada para evitar introducir errores muy significativos, por lo que la cantidad de bits empleados tanto por los datos intermedios como por el resultado final, se han disminuido paulatinamente siendo en todo caso iguales o superiores a los bits empleados para los datos de entrada del filtro. Finalmente, se ha comprobado que 12 bits, 6 para la parte entera y 6 para la parte decimal, funcionaban correctamente sin introducir errores demasiado significativos ni deformar la distribución de errores de forma apreciable.

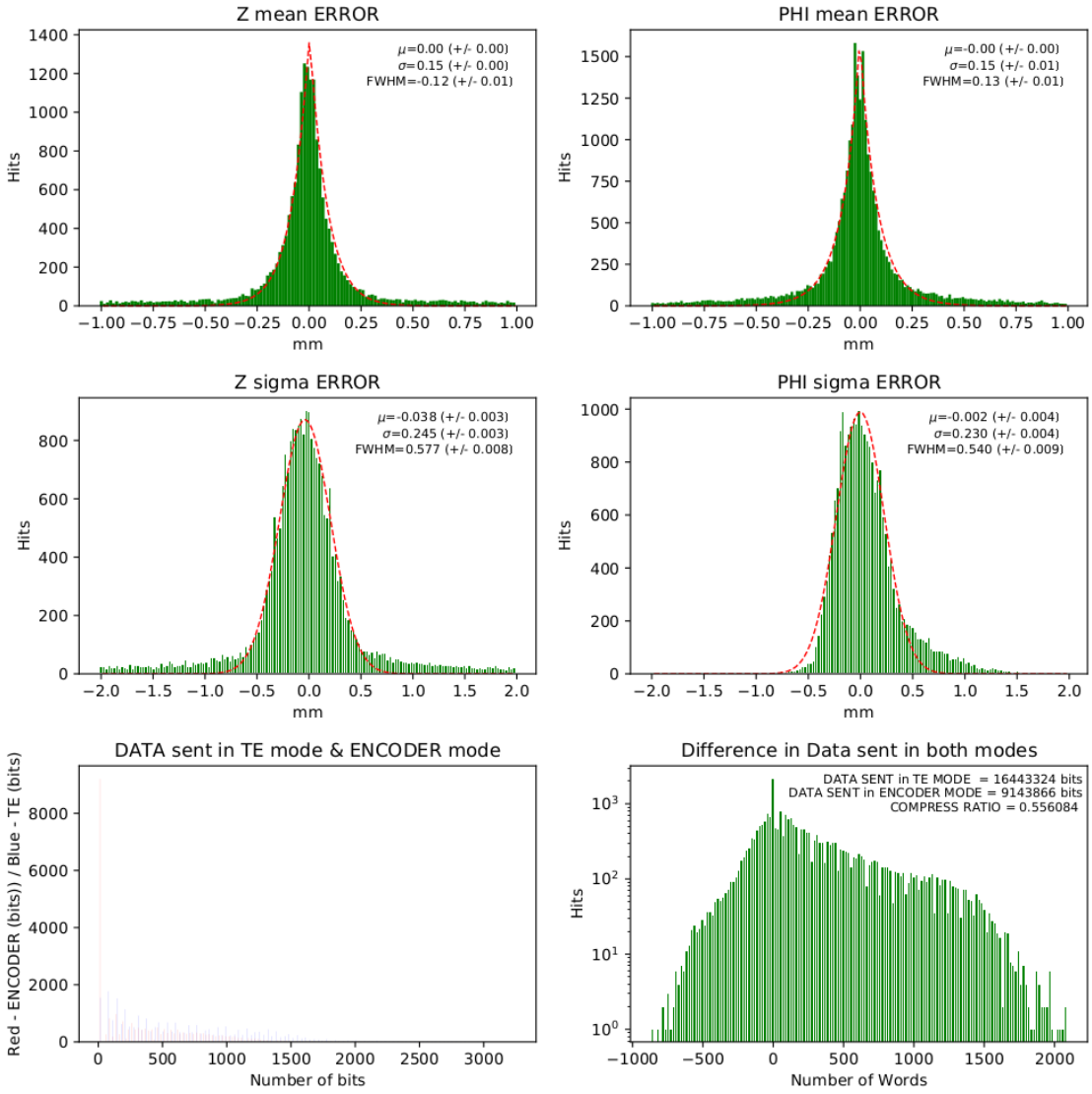

Fig. 29: Distribuciones de errores para 13 bits en los datos de entrada, (5 enteros y 8 decimales), 9 bits en los coeficientes del filtro (1 entero y 8 decimales) y 13 bits en cálculos intermedios (5 enteros y 8 decimales), con un incremento de 1 bit decimal en cada etapa del filtro, y 23 bits en los datos de salida (5 enteros y 18 decimales).

En conclusión, se ha decidido usar 12 bits tanto de entrada de datos como de cálculos intermedios y de salida, repartidos en 6 bits para la parte entera y 6 bits para la parte decimal, mientras que en los coeficientes, al carecer de parte entera, se ha dejado un único bit para contener el signo en esta parte y 6 bits para la parte decimal, obteniéndose los resultados mostrados en la Fig. 30. De esta manera se evita el efecto de la mayor dispersión de errores en la desviación media de la coordenada Z y la pérdida de simetría en la desviación típica de la coordenada φ, tal y como se muestra en la Fig. 29. Con esta medida se aumenta ligeramente la cantidad bits a cambio de eliminar estos artefactos que pueden afectar seriamente a la obtención de los datos relevantes en el sistema de decodificación.

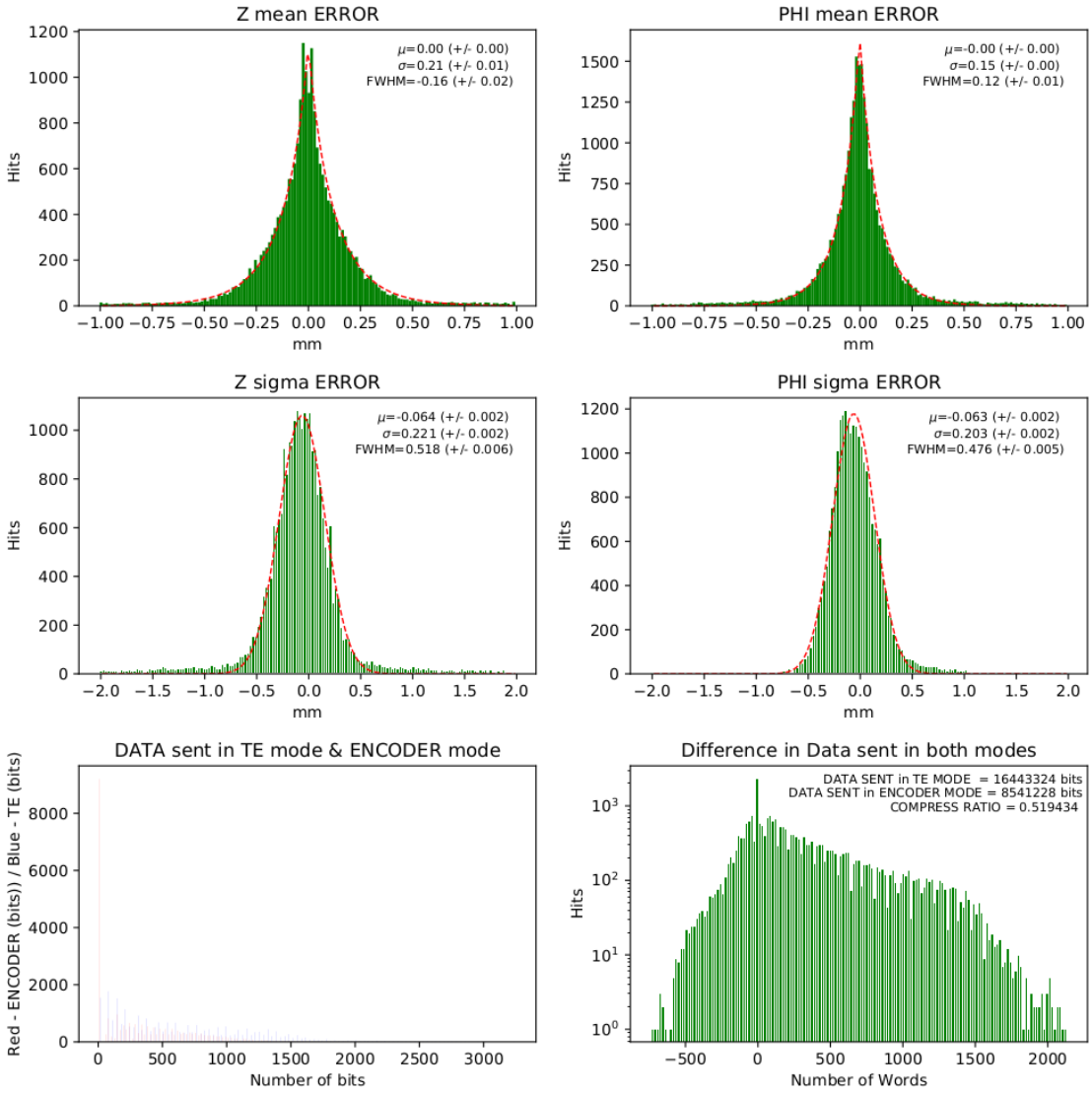

Fig. 30: Resultados de la simulación con los datos y coeficientes en punto fijo para los tamaños elegidos

Finalmente queda por comentar la aparente mejora en el funcionamiento del sistema al usar cálculos en punto fijo, la cual probablemente sea debida a los truncamientos, los cuales pueden desplazar ligeramente los resultados, consiguiendo leves mejoras en algunos aspectos como la menor dispersión de la media de la coordenada Z, y empeorando ligeramente otros aspectos como la dispersión de la desviación típica tanto de la coordenada Z como de la coordenada φ, aunque sin afectar en ningún caso excesivamente a las distribuciones de errores originales.

Capítulo 6. Implementación hardware del sistema

La implementación hardware del sistema de compresión es la que cierra el apartado de diseño iniciado con la elección de la wavelet a emplear, y continuado con el desarrollo del prototipo software, en el cual se basará la implementación hardware. Esta implementación se desarrollará de manera distinta al prototipo software, ya que en este último la metodología de diseño fue top-bottom, es decir, se partió del funcionamiento de la Transformada Wavelet Discreta de dos dimensiones (DWT2) en alto nivel para dividirla primero en funciones DWT y transposiciones, y depués dividir la DWT en extensión de datos, filtrado y diezmado. Este método fue tremendamente útil a la hora de desarrollar y estructurar las funciones que realizan cada una de estas tareas, aunque en el caso de la implementación hardware se ha preferido utilizar una metodología de diseño bottom-up, aprovechando que la estructura del diseño será muy similar a la del prototipo software. Las ventajas asociadas a esta metodología de diseño son varias, por una parte permite verificar cada uno de los módulos por separado antes de realizar la implementación de la DWT2, también permite comprobar que el diseño es sintetizable y no existen elementos indeseados como latches, además de ofrecer la posibilidad de definir parámetros en los módulos que permitan configurar el diseño desde el tope de jerarquía o desde el propio banco de pruebas. Esto trae como consecuencia una depuración de errores más rápida y sencilla en pequeños bancos de simulación, que son mucho más simples que el banco de verificación del módulo que implementa la DWT2, y cuyo único fin es dar el visto bueno de forma visual a cada módulo sin entrar en detalles como la cuantificación, la cual se tratará en el banco de verificación de la DWT2. Éste último será el encargado de comprobar detalles del diseño como el funcionamiento del módulo completo, los efectos que tiene la cuantificación sobre los datos de las imágenes reconstruidas y el efecto visual que tiene aplicar la DWT2 sobre las imágenes de los eventos. Será durante el apartado de verificación del diseño donde se abordarán estos temas más en profundidad, mientras que en este apartado se tratará sobre el diseño de cada uno de los módulos que componen el diseño hasta llegar al tope de jerarquía, siguiendo la misma metodología que se ha empleado para el diseño.

6.1 Implementación del extensor de muestras

El extensor de muestras tiene como objetivo eliminar los efectos de bordes producidos por el filtro. En filtros digitales, para poder realizar una convolución lineal, es necesario añadir una serie de muestras a los datos que serán filtrados ya que de otra forma se realizaría una

convolución circular, la cual no es útil para realizar el filtrado. Generalmente estas muestras son ceros que se añaden antes de los datos, lo que suele denominarse "zero-padding". En la transformada wavelet sin embargo, la extensión que se realiza consiste en extender simétricamente los datos tanto al inicio como al final, lo que resulta más complejo en la implementación, pero elimina el efecto de bordes que se produciría de usar relleno con ceros.

Los elementos que componen el módulo se ilustran en la Fig. 31, los cuales están conectados mediante buses de datos (en color negro) y buses de estado y control (en color azul). A pesar de la aparente complejidad del módulo su funcionamiento es bastante simple, en primer lugar, se inicializa el contador de datos de salida con la dirección de memoria inicial y con el sentido inicial de la cuenta, ascendente o descendente. Una vez inicializado, comienzan a llegar datos a la memoria RAM, los cuales se escriben secuencialmente gracias al contador de datos de entrada. Los datos de salida, por otra parte, no pueden salir inmediatamente, sino que deben esperar a que la dirección de memoria inicial se encuentre almacenada en la memoria RAM, independientemente del sentido inicial de la cuenta, ya que la salida siempre irá un ciclo por detrás de la escritura en memoria, y por lo tanto, el dato a la salida siempre se encontrará almacenado en la memoria. Una vez la primera muestra a la salida se encuentre almacenada en la memoria, comenzarán a salir los datos hasta que el contador de datos de salida termine de realizar la extensión, momento en el cual el módulo volverá al estado de espera, en el que se mantendrá hasta volver a recibir la señal "START" para iniciar una nueva extensión con otros datos. Toda esta coordinación se lleva a cabo mediante una máquina de estados, la cual es de tipo Moore por ser la más efectiva y simple para este diseño.

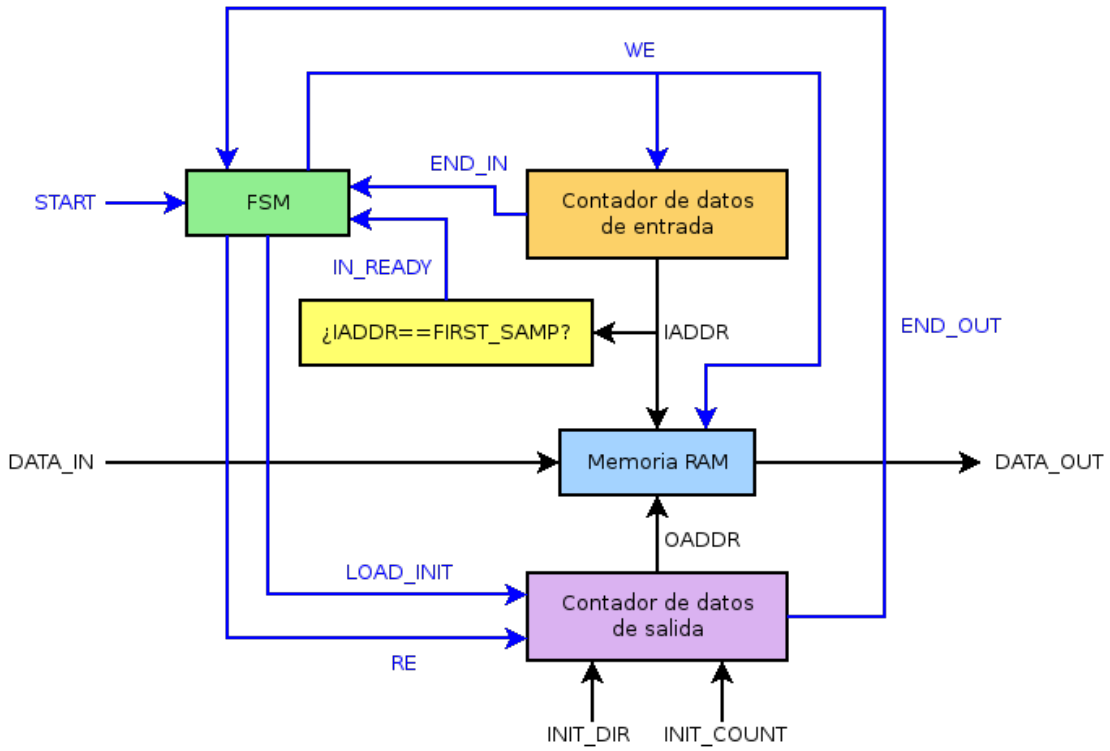

Fig. 31: Diagrama de bloques del extensor de muestras

Es necesario destacar que el contador de datos de salida es, en realidad, un módulo formado por dos contadores distintos. Uno de los contadores es un contador de la misma naturaleza que el contador de muestras de entrada y únicamente funciona en modo ascendente, siendo utilizado únicamente para definir la señal de fin de extensión. El segundo contador permite funcionar en modo ascendente o descendente, alternando entre estos dos modos cuando se llega a final de cuenta y pudiendo, además, comenzar la cuenta en un valor y dirección concretos. Este segundo contador es el encargado de realizar la extensión de muestras, aunque necesita del primero para poder generar la señal de fin de extensión, ya que durante el desarrollo se comprobó que intentar generar esta señal con el mismo contador resultaba demasiado complejo y no ahorraba una cantidad significativa de recursos.

Como última puntualización respecto al diseño de este módulo, queda por decir que se ha diseñado de forma parametrizable, lo que significa que, si bien está diseñado específicamente para trabajar con la wavelet rbio1.5 y con tamaños de entrada de 60 y 16 muestras, en realidad es capaz de trabajar con todo tipo de tamaños de entrada y de número de coeficientes del filtro. De esta forma, este módulo podrá trabajar sin problemas con otras wavelets en caso de necesitar trabajar con otra por cualquier motivo, cambiando simplemente los parámetros de tamaño en la instanciación del módulo.

6.2 Implementación del filtro

Uno de los principales requisitos para la implementación de la Transformada Wavelet Discreta (DWT) es la necesidad de usar un módulo de filtrado FIR debido a que la respuesta de la DWT es finita, por lo que implementar un IIR supondría un gasto de tiempo y recursos innecesario. Antes de poder proceder a implementar el filtro deberán valorarse, en primer lugar la forma en la que será implementado, y en segundo lugar qué resolución será necesaria para obtener a la salida unos resultados con un error lo menor posible respecto a una implementación ideal, es decir, en punto flotante. Esta resolución deberá escogerse de tal manera que se llegue a un compromiso entre el número de bits usados y la cantidad de recursos de la FPGA que se destinarán al diseño, teniendo en cuenta que el diseño objeto de este proyecto estará funcionando como un módulo en un proyecto de mayor envergadura y, por lo tanto, no se pueden desperdiciar recursos en lograr un error ínfimo si esto implica realizar cambios en la plataforma usada para poder emplazar el diseño en la FPGA. Además, dado que las especificaciones del proyecto donde se englobará el diseño aún pueden sufrir algunos cambios, se ha optado por realizar, en la medida de lo posible, un diseño paramétrico que permita ajustar el diseño a las especificaciones con cambios mínimos aunque respetando, eso sí, la decisión de implementar sólo el filtro de baja frecuencia, ya que de implementar también el filtro de alta frecuencia y aumentar la capacidad de las memorias, el diseño ocuparía unos recursos que, en realidad, no resultarían útiles para el funcionamiento del diseño. En este apartado acerca de la implementación se va a describir la forma de implementación, dejando la resolución en número de bits que tendrán las entradas, los elementos aritméticos (multiplicadores y sumadores) y la salida como parámetros modificables en la instanciación del módulo, los cuales serán ajustables desde el propio banco de pruebas de verificación del hardware. Esto permitirá llegar a un compromiso óptimo entre recursos utilizados y error cometido, lo cual repercutirá, además, en

la velocidad máxima que podrá lograr el módulo, siendo en general mayor cuanto menor sea el tamaño de los operandos.

Teniendo en cuenta lo anterior es necesario replantear la forma de funcionamiento del filtro, ya que en anteriores apartados se ha llevado a cabo una implementación en un lenguaje de programación de alto nivel, el cual se interpreta y ejecuta sobre un procesador. Por otra parte, en el caso del diseño hardware, se trata de diseñar el propio dispositivo que llevará a cabo el filtrado, lo que supone un cambio en el paradigma de diseño. Siguiendo con este planteamiento, es necesario recalcar que un filtro digital se compone básicamente de tres elementos: sumadores, multiplicadores y elementos de retardo, los cuales se combinan para realizar el filtrado. En el caso de la implementación en un lenguaje de programación la suma y la multiplicación se llevan a cabo con instrucciones del microprocesador, mientras que en la implementación hardware se lleva a cabo mediante dispositivos dedicados, aunque la forma de funcionamiento es similar en ambos casos. Es sin embargo en la implementación de los elementos de retardo donde existe la mayor diferencia, ya que en un lenguaje de programación, se introduce un vector de muestras y se realiza la convolución de todo el vector, mientras que en la implementación hardware las muestras entrarán una a una y la convolución se realizará en tiempo real. Este funcionamiento puede parecer a priori más complejo, aunque en la práctica significa que bastará implementar cada elemento de retardo como un flip-flop tipo D, lo que resulta más simple incluso que la implementación del prototipo software. Por otra parte, el hecho de que la implementación hardware pueda funcionar en tiempo real hace necesario tener en cuenta la velocidad a la que puede funcionar el filtro, lo que obliga a pensar si es posible segmentar el filtro para conseguir un throughput mayor y lograr frecuencias de funcionamiento más elevadas. De esta manera surge la posibilidad de realizar la implementación en forma directa, directa segmentada o en forma transpuesta [14], teniendo cada una de las implementaciones una serie de ventajas y desventajas sobre las que se hablará a continuación. Otras forma de implementación como la estructura Lattice no se han tenido en cuenta debido a que no resultan tan interesantes para la implementación sobre dispositivos de electrónica programable.

6.2.1 Implementación directa de un filtro FIR

La implementación directa de un filtro FIR es la más conocida y simple de estudiar, ya que se trata de la implementación del desarrollo de la expresión de la convolución directamente, de ahí su nombre. La expresión de un filtro FIR causal de orden N puede observarse en la ecuación (6.1), en este caso concreto, el orden N es igual a 10, dado que la wavelet empleada, rbio1.5, se implementa con filtros que poseen 10 coeficientes.

$$y[n]=\sum_{i=0}^{N-1} x[n-i]\cdot b_i = x[n]\cdot b_0 + x[n-1]\cdot b_1 + \ldots + x[n-(N-1)]\cdot b_{N-1} \qquad (6.1)$$

El resultado de implementar directamente en hardware la expresión (6.1) se puede observar en la Fig. 32, donde se observa la implementación en el caso N=5. Como se puede apreciar, la

implementación es muy sencilla aunque presenta un gran retardo debido al path crítico, el cual se ha resaltado en azul. En esta implementación el retardo total del path crítico viene expresado en la ecuación (6.2), siendo N el número de coeficientes, Tsum el retardo de un sumador (suponiendo que todos sean iguales) y Tmul el retardo de un multiplicador (en concreto multN-1 aunque se han supuesto que todos son iguales). Para el cálculo también se ha supuesto que el retardo de los flip-flops es despreciable, ya que el retardo introducido por la lógica combinacional es muy superior a éste.

$$T_{\text{path\_crítico}} = (N-1)\cdot T_{\text{sum}} + T_{\text{mul}} \tag{6.2}$$

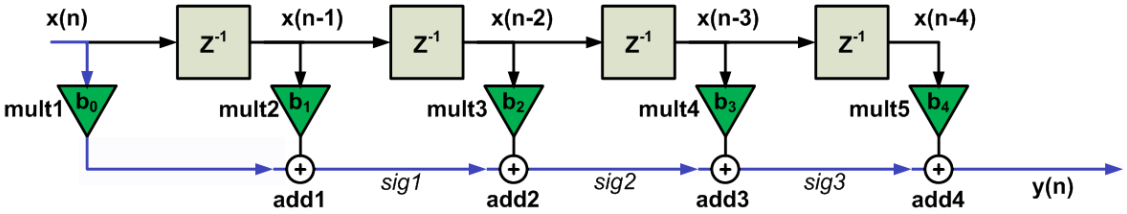

Fig. 32: Implementación directa de un filtro FIR [14]

Al excesivo retardo debido al path crítico es necesario añadir una segunda desventaja, la cual es de gran importancia en este diseño, la forma directa no está segmentada. Esta carencia de segmentación puede no resultar importante si tan sólo se desea filtrar una muestra o la velocidad de filtrado puede ser relativamente baja. Sin embargo, en este diseño, dado que debe operar en tiempo real y se precisa una frecuencia máxima de funcionamiento lo más elevada posible, la segmentación resultaría de gran ayuda. Es por ello, que puede plantearse esta misma implementación directa añadiendo una serie de registros, los cuales permitan segmentar el filtro, lo que permitirá, además, reducir el retardo del path crítico a cambio de usar algunos registros de desplazamiento adicionales.

6.2.2 Implementación directa segmentada de un filtro FIR

Conocida la implementación en forma directa, resulta sencillo introducir la implementación directa segmentada. Esta implementación segmentada consiste en dividir el flujo de datos del filtro en distintos segmentos, los cuales pueden trabajar con datos distintos en el mismo ciclo de reloj, lo que permite aumentar el rendimiento del filtro o throughput y a la vez la frecuencia máxima de funcionamiento, ya que al estar dividida la implementación en segmentos, el path crítico es menor. En este caso, la expresión (6.3) define el retardo del path crítico para esta implementación, el cual es prácticamente igual a la expresión (6.2) de la implementación directa no segmentada, con la salvedad de que en este caso la expresión depende de K, el número de etapas o registros que no forman parte de la segmentación, en cada segmento en los que se divida el filtro.

$$T_{\text{path\_crítico}} = K\cdot T_{\text{sum}} + T_{\text{mul}} \tag{6.3}$$

A modo de ejemplo, en la Fig. 33 se puede observar la implementación directa segmentada del filtro, donde un filtro de 5 coeficientes se ha dividido en dos segmentos, siendo el path crítico el mostrado en azul (asumiendo que todos los multiplicadores y sumadores son iguales). En este caso se puede observar que el path crítico es mucho menor y la estructura permanece igual a excepción de los registros añadidos para generar la segmentación. De esta forma esta implementación resulta mucho más interesante para generar el módulo hardware de filtrado que la opción sin segmentar.

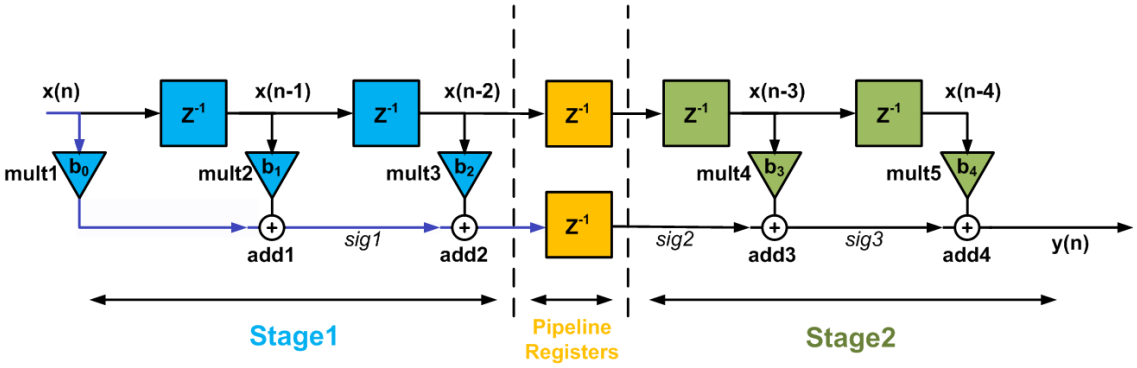

Fig. 33: Implementación directa segmentada de un filtro FIR [14]

Sin embargo, esta opción presenta una clara desventaja frente a la implementación directa no segmentada, el número de registros necesarios para la implementación de esta opción aumenta considerablemente. La cantidad de registros adicionales para lograr la segmentación se puede calcular con la expresión (6.4) donde S es el número de segmentos en los que se dividirá el filtro. Estos registros adicionales pueden no suponer un gran aumento de elementos respecto a los necesarios para implementar sumadores y multiplicadores, aunque se puede plantear la posibilidad de tratar de usar los propios registros de la implementación directa no segmentada para realizar la segmentación. Es con esta idea en mente con la que surge la opción de la implementación transpuesta, la cual permite reutilizar los registros que forman el filtro para llevar a cabo la segmentación, logrando así una reducción en la cantidad de elementos usados para implementar éste.

$$n^o{}_{FlipFlops} = 2 \cdot (S-1) \tag{6.4}$$

6.2.3 Implementación transpuesta de un filtro FIR

La implementación transpuesta es una forma distinta de enfocar el problema, la cual se basa en el teorema de la transposición o “teorema de flujo de datos inverso”, el cual demuestra que dos implementaciones de un flujo de datos son equivalentes y realizan la misma función si se cumplen ciertas condiciones. Estas condiciones son:

- Intercambiar la entrada y la salida.
- Invertir la dirección del flujo de datos en todas las ramas sin cambiar su función.

Si al aplicar estas reglas a una implementación, se llega a la otra implementación a examinar, se puede concluir que ambas son equivalentes y realizarán la misma función, aunque pueden tener propiedades distintas en lo relativo a su comportamiento. Tras aplicar el teorema de transposición a la forma directa no segmentada se llega a la forma transpuesta, la cual se muestra en la Fig. 34. Para demostrar la validez de este teorema se puede realizar una sencilla comprobación a partir de la implementación transpuesta, si esta implementación posee la misma expresión matemática que la implementación directa, ambas serán equivalentes y podrán usarse indistintamente.

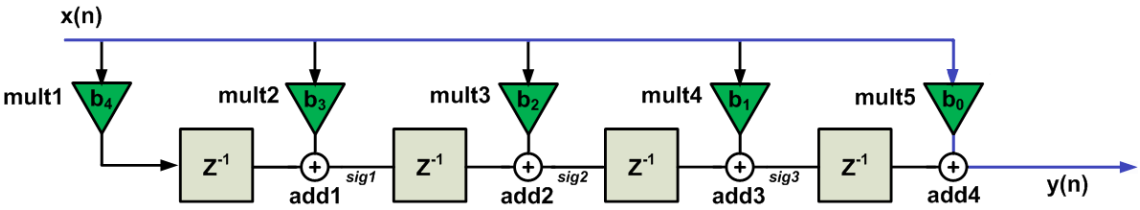

Fig. 34: Implementación transpuesta de un filtro FIR [14]

A partir de la Fig. 34 puede deducirse la expresión (6.5), la cual es, desarrollada, la misma expresión que la reflejada en la ecuación (6.1) particularizada para N=5. De esta manera queda demostrado que la implementación transpuesta es equivalente a la implementación directa, por lo que pueden emplearse ambas indistintamente para la implementación del filtro.

$$y[n]=x[n](((((b_4*\delta[n-1]+b_3)*\delta[n-1]+b_2)*\delta[n-1]+b_1)*\delta[n-1]+b_0)$$

$$y[n]=x[n-4]\cdot b_4+x[n-3]\cdot b_3+x[n-2]\cdot b_2+x[n-1]b_1+x[n]b_0=\sum_{i=0}^{4} x[n-i]\cdot b_i \quad (6.5)$$

No obstante, aunque ambas implementaciones son equivalentes poseen propiedades distintas debido a la distribución de sus componentes. En este caso los registros de desplazamiento realizan tanto la función de filtrado como la función de segmentación, con lo que se consigue el mismo uso de recursos que en una implementación directa, pero añadiendo segmentación en cada etapa. De esta forma se consigue aumentar el throughput en gran medida a la vez que se reduce el retardo del path crítico, el cual puede calcularse mediante la expresión (6.6), y, por tanto, se aumenta la frecuencia máxima de funcionamiento. Esta implementación sin embargo tiene un problema debido a su propio diseño, la entrada requiere un fan-in elevado, ya que esta señal debe llegar a todos los multiplicadores del filtro. En diseño VLSI este problema podría resolverse mediante buffers debidamente dimensionados, mientras que en electrónica programable el problema se resolvería por el propio sintetizador, probablemente añadiendo buffers entre la entrada y los multiplicadores. Esta solución, si bien no afecta al diseño, ya que el diseñador se limita a definir el comportamiento que debe tener el diseño (diseño comportamental o behavioural), puede afectar al número de elementos usados y disminuir ligeramente la frecuencia máxima de funcionamiento. A pesar de este efecto, el cual apenas tiene influencia en el diseño del filtro, esta es la implementación más usada cuando se desean sintetizar filtros en dispositivos de electrónica programable como FPGA's debido a que ocupan menos recursos y presentan un throughput y frecuencia máxima elevadas.

$$T_{\text{path\_crítico}} = T_{\text{sum}} + T_{\text{mul}} \tag{6.6}$$

6.2.4 Comparativa entre implementaciones

Finalmente, tras haber realizado un barrido sobre las posibles implementaciones del filtro, el cual será el elemento central del módulo que implemente la DWT2, se van a comparar las características para elegir una de las implementaciones propuestas. Para poder comparar las características de cada forma de implementación, se ha creado la Tabla 6.1, donde se muestran las características más relevantes para realizar la decisión. En el caso de la forma directa segmentada, se ha elegido la segmentación más favorable a nivel de throughput y retardo del path crítico, lo que implica que el filtro se ha dividido en 8 segmentos, los cuales han añadido, de acuerdo con la expresión (6.4), 16 registros adicionales a los ya empleados por el filtro.

Implementación	Directa	Directa segmentada	Transpuesta
Multiplicadores empleados	10	10	10
Sumadores empleados	9	9	9
Registros empleados	9	25	9
Fan-in (elementos a la entrada)	1	1	10
Retardo máximo	$T_{\text{sum}} + T_{\text{mul}}$	$T_{\text{sum}} + T_{\text{mul}}$	$9 \cdot T_{\text{sum}} + T_{\text{mul}}$
Throughput (muestras/ciclo)	1/9	1 (tras inicialización)	1 (tras inicialización)

Tabla 6.1: Comparativa entre las distintas implementaciones del filtro FIR

A partir de los datos recogidos en la Tabla 6.1, se puede determinar que, dado que el fan-in no es determinante al encargarse el propio sintetizador de resolver este problema, la mejor opción es la implementación de la forma transpuesta. Esta opción demuestra conseguir el mayor throughput y menor retardo empleando la menor cantidad de registros, lo que explica también el porqué esta es la opción preferida para la implementación de filtros en electrónica progrmable. De esta forma, la implementación que finalmente se realizará para el diseño tendrá un esquema similar al mostrado en la Fig. 34, con la diferencia de que en el caso del diseño, el filtro contará con 10 coeficientes, aunque, de la misma manera que en el caso del extensor de muestras, será diseñado de manera paramétrica para poder cambiar tanto el número de coeficientes del filtro, como el valor de éstos, además del tamaño de los operandos de entrada, salida y los coeficientes. Gracias a esto, el filtro podrá ser reutilizado en otros diseños sin necesidad de modificaciones, salvo para cambiar el tipo de datos (con signo o sin signo) con los que trabaja el filtro, y podrá modificarse la wavelet en caso de ser necesario.

6.3 Memoria de intercambio

La memoria de intercambio es uno de los elementos claves en el desarrollo del módulo DWT2, ya que es la encargada de hacer de buffer entre el barrido en el eje X y los barridos en el eje Y. El funcionamiento básico de la memoria es bastante simple, en primer lugar, cuando los datos a la salida del banco de filtros están listos para ser escritos, la memoria se encarga de guardar los datos, los cuales serán leídos posteriormente en un orden diferente, permitiendo así realizar el filtrado en el eje contrario. Sin embargo, la memoria debe realizar algunas funciones adicionales para poder funcionar como buffer, por ejemplo, antes de guardar los datos en la memoria debe realizar un diezmado para evitar guardar datos redundantes, y también debe transponer los datos almacenados en la memoria de escritura para que durante las dos fases de lectura los datos se encuentren en el orden correcto. Estas tareas, mostradas en el esquema de funcionamiento de la Fig. 35, añaden complejidad al módulo de intercambio dado que pasa de ser una simple memoria a ser un módulo compuesto por las memorias de lectura y escritura y los multiplexores de transposición formando el data-path, y una máquina de estados de tipo Mealy con salidas registradas para evitar posibles glitches al atacar la lógica combinacional, además de varios contadores formando el control-path.

Fig. 35: Esquema de funcionamiento de la memoria de intercambio

La memoria de intercambio es, por lo tanto, una serie de elementos que actúan de manera muy similar a una memoria, pero que resulta más compleja como demuestra el diagrama de bloques mostrado en la Fig. 36. El funcionamiento de este módulo permite trabajar con 16 líneas de datos a la entrada en el ciclo de escritura y 17 a la salida en cada ciclo de lectura, lo que permite lograr velocidades de funcionamiento muy elevadas, a cambio de una gestión de memoria más compleja y un mayor consumo de recursos. Esto es así debido a las propias exigencias del escáner, el cual necesita un módulo de filtrado lo más rápido posible, por lo que soluciones más sencillas para el diseño de este módulo no se han podido plantear.

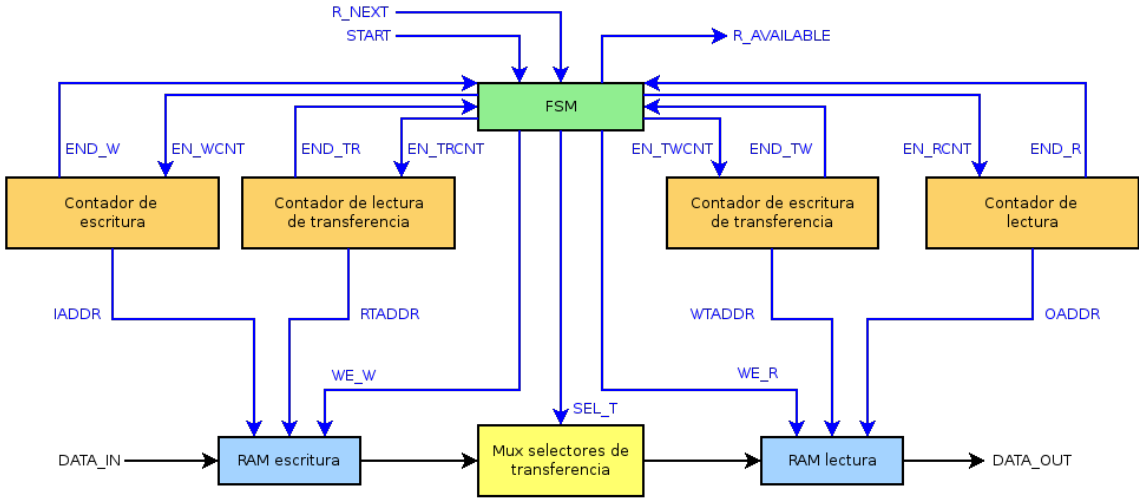

Fig. 36: Diagrama de bloques de la memoria de transferencia

6.4 Implementación DWT2

Tras diseñar los módulos de extensión de datos, filtrado y memoria de intercambio, se puede comenzar el diseño del módulo DWT2, el cual implementará todos los módulos necesarios para realizar la Transformada Wavelet discreta de dos dimensiones a excepción de la gestión de memoria inicial y final, las cuales están aún por diseñar y no forman parte de este proyecto. Es por este motivo que se ha diseñado el módulo de tal manera que sea capaz de operar cargando un dato de cada fila de manera simultánea, pero también sea posible usar buffers a la entrada y a la salida para cargar los datos desde un bus o una FIFO y volcar los datos a otro bus o FIFO. Gracias a esto es posible lograr sacar el máximo partido al diseño y lograr velocidades de filtrado superiores a las que podrían lograrse de emplear un único módulo de filtrado en lugar de un banco de módulos de filtrado.

El diseño de este módulo ha resultado el más sencillo de todos los diseñados, excluyendo los contadores y el multiplexor, los cuales resultaron tan simples de diseñar que no se ha considerado interesante realizar un análisis en profundidad. Esto es debido a que los módulos extensor de muestras, filtro y memoria de intercambio se han diseñado explícitamente para funcionar en un mismo diseño, con lo que en el módulo DWT2 la dificultad se reduce a conectar correctamente dichos módulos, tal y como se puede observar en la Fig. 37, y crear una máquina de estados finitos sencilla cuya única función sea coordinar los módulos anteriores. Esta máquina de estados finitos es de tipo Moore debido a que resultaba la implementación más idónea y, dado que todas las salidas de la máquina atacan a registros, no ha sido necesario registrar las salidas para evitar posibles glitches en el diseño final.

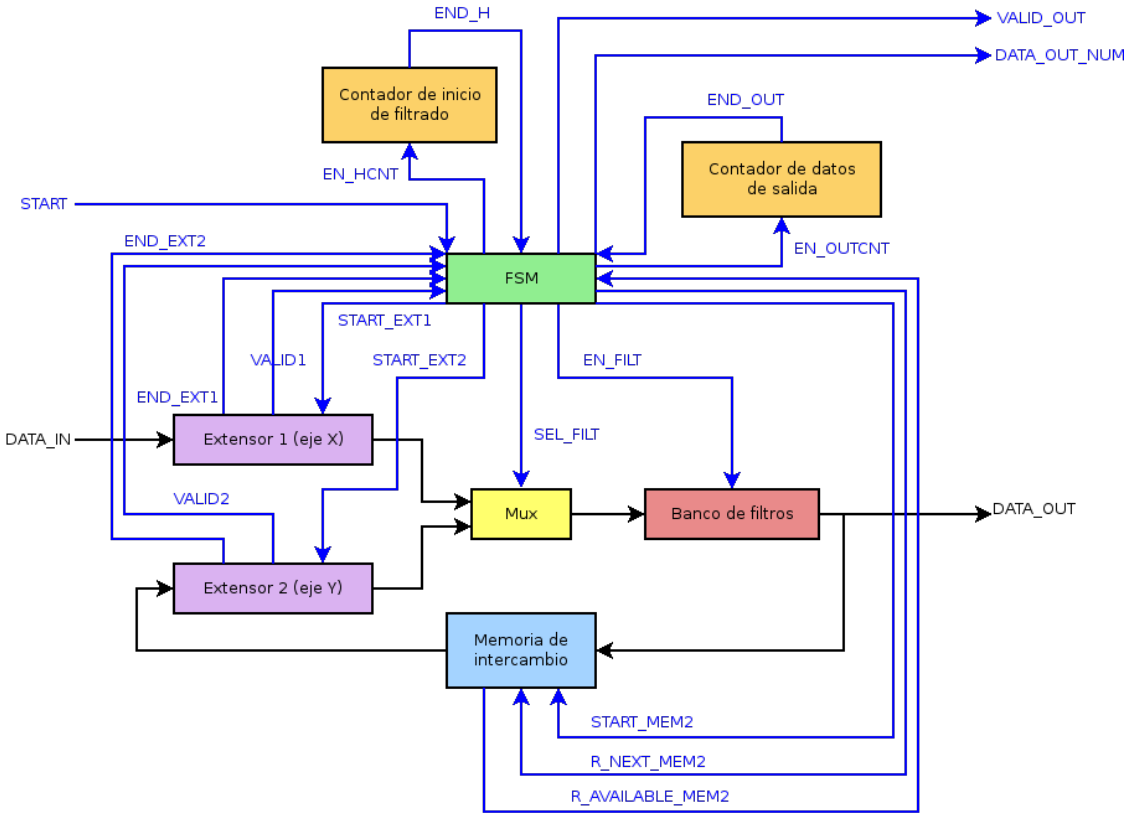

Fig. 37: Diagrama de bloques del módulo DWT2

Por último en relación al diseño del módulo es necesario destacar que, para lograr que el módulo pueda funcionar, debe colocarse a la salida un diezmador de las mismas características que el empleado en la memoria de intercambio. Este diezmador no se ha incluido en el propio diseño dado que puede emplearse como tal una memoria con un FlipFlop tipo D conectado a la señal de habilitación de escritura, lo que ahorraría recursos de la FPGA. Por desgracia, al no estar totalmente definida la gestión de memoria de entrada y salida del módulo se ha preferido dejarlo tal cual, indicando una posible implementación para evitar realizar un diseño que posteriormente sea necesario modificar por usar una gestión de memoria distinta.

6.5 Síntesis del diseño

Terminado el diseño de todos los módulos se ha procedido a realizar la síntesis del módulo DWT2, el cual se sitúa como tope de jerarquía y contiene a todos los demás. Durante la síntesis, al igual que durante la realización de las pruebas de funcionamiento, como más adelante se comentará, ha sido necesario realizar ciertas modificaciones al diseño para que éste funcione correctamente y sea un diseño sintetizable que no tenga elementos indeseados como latches. Estos últimos pueden interferir con el correcto funcionamiento del diseño e incluso crear osciladores no deseados si en algún punto del diseño existe realimentación. Por este motivo, los datos que se expondrán a continuación son los correspondientes al diseño funcional tras realizar la verificación de funcionamiento en el banco de pruebas, siendo por tanto datos relativos al diseño real que es posible implementar en la FPGA, definiendo eso sí las entradas y salidas correspondientes.

En la Tabla 6.2 se muestran la cantidad de elementos totales de la FPGA, los elementos utilizados para implementar el diseño y el porcentaje que resultan estos últimos sobre el total. De esta manera se puede comprobar que el uso de CLBs y LUTs es muy bajo, siendo de poco más de un 1% de los elementos totales, a pesar de que el diseño trabaja con imágenes, lo que requiere una gran cantidad de recursos para tratar y almacenar los datos. Por otra parte también se observa que la cantidad de elementos DSP es relativamente elevada, un 8,85% del total, lo que se explica por la gran cantidad de este tipo de recursos necesaria para implementar el filtro. A pesar de esto, puede lograrse un menor uso de este tipo de recursos eliminando los multiplicadores cuyos coeficientes son ceros, algo que ciertos sintetizadores pueden realizar de manera automática sin alterar el diseño, no teniendo esta capacidad el sintetizador de Vivado. Por este motivo, de querer realizar esta optimización con el sintetizador empleado, debería cambiarse el diseño del filtro FIR, el cual perdería la capacidad de ser paramétrico y poder emplearse en otros diseños, algo que impediría reutilizar este filtro en caso de realizar modificaciones, motivo por el que no se ha llevado a cabo esta optimización.

Además de los elementos DSP, el diseño hace uso de una cierta cantidad de memoria RAM dedicada de la FPGA, en concreto un 5,5% de los elementos totales disponibles, algo que si bien no es excesivo, si bien resulta más elevado que los elementos CLB. Sin embargo, de la misma manera que en el caso de los DSP, podría incluirse una optimización en las memorias del extensor de muestras del eje X para poder utilizar esta memoria tanto por el módulo DWT2

como por el módulo de gestión de memoria, el cuál está aún por diseñar. Debido a esto último, no se ha podido implementar ninguna optimización, aunque se deja abierta la posibilidad de realizarla cuanto el sistema de gestión de memoria esté terminado sin necesidad de modificar por completo el módulo DWT2.

Tipos	Elementos totales de la FPGA	DWT2	Porcentaje de uso
CLB LUTs	242400	1054	0,43%
Block RAM	600	33	5,5%
DSP	1920	170	8,85%
F7 muxes	121200	340	0,28%
F8 muxes	60600	170	0,28%
CLB	30300	339	1,11%
LUT as logic	242400	1054	0,43%

Tabla 6.2: Elementos usados por el diseño

Otras optimizaciones que sí se ha realizado para reducir el uso de recursos de la FPGA manteniendo la misma funcionalidad se centran en el banco de filtros. Por una parte, como se demostró durante la investigación sobre la wavelet óptima, la codificación y los umbrales, los umbrales no se han implementado ya que generaban errores muy significativos y consumirían una cierta cantidad de recursos sin aportar ventaja alguna. También se han suprimido los filtros de alta frecuencia, dejando únicamente el banco de filtros de alta frecuencia, lo que no sólo ha permitido recortar a la mitad los recursos necesarios para la implementación del filtro, sino que también ha recortado a la mitad la cantidad de memoria necesaria y ha facilitado la implementación del diseño. Siguiendo con la implementación del filtro, aunque en este caso con una repercusión bastante menor en los recursos utilizados, se ha decidido usar lógica sin signo debido a que, tanto los datos de entrada como los coeficientes del filtro son siempre positivos, lo que permite ahorrar el bit de signo sin afectar al resultado. En caso de que en un futuro se desee utilizar coeficientes o datos negativos bastaría con añadir un bit más a los datos de entrada, coeficientes y módulos aritméticos (multiplicadores y sumadores), además de explicitar el uso de lógica con signo añadiendo “signed” a las definiciones de los puertos de entrada y salida y los registros empleados en el filtro.

También es necesario comentar una de las opciones utilizadas para ayudar a sintetizar la memoria usada en el diseño, dado que por defecto Vivado tratará de sintetizar memoria distribuida o memoria dedicada en función de si la síntesis logra mejores resultados haciendo uso de un tipo u otro. El sintetizador de Vivado, en diseños relativamente pequeños suele preferir la implementación de memoria distribuida, lo que logra velocidades ligeramente superiores a cambio de un mayor uso de recursos CLB en lugar de bloques de memoria RAM

(BRAM). Para obligar al sintetizador a utilizar un tipo de memoria en concreto, existe una directiva que fuerza al sintetizador al uso de dicho tipo de memoria, siempre que sea posible, ya que, por ejemplo, en caso de querer sintetizar memorias RAM asíncronas no será posible el uso de memoria dedicada, ya que ésta es, por diseño, síncrona. En el caso de este diseño la memoria es síncrona, en concreto la memoria usada es “Simple Dual Port”, la cual puede sintetizarse sin ningún problema como memoria dedicada, aunque se ha debido hacer uso de la directiva mencionada añadiendo “(* ram_style = "block" *)” antes de definir la memoria.

Por último quedan por añadir los datos de velocidad de funcionamiento del sistema, la cual se ha intentado que sea lo mayor posible para cumplir con los requisitos de funcionamiento del resto del sistema que integra el escáner y tener cierto margen por si la frecuencia máxima fuese menor como resultado del rutado del sistema completo. En concreto, la frecuencia máxima de funcionamiento llega hasta 172 MHz aproximadamente, la cual se ha logrado añadiendo una limitación o “constratint” de 5,8 ns durante la síntesis del diseño, la cual, según el propio sintetizador, se ha podido satisfacer sin problemas, logrando los datos de implementación expuestos en la Tabla 6.2. En cuanto al throughput, la obtención de este parámetro no la realiza el sintetizador, sino que debe ser el diseñador el que, conociendo el diseño obtenga el dato, o usar un banco de pruebas que obtenga este parámetro automáticamente como en este caso. El dato obtenido a partir del testbench, contando todos los ciclos de reloj desde que se activa la señal “START” que da inicio a la carga de datos, hasta que todos los datos han salido del módulo, es decir, cuando se desactiva la señal “VALID_DATA_OUT” que a nivel alto marca la validez de los datos de salida. Este dato se sitúa en 185 ciclos de reloj para realizar el filtrado de una imagen completa, por realizar una comparativa, si se hubiese utilizado un único filtro para realizar el filtrado de la imagen la cantidad de ciclos necesaria para filtrar una imagen correspondería a filtrar cada fila por separado una vez extendidos los coeficientes y después realizar lo propio con cada columna, algo que puede calcularse con la expresión (6.7), deducida a partir de la expresión (4.1) aplicada a dos dimensiones. Para las imágenes empleadas por el escáner, de lX=60 píxeles de ancho en el eje X y lY=16 píxeles de alto en el eje Y, y un filtro de lH=10 coeficientes da como resultado 2404 ciclos, haciendo necesario casi 13 veces más ciclos de reloj para filtrar cada imagen. Debido al bajo throughput del diseño el filtro no producirá cuellos de botella en el procesado de datos y permitirá usar técnicas para paralelizar el procesamiento de imágenes. Con esto se concluye que, asociado a un buen sistema de gestión de memoria, el diseño presentado resulta más interesante que usar un único filtro a pesar de la gestión de memoria más compleja y el mayor empleo de recursos de la FPGA, dado que esta otra posibilidad implicaría la aparición de un cuello de botella en el filtrado y obligaría a detener el flujo de datos o complicar el diseño para solventar este inconveniente.

$$nCLK = [lX + 2(lH - 1)] \cdot lY + [lY + 2(lH - 1)] \cdot (lX + lH - 2)/2 \tag{6.7}$$

Capítulo 7. Verificación hardware

El diseño del hardware que implementa la DWT2 es una parte importante del presente trabajo, pero esta implementación no tendrá realmente ninguna utilidad si no se verifica que su funcionamiento es el adecuado. Con este objetivo en mente se deberá realizar una verificación del diseño que permita comprobar, por una parte que el diseño es funcional, y por otra parte qué tamaño deben tener la entrada, los coeficientes del filtro y los elementos aritméticos (multiplicadores y sumadores) para lograr unos buenos resultados. Para poder alcanzar estos objetivos será necesario, en primer lugar, construir un banco de pruebas o "testbench" en el lenguaje SystemVerilog, el cual llevará a cabo la simulación del hardware y una primera comprobación de las señales generadas por el diseño. Este banco de pruebas por sí sólo podría validar el diseño a nivel eléctrico, pero para evaluar el tamaño de los elementos que componen el diseño y comprobar la forma de la distribución de los errores al regenerar la señal deberá emplearse otro banco de pruebas. Dicho banco deberá tomar los datos obtenidos en la simulación anterior, aplicarles la DWT2 inversa y obtener las estadísticas de errores de manera similar a como se ha realizado durante la investigación de la wavelet óptima. Teniendo en cuenta este segundo banco de pruebas surge una posibilidad para facilitar la comprobación del diseño, fusionar ambos bancos de pruebas en uno sólo, de tal manera que se pueda realizar la verificación completa del diseño con la ejecución de un sólo programa. Esta idea tan interesante se ha desarrollado hasta lograr dicha fusión, la cual ha cristalizado en un banco de pruebas de alto nivel programado en Python y haciendo uso del entorno Jupyter-Notebooks de manera similar a los bancos de pruebas del prototipo software, el cual se encarga de lanzar la simulación en SystemVerilog haciendo uso del software ModelSim, recoger los datos y generar las estadísticas de errores y ratio de compresión.

7.1 Prototipo hardware de comprobación

A pesar de que en este documento el diseño se encuentre en un capítulo anterior, lo cierto es que cronológicamente tanto el diseño de la implementación como el diseño del banco de pruebas se han llevado a cabo simultáneamente para ayudar a resolver los errores de diseño de manera más rápida y evitar extender el tiempo de desarrollo en exceso. Por este motivo, dado que el diseño no está terminado y el banco de pruebas no tiene un diseño con el que comprobar su propio funcionamiento, se ha desarrollado un prototipo de comprobación, el cual está basado en el diseño hardware y diseñado igualmente en SystemVerilog, pero a diferencia de éste, el prototipo

no es sintetizable y por lo tanto no constituye un diseño válido. La misión de este prototipo, por tanto, es emular el comportamiento del diseño para que una vez se termine éste se pueda verificar de manera rápida y sencilla, teniendo mayor libertad que en el diseño del hardware, dado que el prototipo no necesita ser sintetizable. Sin embargo, debido a que ciertos módulos son más fáciles de implementar, en concreto el extensor de muestras y el filtro FIR, se ha decidido usar estos módulos dentro del prototipo, con lo que se comprobará su funcionamiento para que una vez el diseño completo esté en el banco de pruebas queden menos errores por depurar.

Otra de las características del prototipo tiene que ver con la forma generalista en la que se desarrollará el banco de pruebas, permitiendo verificar el diseño desarrollado en el capítulo anterior, pero también otros diseños similares. Esta característica precisa que el banco de pruebas sea capaz de trabajar con los cuatro tipos de coeficientes resultantes de aplicar la DWT2, baja frecuencia (cA), detalles en el eje horizontal (cH), detalles en el eje vertical (cV) y detalles en diagonal (cV). Debido a esto, la optimización empleada en el diseño para eliminar los filtros de alta frecuencia no es aplicable en el prototipo, como tampoco lo es la eliminación de los umbrales. Gracias a esto, en caso de que el diseño del hardware no se ajuste a la gestión de memoria deseada y necesite un rediseño completo, el banco de pruebas podrá utilizarse cambiando el modelo a verificar y realizando las modificaciones que proceda en el banco de pruebas en SystemVerilog para que este nuevo diseño funcione correctamente.

El prototipo contiene, por lo tanto, una mayor cantidad de elementos que el diseño hardware desarrollado en el capítulo anterior, por lo que el diagrama de bloques mostrado en la Fig. 37 ya no es aplicable. En su lugar se muestra en la Fig. 38 el diagrama del bloques del prototipo, donde se puede apreciar que se han añadido los módulos de codificación en número de bits, que se queda únicamente con los bits más significativos seleccionados de cada grupo de coeficientes (cA, cH, cV y cD), y de umbrales, que eliminan cualquier dato cuyo valor absoluto no esté por encima del umbral establecido para cada grupo de coeficientes. A primera vista puede parecer que diagrama del prototipo es igual al diagrama del diseño, pero en realidad sólo comparte los módulos de extensión de datos, filtrado y los contadores, siendo diferentes la máquina de estados y la memoria de intercambio, las cuales no son sintetizables y no podrían llegar a implementarse en un diseño real pero son funcionales a efectos de simulación, por lo que son útiles para realizar las pruebas de funcionamiento del banco.

Por último en relación al prototipo, debe destacarse que, debido a su configuración, es posible configurar éste para realizar pruebas sobre la DWT2 directamente en hardware. Esto abre la puerta a poder realizar nuevos experimentos sobre la wavelet a emplear, umbrales y codificación directamente sobre hardware, ya que este prototipo funcionará de manera muy similar al diseño. Dicha característica, si bien no podrá ser aprovechada por el diseño actual, al encontrarse en una fase muy avanzada, podría usarse en caso de querer realizar otro diseño distinto que implementase la DWT2. Sin embargo, también existen algunas sombras en relación a la simulación de este prototipo, ya que la simulación en ModelSim es notablemente más lenta

que la simulación del prototipo software sobre Python, lo que puede llevar a simulaciones de varios minutos para decenas de muestas a varios días para decenas de miles. Por ello se recomienda que, en caso de querer aprovechar estas ventajas se realice primero una simulación con el prototipo software en punto fijo y posteriormente se elija un subconjunto de esas muestras para evaluar el comportamiento del hardware.

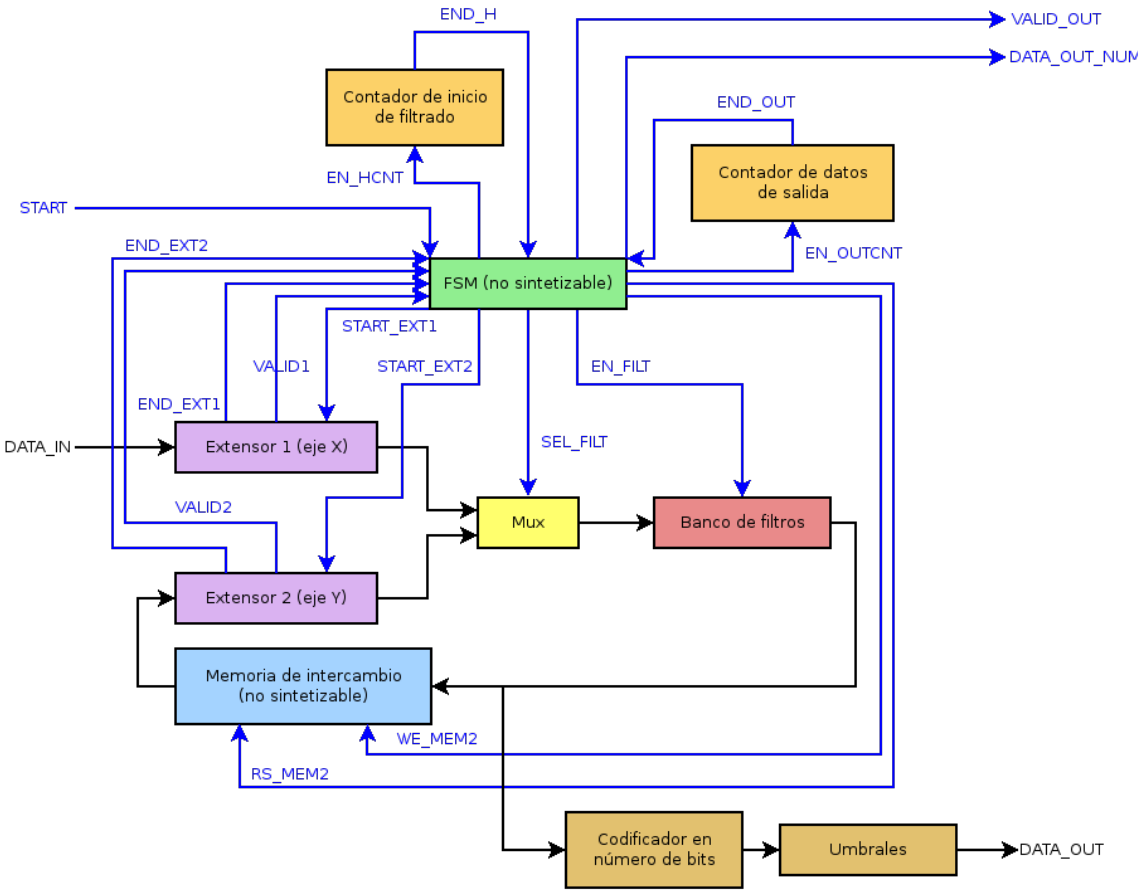

Fig. 38: Diagrama de bloques del prototipo hardware

7.2 Diseño del banco de pruebas

El diseño del banco de pruebas para llevar a cabo la verificación y caracterización, como se ha comentado anteriormente, en realidad es la fusión de un banco de pruebas desarrollado en SystemVerilog, el cual se encarga de generar los estímulos, introducir los datos en en el diseño hardware y obtener los datos de la simulación, y de un banco de pruebas desarrollado en Python, el cual se encarga de generar los ficheros de datos para la simulación, cargar los datos tras ésta y realizar las estadísticas de errores al regenerar los datos.

Para poder explicar el funcionamiento más en profundidad, primero se va a describir el funcionamiento del banco de pruebas en SystemVerilog, cuyo diagrama de bloques se puede observar en la Fig. 39, el cual se lanza desde el banco de verificación diseñado en Python sin necesidad de abrir la interfaz gráfica de ModelSim. Durante la inicialización el banco carga en

la memoria de datos las imágenes de un fichero de inicialización de memoria donde previamente se han almacenado de forma secuencial las imágenes que se introducirán al diseño. Tras esto, el banco comenzará a funcionar poniendo en marcha el contador de throughput y cargando una imagen en el registro de entrada, el cual introducirá ésta en el Dispositivo Bajo Verificación (DUV), el cual se encargará de procesar la imagen y obtener el resultado final. Una vez estén listos los datos del DUV, éste activará la señal "VALID_OUT", la cual servirá para habilitar la escritura en el registro de salida, mientras que la señal NUM_OUT indicará en qué zona del registro deberá escribirse, ya que no se procesa toda la imagen a la vez, sino que se procesa por partes. Además, es necesario indicar que este registro de salida también realiza la función de diezmado, el cual es necesario para realizar la DWT2 y evitar datos redundantes. Una vez la imagen completa se ha escrito en el registro de salida, el contador de datos de salida notifica al controlador para que guarde estos resultados en la memoria de guardado, prepare la nueva imagen y detenga el contador de throughput. Este ciclo se repetirá hasta que se hayan procesado todas las imágenes, momento en el que el controlador activará la señal "END", la cual guardará los datos de la memoria de guardado en un archivo de memoria de la misma naturaleza que el archivo de datos inicial. De esta forma los datos procesados están listos para poder leerse desde el banco de pruebas desarrollado en Python, realizando el filtrado de varias imágenes con una única ejecución del banco de verificación, lo que permite ahorrar tiempo, pero también comprobar que el módulo es capaz de trabajar en modo continuo y no es necesario realizar un reset asíncrono para que funcione correctamente tras cada imagen.

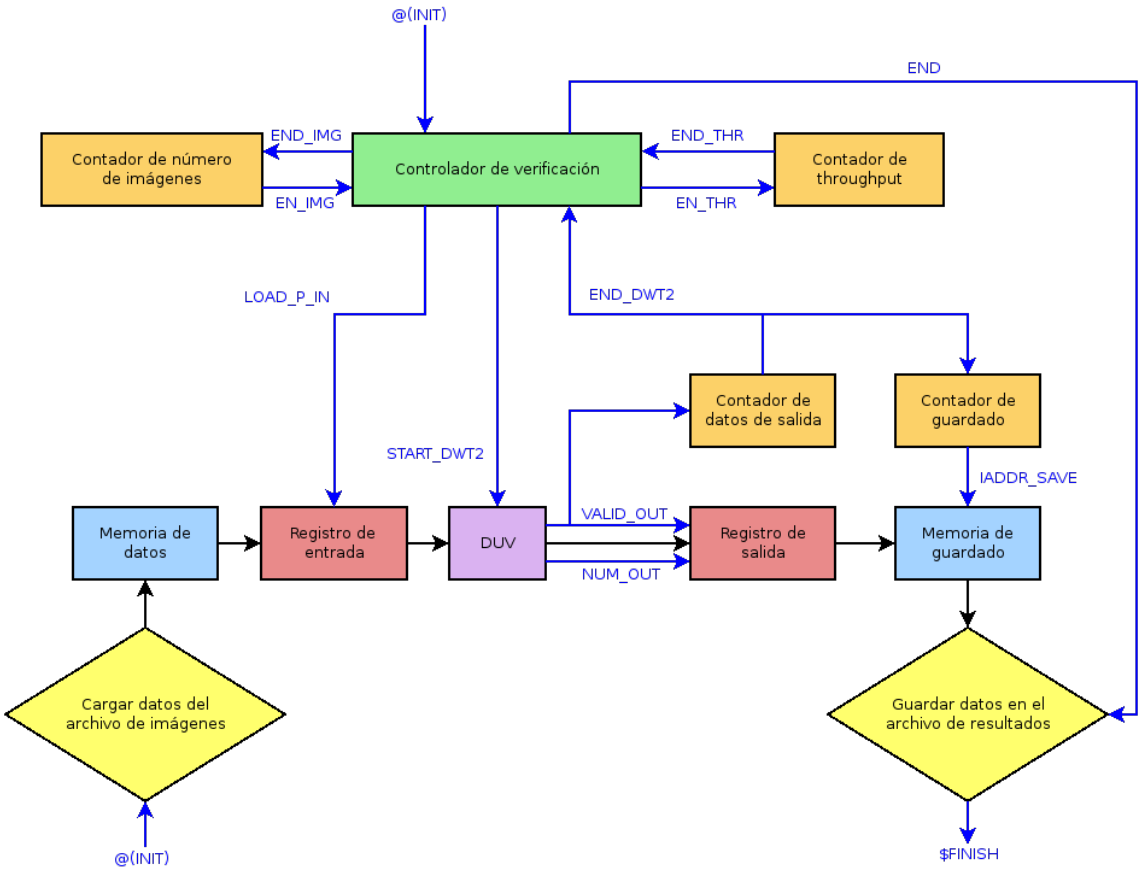

Fig. 39: Diagrama de bloques del banco de verificación

Este banco de simulaciones desarrollado en SystemVerilog se engloba dentro del banco de verificación desarrollado en Python, el cual se encarga de realizar las funciones auxiliares para llevar a cabo la simulación y generar las estadísticas de errores, tareas mostradas en la Fig. 40. De todas éstas, la primera que debe llevarse a cabo es generar los datos de la simulación a partir de las imágenes de los eventos detectados por el escáner. Estas imágenes están contenidas en un archivo tipo HDF, el cual permite guardar de forma ordenada grandes cantidades de información, como las tablas de datos donde se almacenan las imágenes de los eventos. Por desgracia los simuladores de hardware como ModelSim no tienen capacidad para trabajar con este tipo de archivos, por lo que será necesario cargar los archivos de imágenes en Python, convertir los datos a números expresados en base 2 y guardarlos como un archivo de inicialización de memoria, con el cual sí puede trabajar el simulador. Los archivos de coeficientes de los filtros, los cuales se cargan a partir de la librería “pywt” donde se encuentran definidos, se almacenarán de la misma forma que los datos en dos archivos separados, separando los coeficientes de alta y baja frecuencia. Hay que destacar además, que es durante la generación de estos archivos cuando se decide la cantidad de bits que se asignarán a las entradas y a la salida, configurando así el diseño, además de qué parte de los bits estará dedicada a decimales y qué parte a enteros. La cantidad de bits dedicada a decimales y enteros es información que se utiliza para generar los archivos de datos y coeficientes y obtener los datos a partir del archivo de resultados de la simulación, pero el hardware es independiente del lugar donde se sitúe la coma y siempre actuará de la misma forma. Estos datos serán, por tanto, útiles únicamente para interpretar los datos de la simulación y poder generar los datos estadísticos del error generado por el módulo, pero no tendrán ninguna repercusión en la propia simulación.

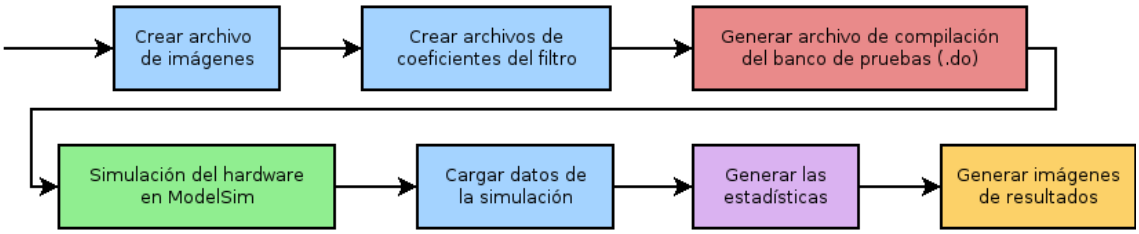

Fig. 40: Esquema de funcionamiento del banco de pruebas

Siguiendo el esquema de funcionamiento de la Fig. 40, el siguiente paso es generar el archivo de configuración de la simulación, de extensión “.do”, el cual se interpretará por ModelSim y contiene la configuración paramétrica de la simulación. Esta configuración se lleva a cabo mediante definiciones de parámetros en la llamada al comando “vlog”, encargada de compilar los módulos que componen el diseño, los cuales pueden contener valores que se pueden utilizar dentro del banco de simulación de SystemVerilog. Algunos de estos parámetros configurables son el tamaño de las entradas y la salida, el número de imágenes a usar, el tamaño de éstas, los nombres de los archivos de entrada y salida, etc. Tras realizar la configuración de la simulación, ésta se puede lanzar, ya configurada, y generar el archivo de datos de simulación o de resultados, el cual deberá leerse en Python al terminar la simulación e interpretarse correctamente. Esta interpretación consiste básicamente en leer los datos obviando los comentarios introducidos por la simulación, traducir estos datos a números en base decimal y ordenarlos correctamente, separando las distintas imágenes y ordenando los datos de éstas en filas y columnas.

Tras haber obtenido los datos de la simulación puede procederse a la generación de las estadísticas de funcionamiento, las cuales se dividen en dos métodos bien diferenciados, el primero y más sencillo, cálculo del error cuadrático cometido en la reconstrucción de cada imagen, y el segundo, más complejo pero más útil, la obtención de las estadísticas de errores, las cuales poseen la misma naturaleza que las obtenidas durante la creación del prototipo software. El error cuadrático se calcula realizando la raíz cuadrada de la suma cuadrática de los errores o diferencias entre la imagen original y la reconstruida obtenidos al reconstruir la imagen a partir de los datos obtenidos por la simulación como marca la expresión (7.1). Este error, si bien puede informar sobre las diferencias entre ambas imágenes, no aporta información realmente útil sobre la simulación, por lo que para obtener dicha información deberán obtenerse las estadísticas de errores, de manera similar a las que aparecen, por ejemplo en la Fig. 30, pero aplicados a la simulación del diseño hardware. Para lograr estas estadísticas se van a emplear las funciones usadas en la librería "Simlib" empleadas en capítulos anteriores cambiando únicamente la forma de obtener los datos reconstruidos para que sean los obtenidos en la simulación hardware y la codificación empleada para enviar los datos con la que se calcula el ratio de compresión, tema sobre el que se hablará durante la verificación del diseño.

$$e=\sqrt{\sum_{k=0}^{60}\sum_{m=0}^{16}\left(píxel_{original}[k][m]-píxel_{reconstruido}[k][m]\right)^2} \tag{7.1}$$

Tras obtener las estadísticas de errores, las cuales se han realizado con un subconjunto de todas las imágenes disponibles para evitar que la simulación pueda tardar hasta varios días en completarse, se va a generar un archivo PDF con estos datos para poder guardar los resultados y compararlos sin necesidad de realizar capturas de pantalla. Este documento contendrá la información más relevante respecto al funcionamiento del diseño, aunque es algo abstracta y no da detalles visuales sobre el efecto que tiene la compresión sobre éstas. Para poder visualizar el efecto del diseño sobre las imágenes de los eventos se ha añadido la opción de generar un documento donde se muestre la imagen original, los coeficientes obtenidos tras atravesar el filtro y la reconstrucción a partir de los datos anteriores.

Por último queda comentar el manejo que se hace en Jupyter-Notebook de los mensajes enviados por ModelSim, el cual, cada cierto tiempo, envía al banco de verificación los números de imágenes ya filtradas y la fecha y hora en la que se terminó el filtrado de cada imagen. Esta información permite comprobar la cantidad de imágenes que ya han sido procesadas, cuánto tarda en procesarse cada una y si, por alguna razón, la simulación se ha colgado en algún punto. Además de estos mensajes, ModelSim también informa al final de la simulación cuál es el throughput del diseño, de manera que no es necesario calcularlo a mano empleando fórmulas deducidas a partir del funcionamiento de éste. Todos estos mensajes junto con los errores cuadráticos de cada imagen se muestran por terminal, pero también se almacenan en un archivo con extensión ".log" para que pueda tenerse un registro de la ejecución de la verificación y permitir comprobar si existen errores aunque Jupyter-Notebook se haya cerrado o colgado.

7.3 Resultados de la verificación

Creado el banco de verificación y explicadas sus características más importantes es posible comenzar el proceso de verificación, ajuste y caracterización, procesos que se llevarán a cabo en este orden. Al comienzo de la creación del banco de verificación se comentó que, a pesar de estar ubicados en capítulos distintos por motivos de organización, tanto el banco como el diseño hardware se han desarrollado en paralelo, haciendo necesario el uso de un prototipo en el caso del banco para poder realizar pruebas de funcionamiento y aplicar las correcciones necesarias sin necesidad de esperar a tener completado el diseño. Por este motivo es necesario sustituir el prototipo hardware por el diseño, aplicando las modificaciones oportunas para que éste funcione correctamente, de la misma manera que el prototipo hardware sustituyó al prototipo software durante la creación del banco de verificación. Estas modificaciones consisten básicamente, en sustituir los tipos con signo por tipos sin signo tanto en SystemVerilog como en Python y realizar cambios menores para gestionar de manera correcta los datos debido a los cambios de funcionamiento sufridos en la memoria de intercambio para que ésta pueda ser sintetizable.

Aplicados estos cambios el diseño está listo para comenzar la tarea de verificación, la cual se llevará a cabo utilizando la interfaz gráfica de ModelSim, la cual permite representar todas las señales y parámetros que intervienen en el diseño. Con esta representación gráfica pueden detectarse los errores de diseño de manera rápida y eficaz dado que es posible encontrar el foco del problema, aunque una vez terminada la fase de verificación del diseño, cuando quede demostrado que el diseño funciona, se prescindirá de la interfaz gráfica para disminuir el tiempo empleado en la simulación y automatizar el proceso. En esta fase se introducen pequeños conjuntos de imágenes, de 4 o 5 como máximo, las cuales están formadas por patrones de números, lo que permite estudiar más fácilmente los resultados y detectar problemas, algo que sería mucho más complejo de emplear las imágenes obtenidas por el escáner. Hay que comentar además que el tamaño de las entradas y la salida se ha establecido en valores elevados, de 20 o 25 bits empleando gran cantidad de éstos en representaciones la parte entera, con lo que se intenta eliminar el problema del desbordamiento en caso de que los datos sean demasiado grandes. Posteriormente se modificarán estos valores para que el diseño sea más rápido y eficiente en el uso de recursos de la FPGA.

Tras llevar a cabo la tarea de verificación, la cual es mayormente manual debido al proceso de depuración de código que conlleva, se puede comenzar a realizar los ajustes pertinentes al diseño para obtener un menor tamaño y utilización de recursos y, adicionalmente como consecuencia de esta simplificación, una mayor velocidad máxima de funcionamiento. Para ello deberán realizarse ajustes en el número de bits empleados a la entrada del sistema, en los coeficientes del filtro y en la salida, lo que incluye ajustar también el número de bits dedicados a representar la parte entera y la parte decimal, obteniéndose los datos representados en la Tabla 7.1 tras realizar los pertinentes ajustes. Con estos ajustes, los cuales han demostrado tener la mejor relación entre tamaño y errores resultantes, se ha procedido a lanzar la simulación del diseño hardware, teniendo en cuenta que se usarán sólo 6000 imágenes para realizar la simulación, ya que de emplear todas las imágenes disponibles la simulación podría durar hasta varios días. De esta manera, si bien los resultados de esta simulación , mostrados en la Fig. 41,

no son comparables a los resultados de las simulaciones software por la menor cantidad de muestras, son lo suficientemente representativos como para dar el visto bueno al diseño. En la Fig. 41 se puede observar como todas las distribuciones se encuentran centradas en valores muy próximos a cero, siendo el peor valor -0.071 mm, algo casi despreciable, lo que indica un buen comportamiento. Además se puede observar cómo la desviación de los errores es reducida, alcanzando un máximo en 0.411 mm para la desviación de la coordenada Z, teniendo en todos los casos una cierta simetría respecto al 0, aunque no se ajusta tanto como el prototipo software.

Datos			Coeficientes del filtro			Salida		
Entero	Decimal	Total	Entero	Decimal	Total	Entero	Decimal	Total
9	1	10	1	6	7	13	2	15

Tabla 7.1: Resultados del ajuste de tamaños de las entradas y la salida del módulo DWT2

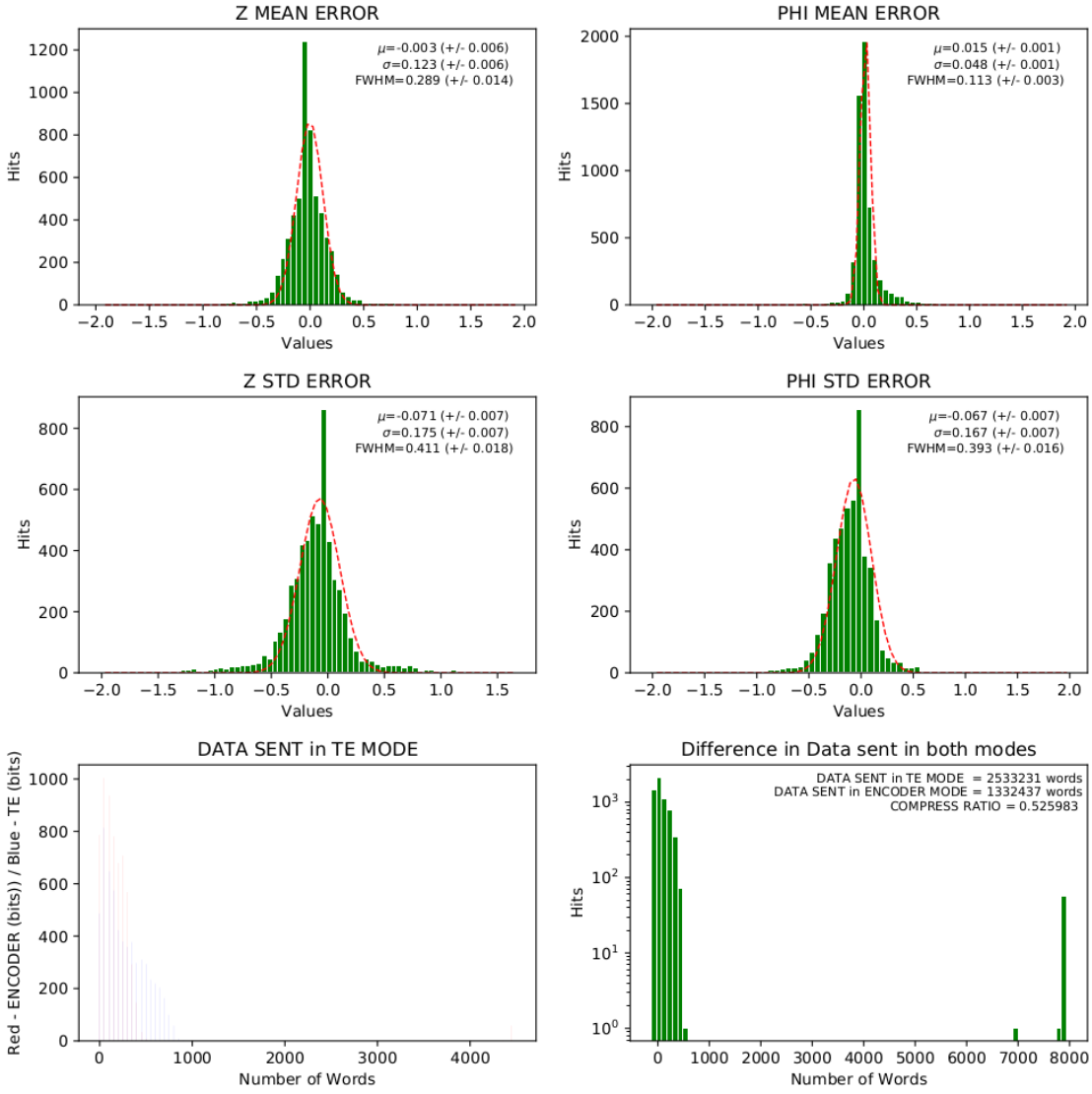

Fig. 41: Resultados de la verificación del hardware

A partir de estos resultados es posible comprobar que el funcionamiento del diseño es correcto y no añade un error excesivo a la información que se desea extraer a partir de las imágenes, pero no habla del grado en el que la compresión afecta a las imágenes. Es por ello que se ha considerado generar documentos sobre las imágenes donde se mostrase la original, la reconstrucción y los coeficientes tras realizar el filtrado tal y como se muestra en la Fig. 42 a modo de ejemplo. En este caso se puede observar que sólo existe información sobre la imagen en los coeficientes de baja frecuencia, dado que el resto de coeficientes no están implementados debido a una optimización introducida basándose en las conclusiones obtenidas durante la fase de diseño del prototipo software. Sin embargo se puede observar cómo la imagen reconstruida mantiene la información sobre las formas del evento, las cuales suelen situarse en la baja frecuencia, aunque se pierde casi toda la información relativa a los detalles, la cual suele situarse en alta frecuencia. Esto puede afectar también a la información sobre eventos con dimensiones reducidas, donde los bruscos cambios propician que la información sobre la forma se concentre menos en las frecuencias más bajas. Sin embargo, dado que el resultado general es bastante satisfactorio, no se ha considerado necesario modificar el diseño para implementar coeficientes de media y/o alta frecuencia.

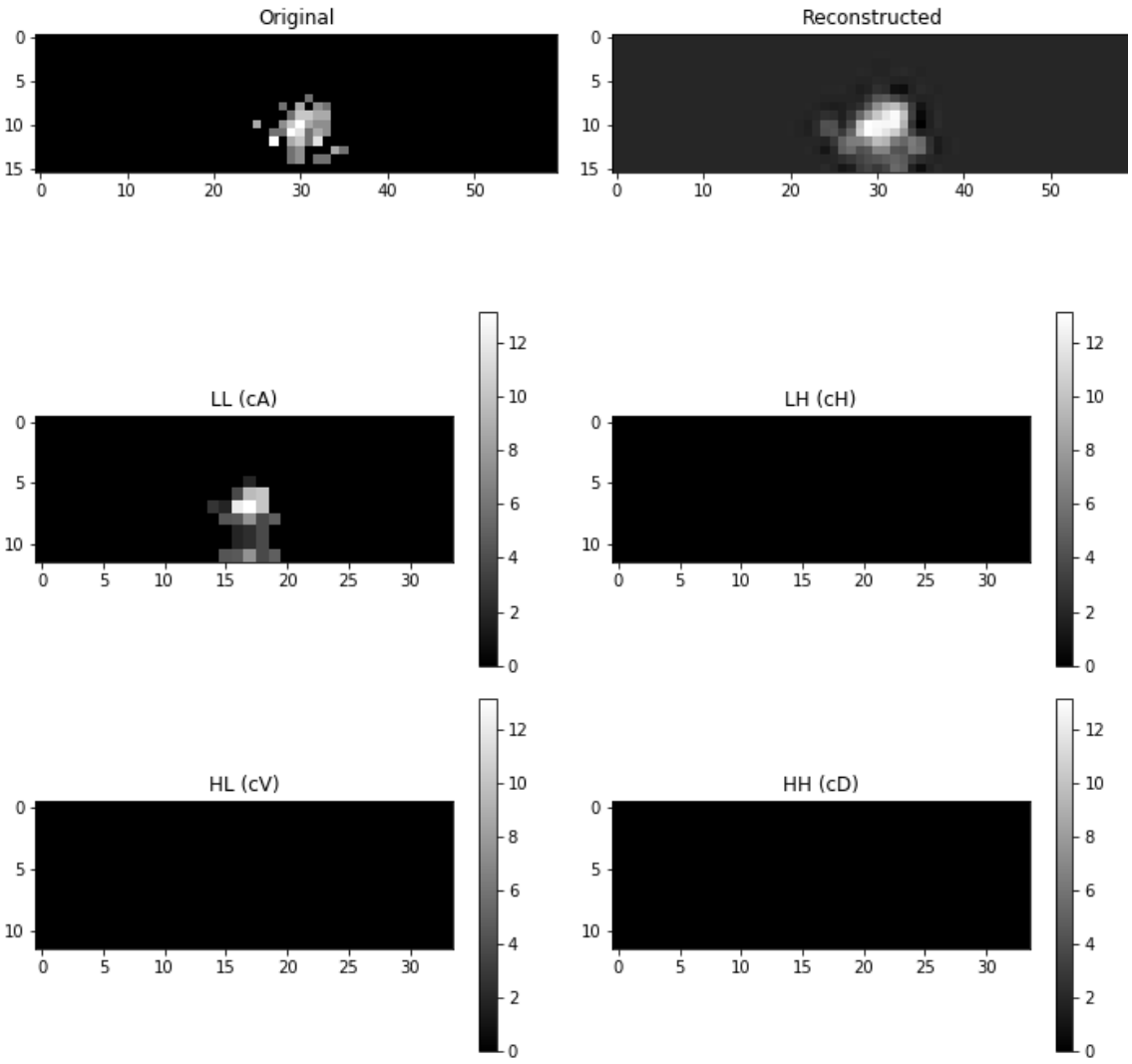

Fig. 42: Imagen original, reconstruida y coeficientes obtenidos tras aplicar la DWT2

Otro tema que es necesario abordar es la codificación empleada por el sistema para enviar los datos, el cual determina en parte el ratio de compresión que aparece en la Fig. 41. Esta codificación es sensiblemente distinta a la empleada durante el diseño del prototipo software y la elección de la wavelet debido a que la codificación anterior no permitía aprovechar todo el potencial de algunas características de este diseño. En concreto, la codificación anterior empleaba algunos bits para identificar qué grupo de coeficientes se enviaba en cada trama, algo innecesario con el diseño actual puesto que únicamente se envían los coeficientes correspondientes a baja frecuencia. Por lo tanto se ha eliminado este identificador de la trama y se ha cambiado por un identificador del tipo de dato, el cual identifica si el dato es un cero o es otro número, siendo el resto de bits de la trama el número de ceros en el primer caso y el valor del píxel en el segundo caso, como se muestra en la Fig. 43. El número de bits enviados en caso de que el valor sea cero se ha establecido en 8 por ser el valor óptimo que envía una menor cantidad de tramas, y en el caso de los datos se ha establecido 10 bits, un tamaño que no compromete los datos, a la vista de los resultados de la Fig. 41, pero que reduce en gran medida la cantidad de bits a enviar realizando un simple diezmado de bits. Con estos datos se logra un ratio de compresión de 0.5260, ligeramente por encima de 0.5198, el dato obtenido en el prototipo software, lo que se debe a que la codificación de trama se aplica tanto a la imagen sin filtrar como a la imagen filtrada, lo que permite obtener datos sobre la compresión usando el módulo. De no usar codificación en la imagen original podrían obtenerse ratios de compresión del orden de 0.02, aunque los datos no servirían tanto para evaluar la compresión que realiza el diseño, sino el conjunto formado por el diseño y el codificador de trama.

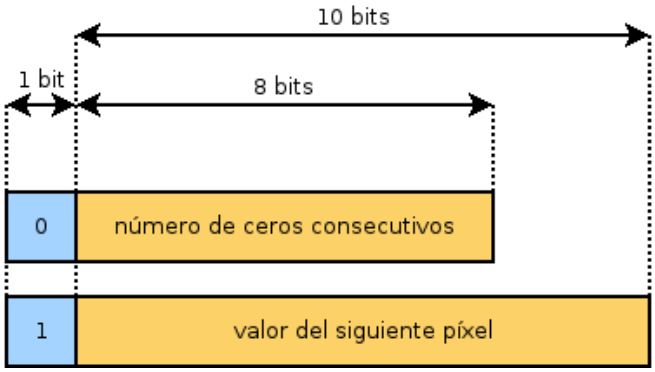

Fig. 43: Tipos de trama enviados por el sistema

Por último es necesario aclarar que las simulaciones se han llevado a cabo a nivel RTL, es decir, antes de implementar el diseño debido a varios motivos, entre los que destaca la lentitud de la simulación. La simulación del hardware a nivel RTL consume una cantidad de tiempo elevada, por lo que, sabiendo que una simulación a nivel de implementación o “Logic-Gate” es más compleja y lenta, podría conllevar tiempos de simulación de más de un día para la misma cantidad de muestras. A esto se une el hecho de que este diseño no forme el sistema completo y por tanto podría observarse un funcionamiento correcto y tras realizar la implementación de todo el sistema presentar fallos debido a la forma en que se generan las señales en otros módulos. Esto unido a la mayor dificultad para configurar este tipo de simulación en el entorno Vivado han provocado que se descarte esta forma de simulación, aunque sería muy recomendable realizarla antes de la implementación del sistema completo para cerciorarse de que éste funciona correctamente antes de introducirlo en la FPGA.

Capítulo 8. Mejoras propuestas para el diseño

Concluido el diseño y las verificaciones realizadas a éste para comprobar su correcto funcionamiento, no está de más hablar de posibles mejoras que podrían realizarse de cara a mejorar su implementación o características.

La primera propuesta para mejorar el diseño pasa por retomar el sistema de redes neuronales para la compresión. Aunque inicialmente se descartó este sistema para realizar la compresión debido a los malos resultados arrojados por las pruebas iniciales, lo cierto es que tiene un gran potencial. Esta nueva implementación podría usar de referencia el compresor desarrollado en este diseño de tal manera que la red se entrene para obtener resultados muy similares a la DWT2. Tras entrenar la red podría realizarse un proceso de simplificación para tratar de eliminar los elementos que tengan menor peso en el resultado final y adicionalmente realizar un entrenamiento partiendo de esta solución para tratar de obtener un diseño con mejor comportamiento. El problema de este diseño es, evidentemente, el tiempo de desarrollo del mismo, ya que primero ha de desarrollarse el compresor wavelet, seleccionar el tipo de red neuronal, dimensionarla, entrenarla y evaluar sus resultados, proceso que previsiblemente será mucho más laborioso. Por tanto se trata de una solución prometedora, aunque su funcionamiento a priori es incierto y requiere de una gran cantidad de trabajo, unido al requerimiento de funcionar en tiempo real, lo que condiciona el tipo de red neuronal elegida y el número de capas, el cual no debe ser demasiado elevado.

Otra posible modificación al diseño consiste en rediseñar el hardware para que realice un filtrado por matrices, en lugar de un filtrado por fila/columna. Esto permitiría simplificar en cierta medida el hardware a cambio de poseer un diseño y teoría de funcionamiento con una complejidad mayor a las del sistema desarrollado. Este diseño podría suponer además un coste en elementos hardware mayor al emplear grandes cantidades de registros para almacenar los datos intermedios y de elementos de filtrado realizados con elementos LUT, pero por contra emplearía una cantidad de memoria mucho menor que el diseño actual.

La última de las modificaciones propuestas para el diseño corresponde, en realidad, al banco de pruebas diseñado para verificar el funcionamiento del hardware. Esta modificación consiste en aprovechar la capacidad de ejecución multiproceso, gracias a la cual, se podrían aprovechar las características de ejecución multihilo presentes en los procesadores de los equipos actuales, lo que permitiría realizar las simulaciones en un tiempo mucho menor. Esto se llevaría a cabo mediante la generación de distintos archivos de datos para cada una de las simulaciones hardware que se llevarán a cabo, repartiendo los datos de manera equitativa. Una vez creados los archivos, se lanzaría una simulación por cada hilo o núcleo de procesamiento, teniendo en cuenta que es recomendable dejar al menos un hilo o núcleo libre para que pueda llevar a cabo otras tareas del sistema y otras aplicaciones menos exigentes. Una vez completadas las simulaciones, las cuales generarían cada una un archivo de resultados, se abrirían los archivos de resultados y se compondrían todos los resultados en una única lista para poder realizar los cálculos estadísticos que permitan evaluar la bondad del diseño. Esta paralelización hace necesaria la posibilidad de determinar los archivos de entrada y salida para cada simulación hardware, los cuales se pueden determinar mediante parámetros en el archivo de compilación del banco de pruebas (archivo .do), los cuales están actualmente soportados. A pesar de este soporte, la complejidad de paralelizar la simulación ha impedido que se pueda llevar a cabo dicha paralelización en el contexto de este diseño, aunque se podrá llevar a cabo, en caso de ser necesario, para llevar a cabo pruebas en el contexto del proyecto del escáner.

A todo lo anterior es necesario añadir que no se han llevado a cabo optimizaciones en el código del banco de pruebas, ya que, por una parte no es objeto de estudio la optimización de software en interpretado reescribiéndolo en lenguajes compilados como C o C++, los cuales se pueden usar en combinación con Python de manera relativamente simple, y por otra parte requiere tiempo y esfuerzo adicionales. Esto unido a que, por ejemplo, en las simulaciones hardware el tiempo requerido por la propia simulación es muy superior al necesario para generar archivos y estadísticas hace que no resulte de interés esta opción, ya que aportaría pocas ventajas y tendría por contra el costoso desarrollo en tiempo de las funciones.

Capítulo 9. Bibliografía

[1] José Millet Roig, Antonio Cebrián Ferriols, María Guillem Sánchez, "Apuntes de la asignatura Instrumentación Biomédica", Curso 2018-2019.

[2] Ruth E. Schmitz, Adam M. Alessio, Paul E. Kinahan, "The Physics of PET/CT scanners", Imaging Research Laboratory, Department of Radiology, University of Washington http://depts.washington.edu/imreslab/education/Physics%20of%20PET.pdf

[3] Open Medscience "Diagnostic medical imaging modalities" http://openmedscience.com/wp-content/uploads/2016/03/positron-electron-annihilation.png

[4] William W. Moses, "Fundamental Limits of Spatial Resolution in PET", Nucl Instrum Methods Phys Res A. 2011 Aug 21; 648 Supplement 1 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3144741/

[5] Renaissance School of Medicine, "PET/MRI", Department of Radiology https://renaissance.stonybrookmedicine.edu/radiology/patients/petmri

[6] PhysicsOpenLab, "Scintillation Crystals", http://physicsopenlab.org/2017/08/10/scintillator-crystals/

[7] Qwerty123uiop, "Schematic view of a photomultiplier coupled to a scintillator, illustrating detection of gamma rays", Wikimedia 2013 https://upload.wikimedia.org/wikipedia/commons/5/5f/PhotoMultiplierTubeAndScintillator.jpg

[8] V. Herrero-Bosch, R. Gadea, R.J. Aliaga, J. Rodríguez, J.F. Toledo, R. Torres-Curado, F. Ballester, R. Esteve, J.J. Gómez-Cadenas and P. Ferrario, "PETALO read-out: A novel approach for data acquisition systems in PET applications", IEEE Medical Imaging Conference 2018, Sidney, Australia.

[9] Air Liquide Group, "Physical properties of Xenon", https://encyclopedia.airliquide.com/xenon

[10] Amara Graps, "An Introduction to Wavelets", https://www.eecis.udel.edu/~amer/CISC651/IEEEwavelet.pdf

[11] Omegatron, “Haar wavelet”, Wikimedia 2005, https://upload.wikimedia.org/wikipedia/commons/thumb/a/a3/Haar_wavelet.png/1024px-Haar_wavelet.png

[12] Vicente Herrero Bosch, “Implementación de la Transformada Wavelet-1D sobre FPGA”, Trabajo Final de Carrera, Director: Rafael Gadea Gironés, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Valencia, Departamento de Ingeniería Electrónica.

[13] Filip Wasilewski, “Wavelet Properties”, PyWavelets 2008-2019, http://wavelets.pybytes.com/

[14] Steve Arar, “Pipelined Direct Form FIR Versus the Transposed Structure”, All About Circuits 2018, https://www.allaboutcircuits.com/technical-articles/pipelined-direct-form-fir-versus-the-transposed-structure/