Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería en Tecnologías de la Telecomunicación

Trabajo Fin de Grado

Diseño de una arquitectura eficiente para la Transformada Discreta del Coseno (DCT) en un dispositivo FPGA

Autor: Javier Viana Gordo

Tutor/es: Álvaro Hernández Alonso

2019

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

GRADO EN SISTEMAS DE INFORMACIÓN

Trabajo Fin de Grado

Diseño de una arquitectura eficiente para la Transformada Discreta del Coseno (DCT) en un dispositivo FPGA

Autor: Javier Viana Gordo
Tutor: Álvaro Hernández Alonso

TRIBUNAL:

Presidente: Ignacio Bravo Muñoz
Vocal 1º: Carlos Julián Martín Arguedas
Vocal 2º: Álvaro Hernández Alonso

FECHA: 27 de Septiembre de 2019

Índice de contenidos

Índice de figuras

Índice de tablas

Resumen

En este Trabajo Fin de Grado se pretende abordar el diseño de una arquitectura eficiente para la implementación en tiempo real en un dispositivo FPGA de la Transformada Discreta del Coseno DCT. Para ello, se explorará la tipología de arquitectura más adecuadas (paralela, secuencial, mixta), las restricciones de tiempo real y de consumo de recursos, así como el efecto de la representación en coma fija. Por último, la arquitectura será configurable en ciertos parámetros como el número de puntos o el ancho de palabra, para poder adaptarse a distintos estándares y requisitos en cada caso.

Palabras clave

VHDL: Lenguaje de descripción hardware.

FPGA: Field-Programmable Gate Array.

DCT: Transformada Discreta del Coseno, *Discrete Cosine Transform.*

Representación en coma fija.

Abstract

This Bachelor's Thesis (TFG) aims to treat the design of an efficient architecture for real-time implementation on a FPGA device of the Discrete Cosine Transform (DCT). For that purpose, the most appropriate architecture typology (parallel, sequential, mixed), the real-time and resource consumption restrictions, as well as the effect of the fixed-point representation will be explored. Finally, the architecture will be configurable by specific parameters, such as the number of points or the word width, to be able to adapt different parameters and requirements for each case or application.

Resumen extendido

Hoy en día, la Transformada Discreta del Coseno (DCT, *Discrete Cosine Transform*) es una herramienta habitual en numerosas aplicaciones vinculadas al tratamiento digital de señal. Entre ellas, por su impacto actual, destacan distintas técnicas de modulación multi-portadora, asociadas a estándares actuales de banda ancha en comunicaciones inalámbricas o PLC (*Power Line Communications*).

Se comenzará con el estudio de la DCT para su posterior modelado matemático en el entorno de Matlab©, debido a la necesidad de estudiar las limitaciones y problemas que implica la implementación en la FPGA. Consistirá en una revisión detallada de trabajos previos y alternativas, para de esa forma enfocar de forma correcta las siguientes fases de diseño de una arquitectura eficiente para la implementación de la DCT.

En esta primera parte se abordará una representación en coma flotante por medio de Matlab©, que posteriormente pasará al estudio de la representación en coma fija en el mismo modelo de simulación. Esto permitirá así estudiar el error que se produce entre ambas representaciones, determinándose así el ancho de palabra óptimo de las diferentes señales del sistema, tratando de obtener la mejor precisión posible y minimizando los recursos del sistema. Este estudio se realiza a través de un modelo totalmente automatizado, por lo que, si se produce un error considerable entre ambas representaciones, se puede ir variando el ancho de palabra hasta que se considere aceptable.

El estudio de este error se realiza debido a que el dispositivo FPGA considerado para la implementación final hace uso de la representación en coma fija, por lo que ese error de cuantificación resulta clave para evaluar las prestaciones finales de la arquitectura propuesta.

A continuación, tras el estudio en Matlab© en coma fija y con los anchos de palabra ya fijados, se procederá al diseño de la arquitectura, evaluando aspectos como la topología, la segmentación, la gestión de recursos compartidos, o los rendimientos (*throughputs*) obtenidos en cada salida. La opción elegida se especificará en VHDL, con una arquitectura en la cual se puedan configurar ciertos parámetros como el número de puntos o la anchura de la palabra.

Para finalizar, se hará una comparativa entre los resultados dados por la implementación en la FPGA y los dados por Matlab© con su representación en coma flotante, estudiando y validando de nuevo el error producido. El trabajo finalizará con un estudio de los resultados obtenidos, verificando la propuesta mediante los correspondientes resultados experimentales.

Capítulo 1: Introducción

Se prevé que la demanda de banda ancha en los próximos años tenga un crecimiento vertiginoso. La modulación multi-portadora permite superar las restricciones impuestas por los canales de comunicaciones. Por ello, se realizará el diseño de la DCT como herramienta a usar en dicha aplicación.

Antes de comenzar con el diseño, se debe realizar un estudio previo sobre el funcionamiento propio de la DCT. La realización de la DCT es posible con múltiples algoritmos, por lo que se debe tratar de elegir el algoritmo más idóneo para su diseño. Además, se tratará de buscar una configuración que permita desarrollar una arquitectura lo más flexible posible.

Una vez elegido dicho algoritmo, se desarrollará una modelo ideal de la DCT a través de Matlab©. Este modelo será la base del estudio, ya que los resultados de la DCT a desarrollar deberán de ser lo más parecido posible a los del modelo ideal.

Debido a que el diseño a realizar presentará ciertas limitaciones, se debe crear un modelo acorde a ello. Se deberá de ajustar determinados parámetros de este modelo, tratando obtener resultados que apenas difieran con los del modelo ideal. Con la ayuda de estos modelos, se podrá elegir la mejor forma de realizar el diseño de la arquitectura.

Con las conclusiones que se obtengan del estudio de los modelos, se procederá al diseño de la arquitectura de la DCT. Pudiendo hacer uso de los modelos desarrollados en Matlab© para validar las salidas de nuestra arquitectura.

A continuación, se va a tratar de explicar el contexto, los objetivos y la estructura del presente documento.

1.1. Contexto

La Transformada Discreta del Coseno (DCT, *Discrete Cosine Transform*) tiene múltiples aplicaciones, de las cuales destacan las comunicaciones en banda ancha o la compresión de imágenes, audio y video. Cuyo desarrollo se lleva a cabo gracias al tratamiento digital de señales. Este estudio se ha realizado dentro del Departamento de Electrónica, en el grupo de investigación GEINTRA, el cual está especialmente relacionado con la asignatura de Diseño Electrónico.

1.2. Objetivos

El objetivo principal de este TFG es el diseño de una arquitectura eficiente para la implementación en tiempo real en un dispositivo FPGA de la DCT. Para la realización de esta tarea se hará un desglose en los siguientes objetivos parciales:

- Investigación de los antecedentes de la DCT.

- Diseño de un modelo teórico con representación en coma flotante.

- Haciendo uso del modelo en coma flotante, se hará un diseño del mismo sistema en coma fija.

- Estudio del error que se produce entre ambas representaciones del mismo sistema, siendo los resultados de la representación en coma flotante del sistema, los ideales y a los cuales se tratará de acercar con la representación en coma fija.

- Selección del número de bits necesarios para la representación de cada señal del sistema.

- Estudio del diseño en VHDL. Se procederá a realizar este estudio, observando la salida de cada etapa que compone la DCT y haciendo una comparativa con el modelo en coma fija ya mencionado anteriormente.

- Conclusiones de los resultados obtenidos.

Una vez definidos los objetivos se podrá definir la estructura del documento, que se encuentra descrito a continuación.

1.3. Estructura del documento.

A continuación, se expondrán los principales capítulos de los cuales se compone este TFG. Además, se hará una pequeña explicación de cada uno de ellos.

- **Antecedentes y trabajos previos**: En este capítulo se tratará de hacer una breve explicación sobre la DCT y las FPGAs. Se pondrán ejemplos de sus principales campos de aplicaciones, tratándose de dar a entender su relevancia en el mercado actual.

- **Diseño de modelos teóricos**: En este punto, se tratará de explicar la importancia de la realización de un modelo teórico en coma flotante, el cual permitirá la posterior cuantificación de las señales. Gracias a ello se podrá hacer un estudio del error que se produce entre el modelo de representación en coma flotante y el de coma fija. Y con ese estudio se procederá a la decisión del número de bits con el que se podrá representar cada señal.

- **Diseño de la DCT en VHDL**: Este capítulo se encuentra dedicado a la creación de un modelo en VHDL. Se tratará de explicar la solución adoptada para el diseño de este modelo, describiendo las etapas en las cuales se decide dividir el sistema. Este capítulo estará compuesto por las diferentes etapas, tratando de explicar su desarrollo y su funcionalidad. Por cada etapa se mostrarán los diferentes resultados que se van obteniendo a la hora de simularlas. Se concluirá con un análisis de los recursos usados por el sistema.

- **Conclusión**: Se trata de describir el trabajo realizado a lo largo del desarrollo del TFG y los resultados que se obtienen del mismo. Se comentarán ciertos problemas con objetivo de obtener conclusiones acerca de nuestro diseño. También se tratarán posibles mejores aplicables a nuestro diseño.

Capítulo 2: Antecedentes y trabajos previos

En este capítulo se va a tratar de explicar de una manera más teórica que es una DCT y los diferentes tipos que existen. Se pasará a explicar las aplicaciones que se le pueden dar, haciendo hincapié en alguna aplicación más destacada.

A continuación, se hará una breve introducción a las FPGAs y su arquitectura básica. Tras ello se enumerarán algunos de sus campos de aplicación y se pondrán ejemplos de algunos de los sectores con más demanda hoy en día.

2.1 Transformada Discreta del Coseno (DCT)

La Transformada Discreta del Coseno (DCT) es esencialmente la DFT (Transformada Discreta de Fourier) pero haciendo uso de números reales. La DCT se construye de tal manera que se concentra una gran cantidad de energía en su espectro. Las variantes de la DCT (como DCT-I, II, III, IV) es el resultado de diferentes combinaciones.

En este trabajo se ha hecho uso de la DCT-IV. La DCT-IV viene definida por (1):

$$p[\mathrm{k}] = \sum_{n=0}^{M-1} x[n] \cdot \cos\left[\frac{\pi}{M}\left(n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right], \quad k = 0, \dots, M-1. \qquad (1)$$

Donde *M* es el número de puntos a la entrada, $p[\mathrm{k}]$ es la señal de salida de la DCT y $x[n]$ es la señal de entrada.

Teniendo en cuenta (1), se puede dividir el módulo DCT en 4 fases como las mostradas en la figura 1:

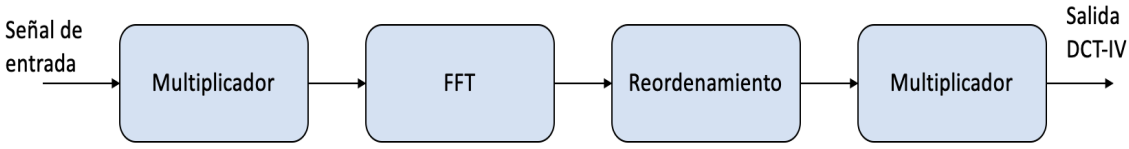

Figura 1. Fases en las que se divide la DCT-IV.

- La señal de entrada $x_m[\mathrm{n}]$ se multiplica por la constante compleja $e^{-jm\pi/2M}$, obteniéndose (2).

$$y_m[\mathrm{n}] = x_m[\mathrm{n}] \cdot e^{\frac{-jm\pi}{2M}} \qquad (2)$$

- Un módulo FFT, denotado como {F} en (3).

$$Y_m[\mathrm{k}] = F\{y_m[\mathrm{n}]\} \qquad (3)$$

- Reordenamiento del array, siguiendo (4) donde i $= 0, \dots, \frac{M}{2} - 1$.

$$Z_{2\cdot i+1}[\mathrm{k}] = Y_i[\mathrm{k}]$$

$$Z_{2\cdot i}[\mathrm{k}] = conj\{Y_{m-i}[\mathrm{k}]\} \quad (4)$$

- Multiplicación por la constante compleja $e^{-j\pi(2m+1)/4M}$, obteniéndose (5).

$$p_m[\mathrm{k}] = 2\cdot Re\left\{Z_m[k]\cdot e^{-\frac{j\pi(2m+1)}{4M}}\right\} \quad (5)$$

La matriz DCT-IV se vuelve ortogonal al multiplicarse por un factor de escala global $\sqrt{2/M}$. La simetría de la matriz de transformación indica que los algoritmos para el cálculo de transformada directa e inversa son idénticos. La implementación de la DCT-IV inversa y la DCT-IV será la misma, pudiéndose usar la misma función de proceso.

2.2. Campos de aplicación de la DCT

La DCT tiene múltiples aplicaciones, desde banda ancha hasta codificar video. En este apartado se tratará, a modo de ejemplo, una de las aplicaciones más destacadas de la DCT, la compresión de imágenes. Debido a que el uso y la dependencia de los ordenadores no para de crecer, nuestra necesidad de almacenar una gran cantidad de datos también lo hace. Si se pone el ejemplo de una tienda online, se haría necesario el uso de docenas o cientos de imágenes; y, por lo tanto, es imprescindible el uso de alguna técnica de compresión de imágenes para poder así almacenarlas.

La cantidad de espacio que se requiere para almacenar una imagen sin ser alterado su contenido puede ser demasiado grande. Afortunadamente hay diferentes métodos de compresión de imágenes hoy en día. Se consideran dos categorías: compresión de imágenes con y sin pérdidas. El proceso JPEG, que es extensamente usado para la compresión de imágenes con pérdidas, se centra en la DCT.

La DCT se usa para separar imágenes en partes de diferentes frecuencias. Cuando la compresión se produce, las frecuencias menos importantes son descartadas, de ahí que se denomine una compresión con pérdidas. Como consecuencia, solo las frecuencias más importantes quedan, y son las que se usarán para la descompresión de la imagen. Como consecuencia, habrá una distorsión en la imagen reconstruida, pero se puede controlar este nivel de distorsión en la compresión.

El proceso que sigue la compresión JPEG es el siguiente:

- La imagen se divide en 8x8 bloques de píxeles.
- La DCT se aplica a cada bloque siguiendo un orden determinado.
- Cada bloque es comprimido.

- El array de bloques comprimidos que constituye la imagen se guarda en un espacio considerablemente menor.
- La imagen se puede reconstruir a través de un proceso de descompresión, a través de la IDCT (Inverse Discrete Cosine Transform).

También destaca su uso en las técnicas de modulación multi-portadora para comunicaciones de banda ancha, las cuales destacan por un rendimiento superior y una mayor eficiencia espectral. Debido a ello, la implementación de transceptores multi-portadora se ha vuelto cada vez más importante en la última década.

Como se ha tratado de explicar, la DCT está presente en diversos sectores con múltiples aplicaciones. Ahora se pasará a explicar la tecnología con la que se llevará a cabo el desarrollo de la misma.

2.3. Field-Programmable Gate Array (FPGA)

Una FPGA (*Field-Programmable Gate Array*) es un dispositivo semiconductor que contiene bloques lógicos programables e interconexiones de los mismos también programables. Las FPGAs nacieron de la idea de combinar el control del diseñador y el time-to-market corto de los PLDs (*Programmable Logic Device*, Dispositivo lógico programable), con la densidad y bajo costo de arrays de puertas.

El diseñador, haciendo uso de lenguaje de descripción hardware (VHDL), es capaz de realizar un sistema digital, el cual se podrá volcar en lógica digital. Se puede comprobar que la funcionalidad del sistema digital es correcta gracias a las simulaciones, las cuales se deben realizar antes de volcar la aplicación en la propia FPGA. Esto posibilita el diseño hardware de múltiples aplicaciones haciendo uso de un único dispositivo. De igual manera facilita la reconfiguración del diseño hardware sin necesidad de realizar alteraciones en la tarjeta o plataforma.

El elemento en las FPGAs más presente son los bloques lógicos con sus recursos de interconexión. Una jerarquía de interconexiones programables proporciona una conexión entre los bloques lógicos de la FPGA según las necesidades del diseñador, pudiendo así desempeñar cualquier función lógica que se necesite. Aunque hay varias familias de FPGAs, presentan algunos elementos en común, tales como bloques de memoria que pueden ir desde un flips-flops hasta bloques más complejos. También se encuentran presentes elementos de entrada y salida y lógica combinacional. Además, suelen presentar bloques capaces de realizar multiplicaciones, sumas y restas especialmente útil para la realización de las diversas operaciones requeridas.

Las FPGAs presentan recursos finitos por lo que una representación en coma flotante no es posible, por ello hacen uso de la representación en coma fija. Debido a esto se debe hacer un estudio de la DCT en coma fija a través de Matlab©, para poder observar el error entre ambas representaciones y poder así obtener el tamaño de las señales a usar en el diseño.

2.4. Aplicaciones de las FPGAs

Las FPGAs tienen un campo de aplicación muy amplio debido a la gran flexibilidad y versatilidad que presentan. Los mayores campos de aplicación de estas FPGAs se encuentran en el procesado digital de señales (DSP, *Digital Signal Processing*), bioinformática, comunicaciones, procesado de datos, etc. La elección principal de estos dispositivos para el tratamiento de señales es debido a su alta frecuencia de trabajo, el paralelismo en los procesos y su bajo coste frente otras tecnologías similares.

Uno de los sectores con más demanda hoy en día es el de la visión artificial. Múltiples dispositivos actualmente requieren de estos sistemas, los cuales sean capaces de reconocer objetos en el entorno, rostros y actuar de la manera adecuada tras ello. Esto puede ser usado en cámaras de video vigilancia, automóviles, robots, etc. Todo esto requiere que los sistemas sean capaces de procesar un gran número de imágenes y trabajar con ellas, y la mayoría de las veces debe hacerse en tiempo real.

Otro ejemplo podría ser la codificación y encriptación. La seguridad hoy en día es de suma importancia con relación al envío de mensaje, compras por internet o en sectores como el militar o gubernamental. Las FPGAs en este terreno presentan una ventaja en el procesamiento de grandes cantidades de información y una gran optimización a la hora de realizar operaciones aritméticas.

Capítulo 3: Diseño de modelos teóricos

Este capítulo está dedicado al estudio teórico de la DCT-IV para su posterior implementación. Se centrará en el estudio del error causado por la representación en coma fija impuesta por las necesidades de la propia FPGA. Para poder estudiar este error se comenzará con el desarrollo de un modelo en Matlab© en coma flotante de la DCT-IV. Los resultados de este modelo servirán de manera ideal, como base del estudio. Como ya se ha explicado con anterioridad estos resultados son inalcanzables debido a su tipo de representación, pero se tratará de obtener un modelo lo más cercano posible a lo ideal, tratando a su vez de ser eficientes a la hora de consumir los recursos propios de la FPGA.

Una vez hecho este modelo en coma fija, se tratará la señal de entrada que deberá ser usada en todos los tipos de simulación de los modelos de DCT-IV; tanto en Matlab©, como en VHDL. Para ello se deberá de limitar el valor de esta señal a un rango determinado por un número de bits fijo. A este proceso se le denomina cuantificación y se debe realizar para pasar de la representación de coma flotante a coma fija. Esto ayuda a hacer un estudio de la diferencia entre ambas representaciones y poder ver con cuántos bits se pierde la mínima información posible; además se debe tratar de no hacer un uso excesivo de los recursos.

A continuación, se trata de reproducir lo usado en la señal de entrada con cada una de las señales presentes, así se procederá a la realización del modelo en como fija. Esto servirá como base para la realización más delante para el desarrollo de la arquitectura en VHDL. En la figura 2 se trata de esquematizar de forma breve los temas que se van a tratar en los próximos puntos.

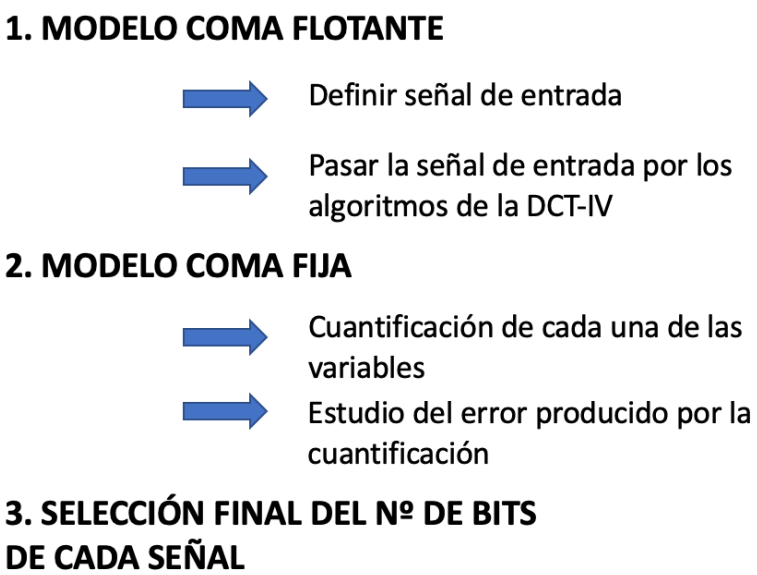

Figura 2. Pasos en nuestro estudio en Matlab©.

3.1. Estudio de los parámetros del sistema

Se pretende realizar un modelo flexible, pudiendo alterar determinados parámetros del sistema y ver de esa forma cómo afectan a los resultados de la DCT. La variable *M* hace referencia al número de puntos de la DCT, debe ser siempre una potencia de dos. El número de bits con los que se va a cuantificar debe ser también un valor variable, el cual se pueda modificar para poder ver su influencia en el error entre las dos diferentes representaciones.

3.2. Diseño del modelo de representación en coma flotante

Se procederá definiendo una señal de entrada $x_m[n]$ para todo nuestro estudio. A modo de ejemplo ilustrativo, pero sin pérdida de ninguna generalidad, se decidió hacer uso de la señal sinusoidal con *M* puntos, la cual se puede ver en la figura 3.

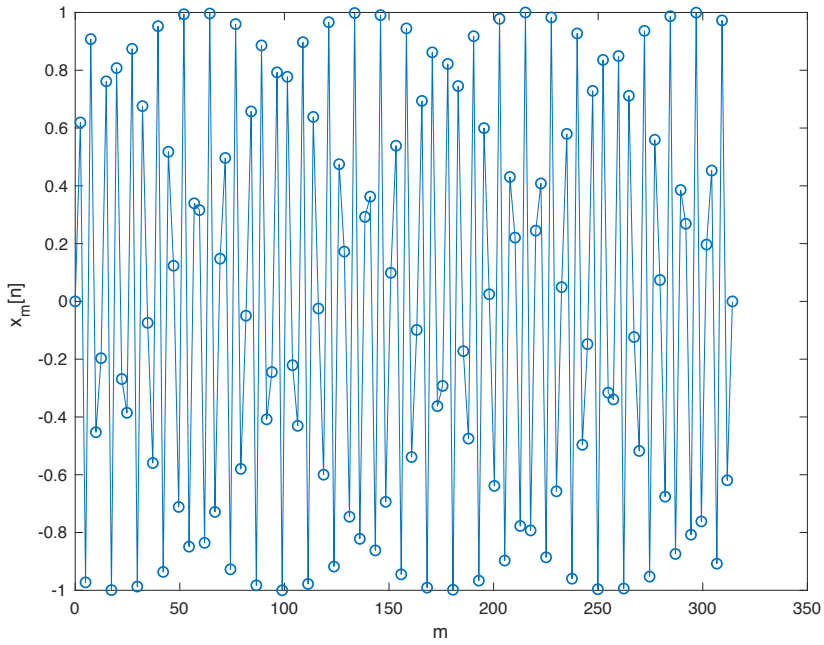

Figura 3. Señal de entrada $x_m[n]$.

Tras tener la señal de entrada, se sacarán los coeficientes que multiplicarán a la señal de entrada, como aparece en (2). En las figuras 4 y 5, se muestra el resultado que se obtendría al representar a través de Matlab© los primeros coeficientes de la DCT y la señal resultante $y_m[n]$ respectivamente.

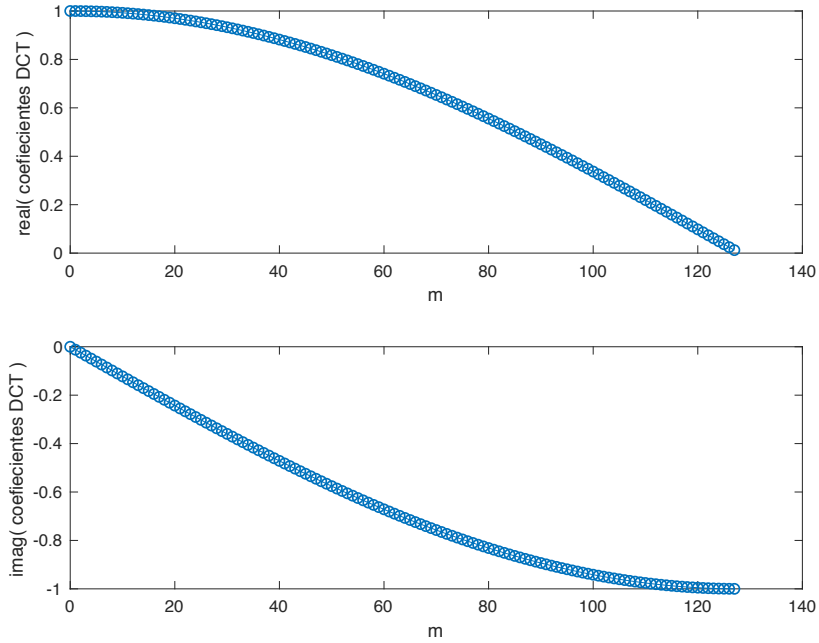

Figura 4. Primeros coeficientes de la DCT.

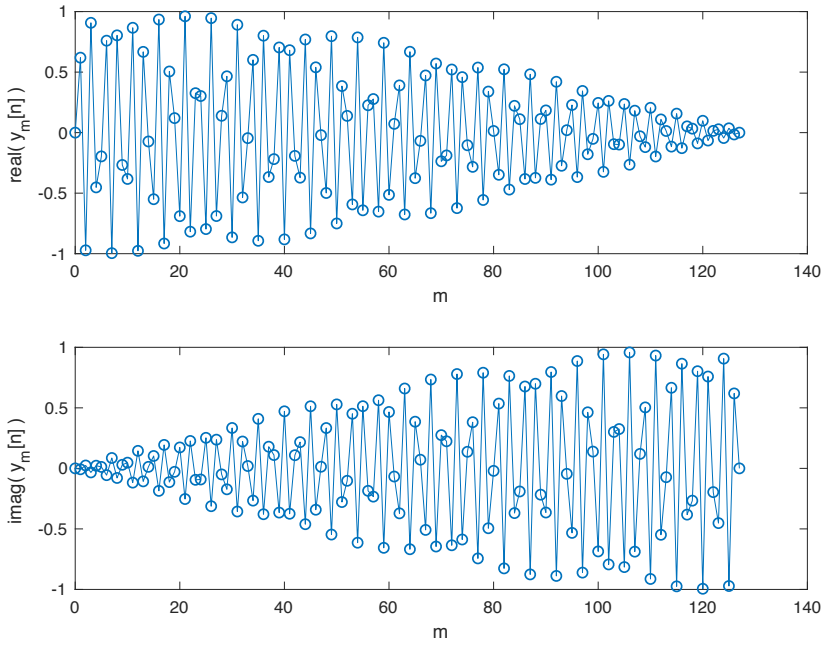

Figura 5. Señal a la salida del primer bloque multiplicador $y_m[n]$.

Una vez obtenida $y_m[n]$, se pasará por el módulo de la FFT como se ve en (3) y se encuentra representado en la figura 6. Con $Y_m[k]$, se pasa a realizar el reordenamiento del array dando como resultado lo obtenido en la figura 7, haciendo uso para ello de la expresión (4).

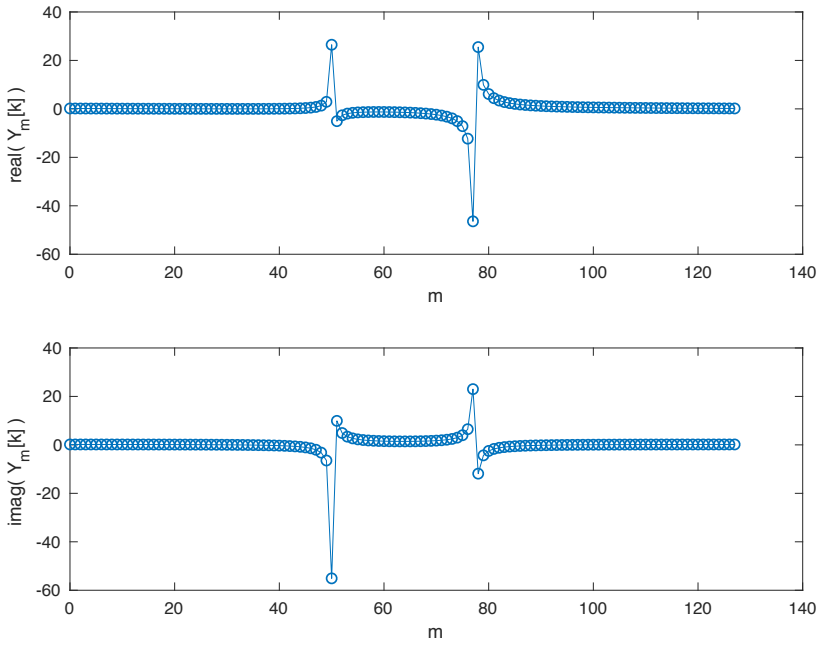

Figura 6.Señal a la salida de la FFT $Y_m[k]$.

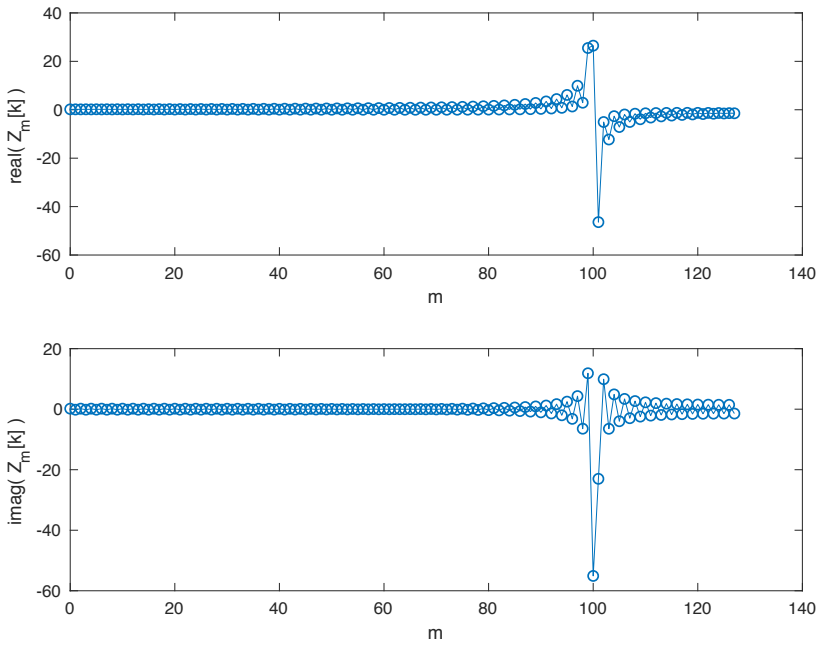

Figura 7. Señal reordenada $Z_m[k]$.

A continuación, se sacan los segundos coeficientes necesarios para la DCT, que se usarán para aplicar (5) y que se encuentran representados en la figura 8. A dicha fórmula se debe añadir una multiplicación por $\sqrt{2/M}$, volviendo así la matriz DCT-IV ortogonal. Dicha fórmula modificada se muestra en (6) y se ve representada en la figura 9.

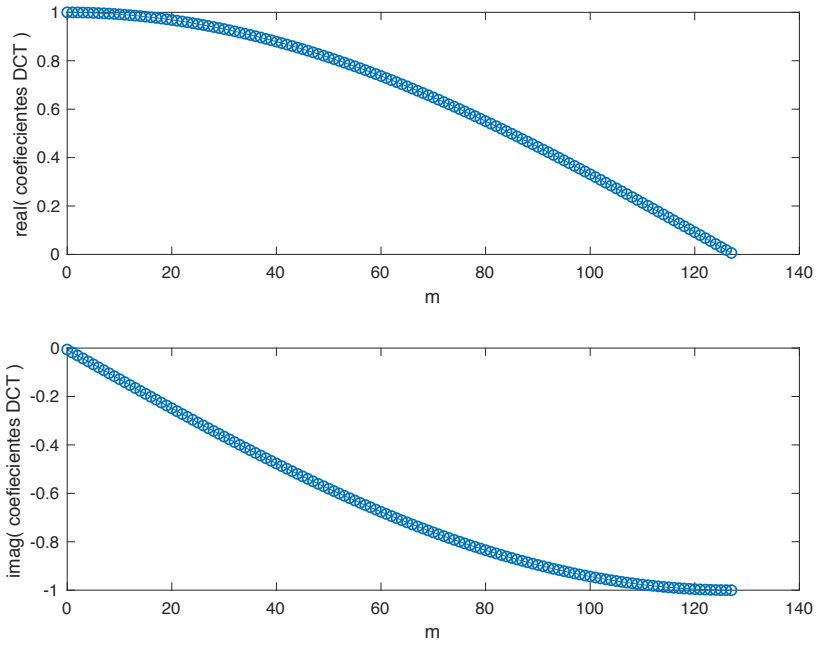

Figura 8. Segundos coeficientes de la DCT.

$$p_m[\mathrm{k}] = \sqrt{2/M} \cdot 2 \cdot Re\left\{Z_m[k] \cdot e^{-\frac{j\pi(2m+1)}{4M}}\right\} \qquad (6)$$

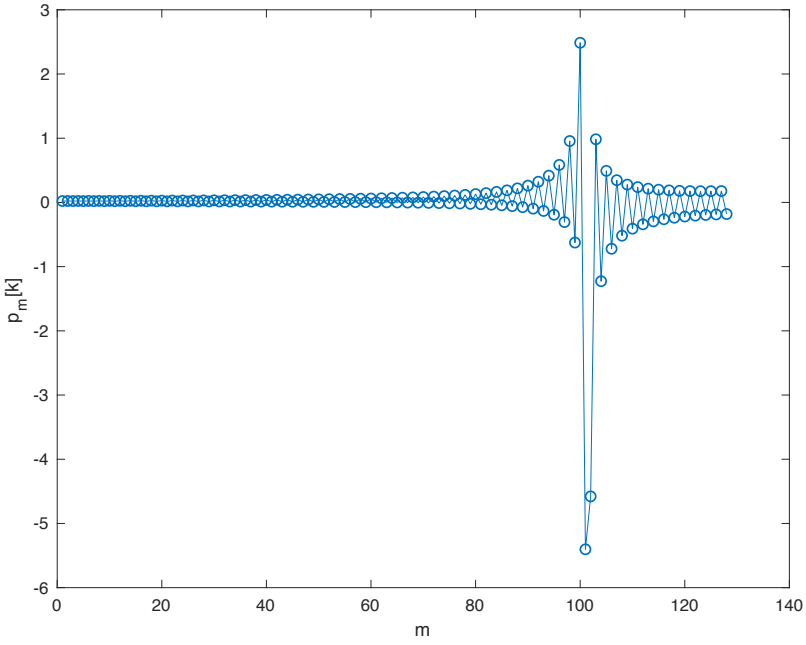

Figura 9. Señal a la salida de la DCT $p_m[k]$.

3.3. Diseño del modelo de representación en coma fija

Se procederá a seguir los mismos pasos que en el modelo de coma flotante, pero en este caso limitando el número de bits de cada señal, tanto de la parte entera como de la fraccionaria. Para esto se hará uso de las funciones *quantizer*()**,** que te permite definir las características de la cuantificación, tales como el tipo de modelo, de redondeo, de overflow y cuántos bits se asignan a la parte entera y a la fraccionaria. La función *quantize*() permite aplicar estas características sobre una señal, produciéndose así la cuantificación deseada.

A continuación, se mostrará cómo varía el error de cuantificación en función de los recursos que se le asignen a cada señal. Hay que tener en cuenta que en la DCT se presentan acciones como el reordenamiento de la $Y_m[\mathrm{k}]$ o Re{}, que no precisan de cuantificación ya que sus valores de la propia señal no se ven alterados.

En la figura 10 se muestra el error máximo de la señal de entrada con ella misma cuantificada. Este error máximo se ira obteniendo cuantificando la señal con un número de bits diferente, pudiéndose así estudiar la progresión del error de la señal de entrada en función de los bits que se codifique, pudiendo tomar así la decisión de cuántos bits se le debe asignar.

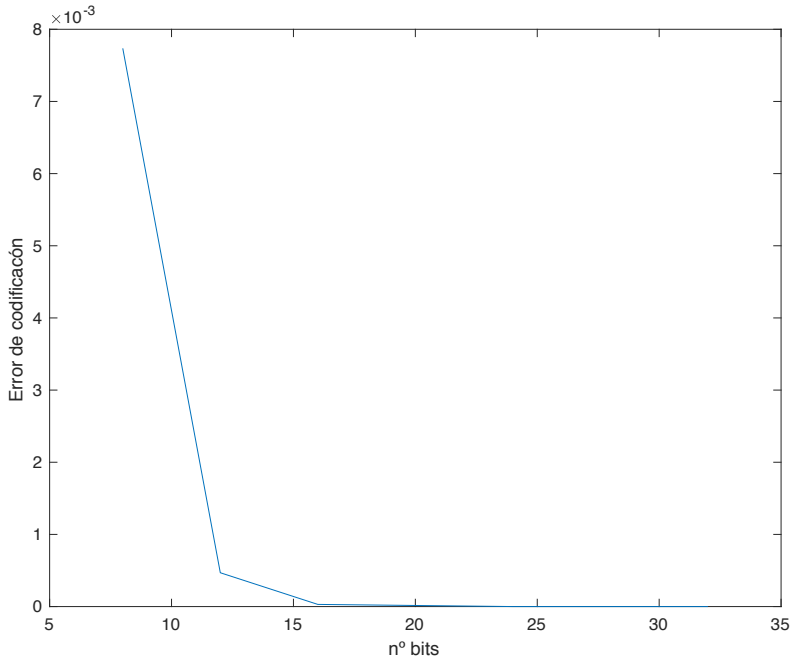

Figura 10. Error producido al cuantificar $x_m[n]$.

En la figura 11 se muestra el error al codificar los primeros coeficientes de la DCT, en función de con cuántos bits se codifique.

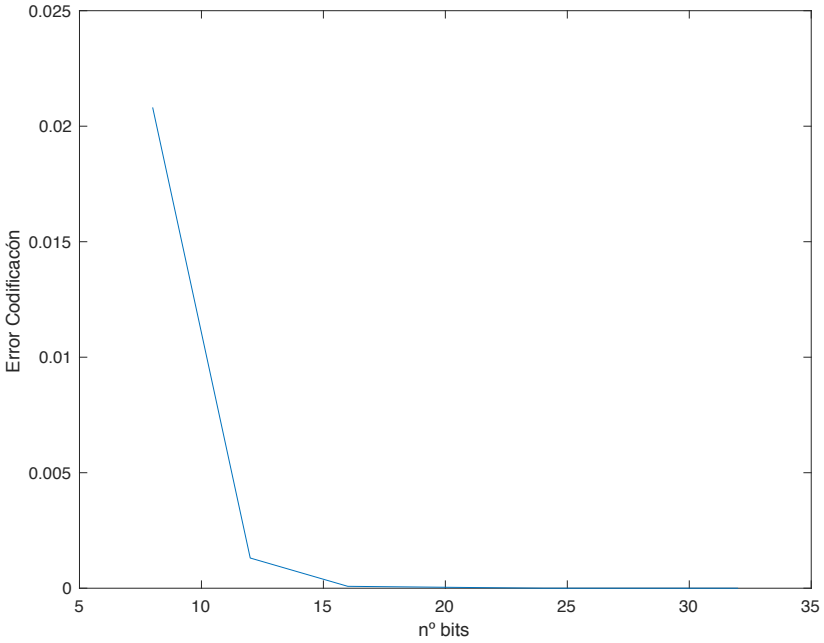

Figura 11. Error producido al cuantificar $e^{-jm\pi/2M}$.

En la figura 12 se encuentra el estudio de la multiplicación de la señal de entrada y los primeros coeficientes de la DCT, en función de los bits con los que se codifique.

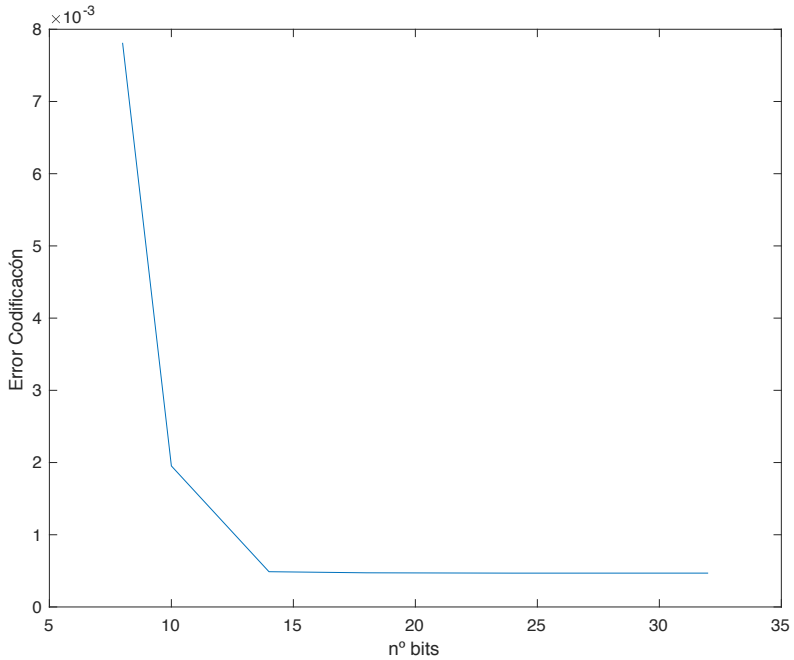

Figura 12. Error producido al cuantificar $y_m[n]$.

En la figura 13 se estudia la señal tras pasar por el módulo de la FFT, viendo el máximo error que se produce al codificar la señal con distintos números de bits.

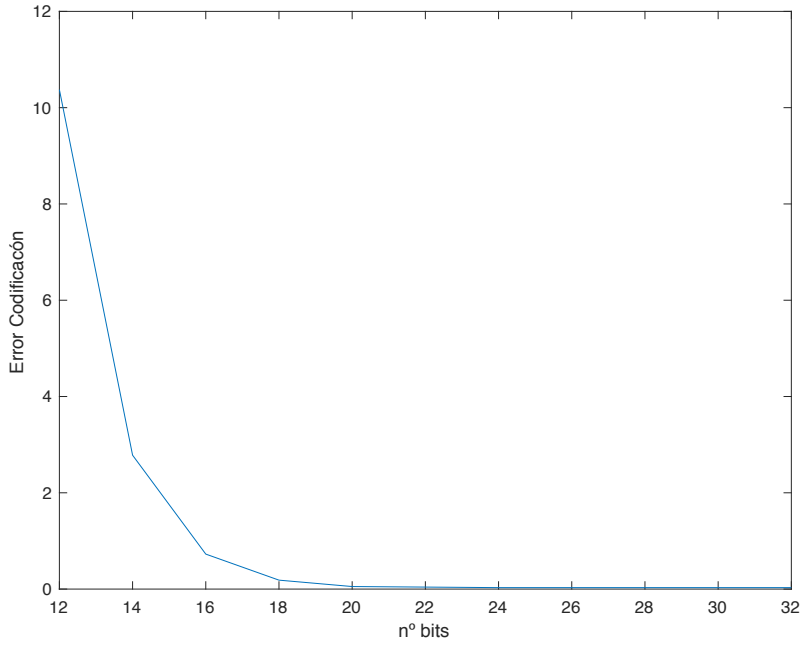

Figura 13. Error producido al cuantificar $Y_m[k]$.

En la figura 14 se muestra el error al codificar los segundos coeficientes de la DCT, en función de del número de bits empleados en la cuantificación.

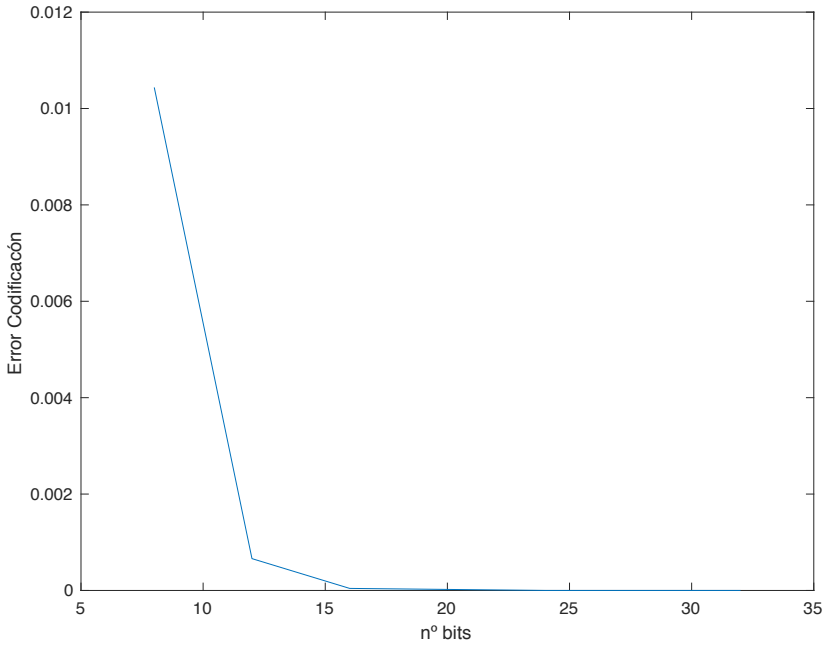

Figura 14. Error producido al cuantificar $e^{-j\pi(2m+1)/4M}$.

Hay que tener en cuenta, que al hacer un modelo en coma fija, hay que cuantificar las señales cada vez que se produzca una operación. Para la obtención de la señal $p_m[k]$, se deben realizar tres operaciones hasta su obtención. Debido a ello se va a tener dos señales intermedias que también deben ser cuantificadas y estudiadas. A continuación, se muestra el estudio de cuantificar las señales intermedias y de la señal final $p_m[k]$ definida en (6), en función del número de bits en las figuras 15, 16 y 17.

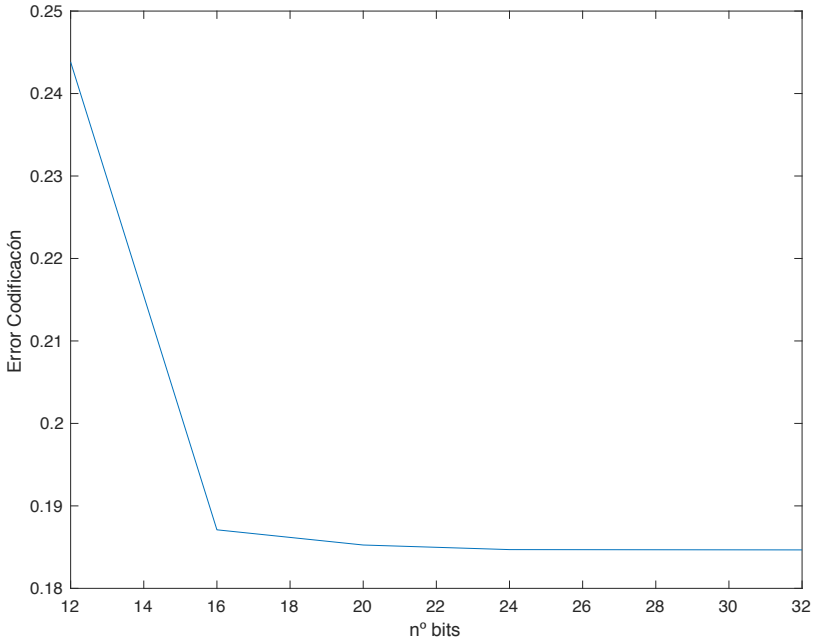

Figura 15. Error producido al cuantificar $\left(Z_m[k] \cdot e^{-\frac{j\pi(2m+1)}{4M}}\right)$.

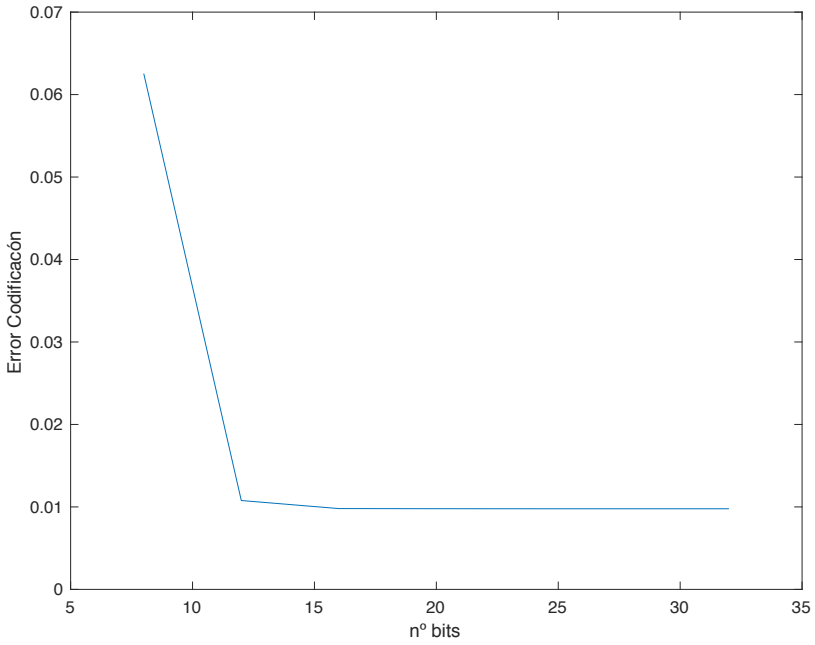

Figura 16. Error producido al cuantificar $\left(\sqrt{2/M} \cdot Re\left\{Z_m[k] \cdot e^{-\frac{j\pi(2m+1)}{4M}}\right\}\right)$.

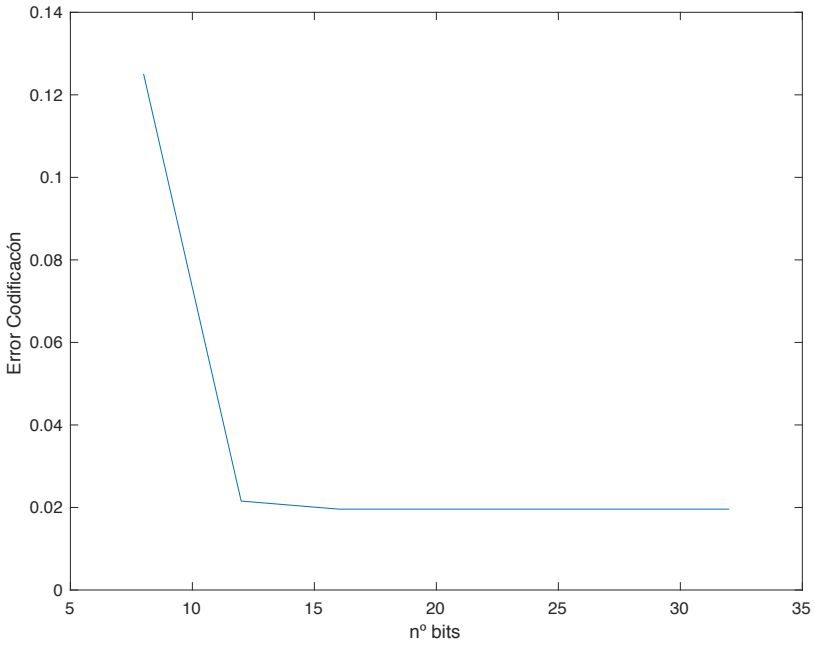

Figura 17. Error producido al cuantificar $p_m[k]$.

Como se puede observar, a partir de cierto valor de bits el error ya permanece casi constante; éste sería el valor adecuado para cuantificar nuestra señal. Aunque también hay que tener en cuenta otras posibles características que pueda presentar la arquitectura, por ejemplo, los buses de entrada de los multiplicadores. Esto se debe tener en cuanta, ya que, si se elige de manera adecuada desde el principio, se evitaría el tener que hacer futuras revisiones, como posibles extensiones de signo de determinadas señales en el futuro diseño.

3.4. Elección final del número de bits de cada señal

Para cada señal que se usa en el modelo de coma fija, a excepción de las que se generan a consecuencia de un reordenamiento del array, se debe fijar un número de bits para su representación. Esta decisión se debe tomar tratando de reducir la pérdida de información y tratando de evitar hacer uso de un gran número de recursos.

A continuación, se mostrará una tabla con los recursos asignados a cada señal del modelo y el error que generaría con respecto al modelo en coma flotante. Se deben definir señales auxiliares que permitan llegar a codificar la señal $p_m[\mathrm{k}]$. Las señales se definirán como: $a_m[k]$ y $b_m[k]$, cada señal se ocupará de una de las operaciones que componen la señal $p_m[k]$. La señal $a_m[k]$ se encarga de multiplicar la señal $Z_m[k]$ con los segundos coeficientes de la DCT, y $b_m[k]$ se va a encargar de almacenar la multiplicación de $\sqrt{2/\mathrm{M}}$ por la parte real de $a_m[k]$.

Señal	Nº bits	Bits parte entera	Bits parte fraccionaria	Error máx. de cuantificación
$x_m[n]$	12	1	11	4.6884e-04
Primeros coeficientes $(e^{-jm\pi/2M})$	18	2	16	1.8963e-05
$y_m[n]$	14	1	13	4.8828e-04
$Y_m[k]$	18	9	9	0.1855
Segundos coeficientes $(e^{-j\pi(2m+1)/4M})$	18	1	17	1.0500e-05
$a_m[k]$	22	9	13	0.1848
$b_m[k]$	17	4	13	0.0098
$p_m[k]$	18	5	13	0.0196

Tabla 1. Detalle de la cuantificación en coma fija empleada en cada una de las señales involucradas en la implementación de la DCT.

Se decide este número de bits para cada señal, debido a que son los valores límites en los cuales el error de codificación ya no disminuye, a excepción de ciertas señales como son los coeficientes. Éstos van a ser entradas de un bloque multiplicador 18x25 (tamaño disponible en la familia 7 de FPGAs de Xilinx, Inc.), y se les va a asignar siempre la entrada de 18 bits, por ello se fijará ese número de bits para su representación. El bloque multiplicador que se va a usar para el diseño viene representado en la Figura 18.

También está el caso de la variable $\sqrt{2/M}$, que se debe cuantificar con un número de bits. Esta variable, para el caso concreto de M=128, tendría un valor de 0.125. Este valor es posible representarlo únicamente con 4 bits (1 bit de parte real y 3 bits de parte imaginaria) sin error alguno. Dado que esta variable será una entrada del bloque multiplicador 18x25, se pasará a representar directamente con 18 bits para no extender bits.

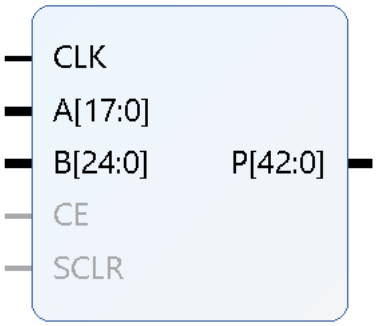

Figura 18. Mult18x25

Una vez definido el número de bits se pasará a calcular el error relativo máximo de la señal de salida. Para ello se hará uso de (7).

$$Error\ relativo\ (\%) = \frac{\max(|\text{Valor exacto} - \text{Valor aproximado}|)}{\max(|\text{Valor exacto}|)} \cdot 100 \qquad (7)$$

Obteniendo un error relativo en la señal de salida tras la cuantifición de 0.7881%.

Capítulo 4: Especificación de la DCT en VHDL

Este capítulo está dedicado a la creación de un modelo en VHDL, con el fin de realizar una arquitectura variable de una DCT. Para ello, se hará uso del software de Vivado Design Suite, producido por Xilinx, Inc. Este software permitirá la realización del diseño y las simulaciones del mismo.

Para un correcto diseño, se debe distinguir las diferentes etapas en las que se encontrará dividido el sistema completo. Para ello se debe tener en cuenta las limitaciones que presenta la creación de un modelo en VHDL. Las etapas en las que se compone la DCT de modo general se encuentra en representado en la figura 19.

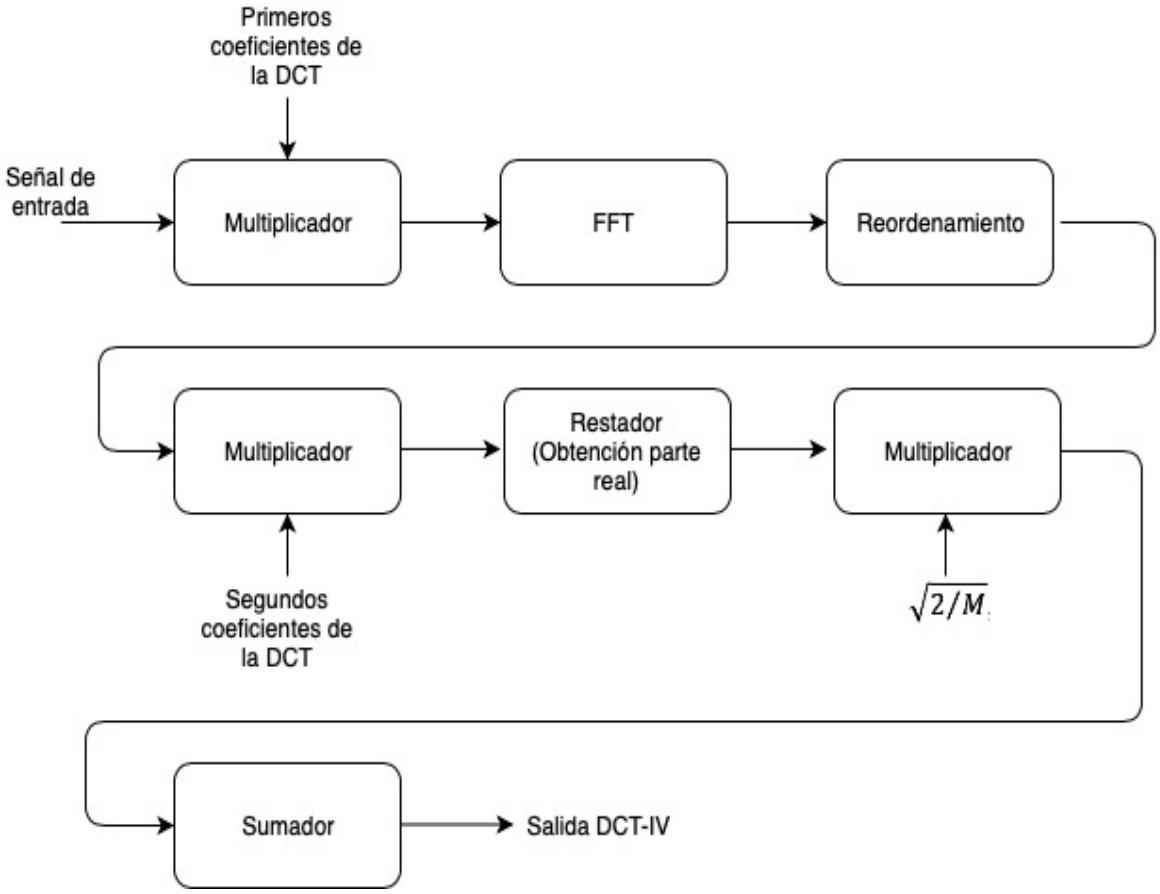

Figura 19. Etapas correspondientes al diseño en VHDL de la DCT.

Los pasos que seguir para la realización del modelo son los siguientes:

- Una vez identificadas las distintas etapas en las que se divide, se comenzará con el diseño de la primera de ellas, los multiplicadores.

- Una vez realizado el diseño, se pasará a simular su comportamiento y pudiendo ver si se comporta de la manera adecuada. Para ello se podrá hacer uso del

modelo en coma fija realizado en Matlab©, viendo si lo obtenido al simular en Vivado se corresponde a la señal cuantificada en nuestro modelo en coma fija.

- Una vez visto su correcto funcionamiento, se pasará al diseño de la siguiente etapa correspondiente y repitiendo el proceso de verificación de la señal simulada que en el punto anterior.

- Una vez realizado el modelo completo y verificado su funcionamiento gracias a las simulaciones, se pasará a realizar una simulación temporal del sistema completo.

Una vez definido los pasos que se deben seguir, pasamos a hacer una explicación previa de la entidad definida y del bloque multiplicador usado. Una vez explicado esto se pasará a realizar el estudio de cada una de las etapas mencionadas, para el caso de los siguientes estudios se van a particularizar la arquitectura para un número de puntos M=128, sin que esto suponga ninguna pérdida de generalidad, pues las conclusiones pueden extrapolarse fácilmente a otras dimensiones.

4.1. Entidad de la DCT

Antes de empezar con el diseño de las etapas definidas en el punto anterior, se deben definir los puertos de entrada y de salida que va a presentar nuestra entidad. Se debe tener en cuenta que los puertos de entrada solo se pueden leer, mientras que los de salida solo pueden ser escritos.

Para las entradas de la DCT se han definido:

- Un reloj, con el cual se va a mantener el sincronismo del sistema.

- Una señal de reset que permitirá la puesta en condiciones iniciales del diseño.

- Una señal de validación a la entrada, la cual indicará el momento en el que se debe empezar a coger valores de la señal de entrada.

- La señal de entrada que se pasará al módulo de la DCT.

La DCT presenta una salida únicamente real, por lo que no es necesario una salida para la parte real y la imaginaria. También presenta una señal de validación, la cual se activará una vez que la DCT se haya completado y se pueda empezar a tomar muestras validas a su salida. En la figura 20 se puede ver el esquema de la entidad descrita.

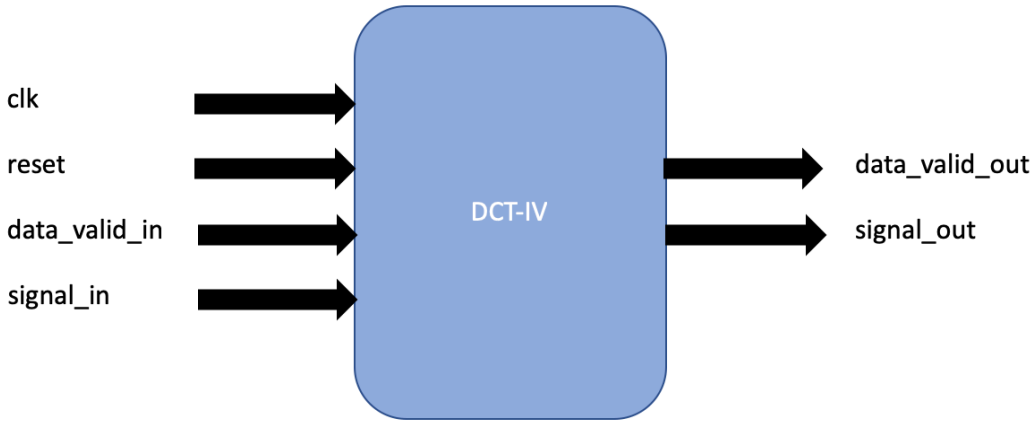

Figura 20. Entidad de la DCT-IV.

Se declararán genéricos, que deberán ser definidos a posteriori en el banco de pruebas, como el número de puntos o número de bits de la señal de entrada. Se hace uso de estos genéricos para una mayor flexibilidad en el diseño.

4.2. Estudio de la señal de entrada a la DCT

Esta señal de entrada vendrá proporcionada por Matlab©. La señal de entrada que definimos en el modelo haciendo uso de la representación en coma fija, podrá ser guardada en un archivo el cual se podrá leer en el banco de pruebas de nuestro modelo en VHDL. Para poder llevarlo a VHDL no puede haber parte fraccionaria, luego se debe convertir a un valor entero. Para obtener ese valor se debe hacer un desplazamiento de la coma hacia la derecha tantas veces como números de bits tenga la parte fraccionaria, esto se consigue multiplicando cada valor por $2^{nº\ bits\ parte\ frac.}$.

Teniendo así la misma entrada en el modelo en coma fija y en nuestro modelo VHDL, lo que facilita la validación de los resultados. Al tener los dos modelos una representación en coma fija, las señales intermedias de ambos modelos deberían coincidir y la señal salida deberían coincidir.

Para realizar la comprobación de que la señal de entrada está siendo leída correctamente en nuestro diseño, se pasará la señal de entrada directamente a la salida de nuestro diseño y se procederá a guardar esa señal de salida en un archivo de, para su posterior representación en Matlab©, pudiendo ver así si ambas señales son idénticas.

Una vez se tiene la señal de salida de Vivado en Matlab© se debe convertir ahora ese valor entero en fraccionario, para poder así tener la misma escala. Para ello se debe multiplicar cada valor de la señal obtenida por $2^{-nº\ bits\ parte\ frac.}$, siendo el número de bits de la parte fraccionaria de la señal de entrada 11, viniendo definido en la tabla 1.

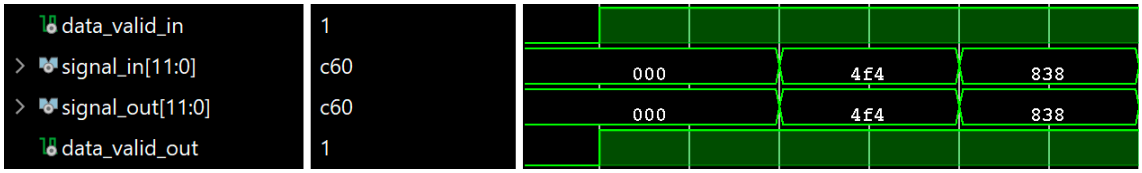

Figura 21. Simulación obteniendo a la salida la señal de entrada par M=128.

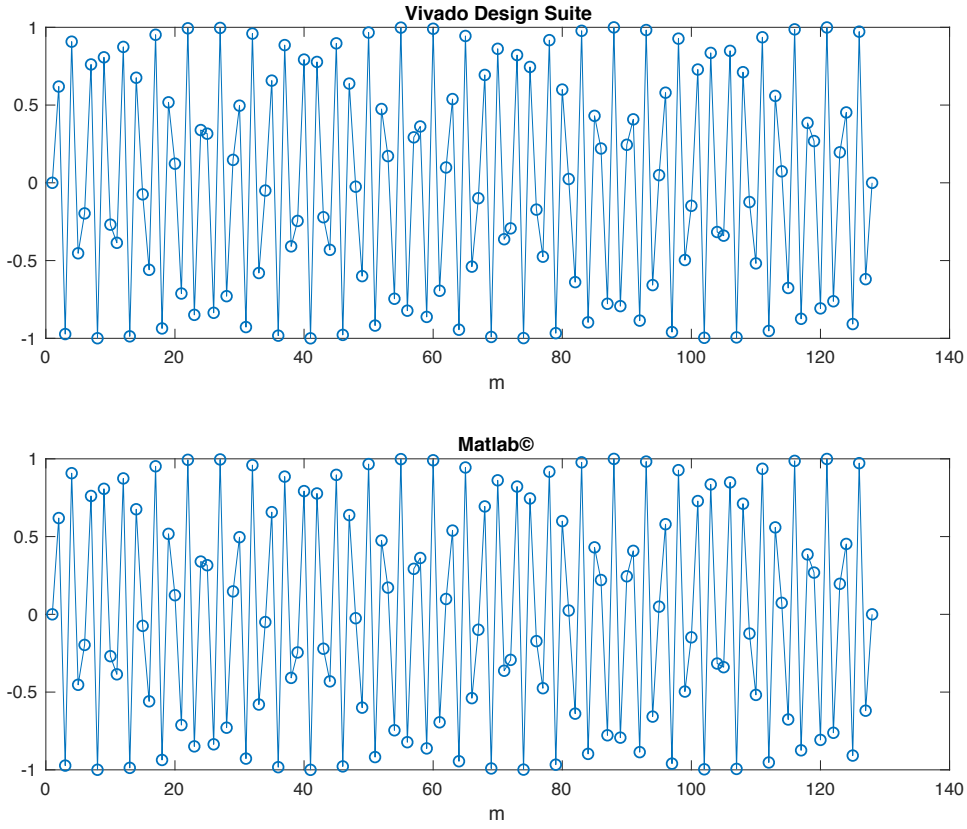

Figura 22. Comprobación de la correcta lectura de la señal de entrada al sistema.

4.3. Estudio de la primera etapa de la DCT (Multiplicador)

Para el desarrollo de esta etapa es necesario la generación del multiplicador ya representado en la figura 18. Para ello se hará uso del IP Catalog, la construcción del mismo se hará haciendo uso de los Mults y pudiendo así disminuir el uso de recursos.

Una vez que se tiene la señal de entrada y el multiplicador generado, solo faltaría tener los coeficientes que se le pasarán como entradas al bloque multiplicador. Para esta etapa son necesarios los primeros coeficientes que han sido cuantificados gracias a Matlab©, para poder ello se hace uso de la función 'num2bin' que permite convertir los valores de la señal cuantificada en una representación binaria de los mismos.

Finalmente, una vez que se tengan los coeficientes en el programa de la forma que se presenta en las figuras 23 y 24, se deben instanciar tantos multiplicadores como número de puntos tenga la DCT, tanto para la parte real como la imaginaria. Esta instanciación se encuentra descrita en la figura 25.

Los coeficientes se pasarán por la entrada de 18 bits del multiplicador, mientras que la señal de entrada se pasará por la de 25 bits. La entrada, como se comentó con anterioridad, viene de un archivo generado por Matlbab© y se extrae en Vivado dicha señal con 12 bits, por lo que se debe extender el de signo de la señal de entrada antes de poder ser pasada a la entrada del bloque multiplicador.

```
type coefs_array is array (0 to M-1) of STD_LOGIC_VECTOR(17 downto 0);
constant coefsr: coefs_array :=(
    "010000000000000000",   "001111111111111011",
    "001111111111101100",   "001111111111010011",
    "001111111110110001",   "001111111110000100",
    "001111111101001110",   "001111111100001110",
    "001111111011000100",   "001111111001110000",
    "001111111000010011",   "001111110110101011", …
```

Figura 23. Parte real de los primeros coeficientes de la DCT para M=128.

```
constant coefsi: coefs_array :=(
    "000000000000000000",   "111111110011011011",
    "111111100110110111",   "111111011010010011",
    "111111001101110000",   "111110000001001101",
    "111110110100101010",   "111110101000001001",
    "111110011011101000",   "111110001111001000",
    "111110000010101001",   "111101110110001100", …
```

Figura 24. Parte imaginaria de los primeros coeficientes de la DCT para M=128.

```
GEN_MULT_r_i: for i in 0 to M-1 generate
  Mult_r : Mult18x25
  PORT MAP (
    CLK => CLK,
    A => coefsr(i),
    B => in_ext(i),
    P => mult_out_r(i));
  Mult_i : Mult18x25
  PORT MAP (
    CLK => CLK,
    A => coefsi(i),
    B => in_ext(i),
    P => mult_out_i(i));
 end generate;
```

Figura 25. Instanciación de los multiplicadores.

Una vez hecho esto se pasará a simular el funcionamiento de esta primera etapa descrita. Para ello se necesitan los valores de la figura 21, los tres primeros valores de la señal de entrada son 0x000 0x4f4 y 0x838, que en decimal tienen un valor de 0, 1268 y -1992. También se debe obtener el valor en decimal de las tres primeras muestras de los primeros coeficientes de la DCT, estos valores se encuentran en las tablas 2 y 3.

Número binario	Número decimal
"0100000000000000”	65536
"001111111111111011"	65531
"001111111111101100"	65516

Tabla 2. Valor decimal de la parte real de los primeros coeficientes.

Número binario	Número decimal
"0000000000000000"	0
"111111110011011011"	-805
"111111100110110111"	-1609

Tabla 3. Valor decimal de la parte imaginaria de los primeros coeficientes.

Con estos valores se pueden hacer las primeras multiplicaciones y observar que los valores que se obtienen al multiplicar la señal de entrada por los coeficientes son correctos. Para poder observar que los 128 valores se están multiplicando correctamente, se representan las salidas de Vivado en Matlab© de la misma manera que se hizo con la señal de entrada.

El primer valor de la señal de entrada es cero, por lo que el primer valor de las salidas también son 0. La segunda muestra de la señal de entrada (1268) multiplicada por 65531 y por -805, da como resultado 83093308 y -1020740. La tercera muestra de la señal de entrada () multiplicada por 65516 y -1609, da como resultado -130507872 y 3205128. Como se puede observar en la figura 26, los resultados de las primeras multiplicaciones son correctas.

Se presenta un retardo en la salida de dos ciclos de reloj, debido a la lectura de la señal de entrada y al retardo que introduce la realización de la multiplicación.

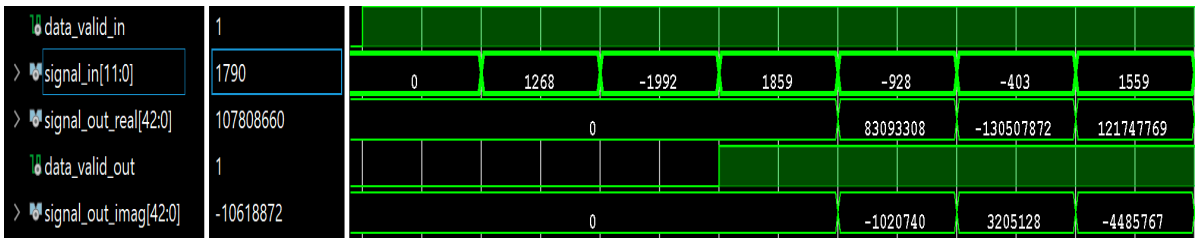

Figura 26. Salida obtenida al multiplicar lo señal de entrada con los coeficientes.

Como se puede ver en la figura 27, la señal que se obtiene en Matlab©, tras las multiplicaciones, es idéntica a la obtenida en Vivado. Se debe tener en cuenta a la hora de representar la señal que todos sus valores son enteros, por lo que se debe multiplicar por $2^{-nº\ bits\ parte\ frac.}$ para recuperar su parte fraccionaria. El número de bits de la parte fraccionaria viene definido en la tabla 1, correspondiendo un valor de 13 bits.

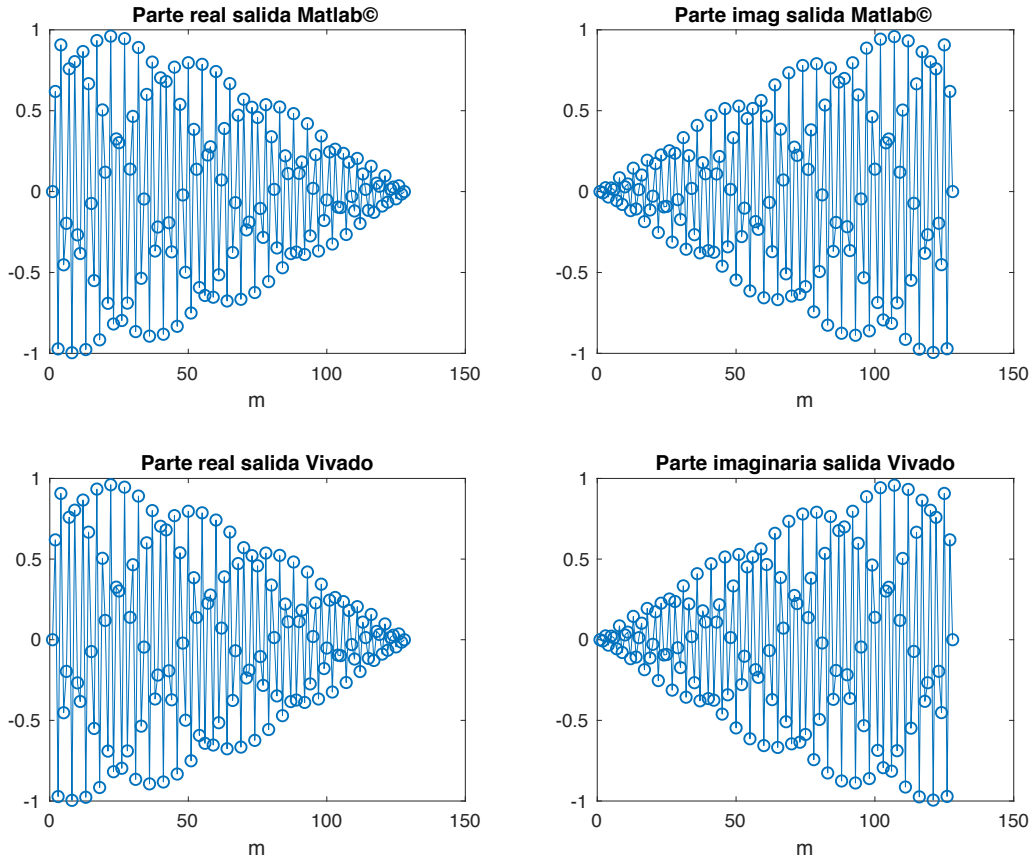

Figura 27. Representación de la señal obtenida al multiplicar la señal de entrada con los coeficientes.

4.4. Estudio de la segunda etapa de la DCT (FFT)

Para el desarrollo de esta etapa, se hace uso de Spiral DFT/FFT IP Generator. Spiral es un sistema de generación de programas para transformaciones lineales y otras funciones matemáticas. Spiral fue desarrollado gracias a la financiación de empresas estadounidenses como DAMPA, Intel, Nvidia y más.

El generador de la FFT permite seleccionar tanto los parámetros de nuestra FFT, como ciertos parámetros de la implementación. Los parámetros seleccionados para la generación de nuestra FFT se muestran en la figura 28.

Una vez completadas las especificaciones se generará un archivo Verilog, el cual se debe incluir en el proyecto de la DCT. Ya en el proyecto, se debe instanciar dicha FFT de manera que a sus entradas se conecten las salidas de la primera etapa. Los multiplicadores de la primera etapa tienen una salida de 43 bits, por lo que deben permanecer los 18 bits más significativos para poder pasárselo así a las entradas de la FFT.

parameter	value	range	explanation
Problem specification			
transform size	128	4–32768	Number of samples (?)
direction	forward		forward or inverse DFT (?)
data type	fixed point		fixed or floating point (?)
	18 bits	4–32 bits	fixed point precision (?)
	scaled		scaling mode (?)
Parameters controlling implementation			
architecture	fully streaming		iterative or fully streaming (?)
radix	2	2, 4, 8, 16, 32, 64, 128	size of DFT basic block (?)
streaming width	128	2–128	number of complex words per cycle (?)
data ordering	natural in / natural out		natural or digit-reversed data order (?)
BRAM budget	1000		maximum # of BRAMs to utilize (-1 for no limit) (?)
Permutation method	DATE'09 [4] (patent free)		Please click for more information
Generate Verilog Restaurar			

Figura 28. Especificaciones de la FFT a generar.

A continuación, se tratará de simular el funcionamiento del sistema hasta el momento, llevando la salida de la FFT a un archivo para su posterior representación en Matlab©. Hay que tener en cuenta que la señal de salida presentará un retardo añadido al de la primera etapa, este retardo es debido a la FFT. La FFT presenta un retardo de 31 ciclos de reloj, pero se debe tener en cuenta que para que la FFT opere correctamente necesita tener las 128 muestras a sus entradas. Luego el retardo total presente en esta etapa es de 159 ciclos de reloj.

De la misma manera que en etapas anteriores se debe recuperar la parte fraccionaria de la señal, debemos multiplicar la señal por $2^{-nº\ bits\ parte\ frac.}$, teniendo un valor el número de bits de la parte fraccionaria de 9. Esto se ve representado en la figura 29.

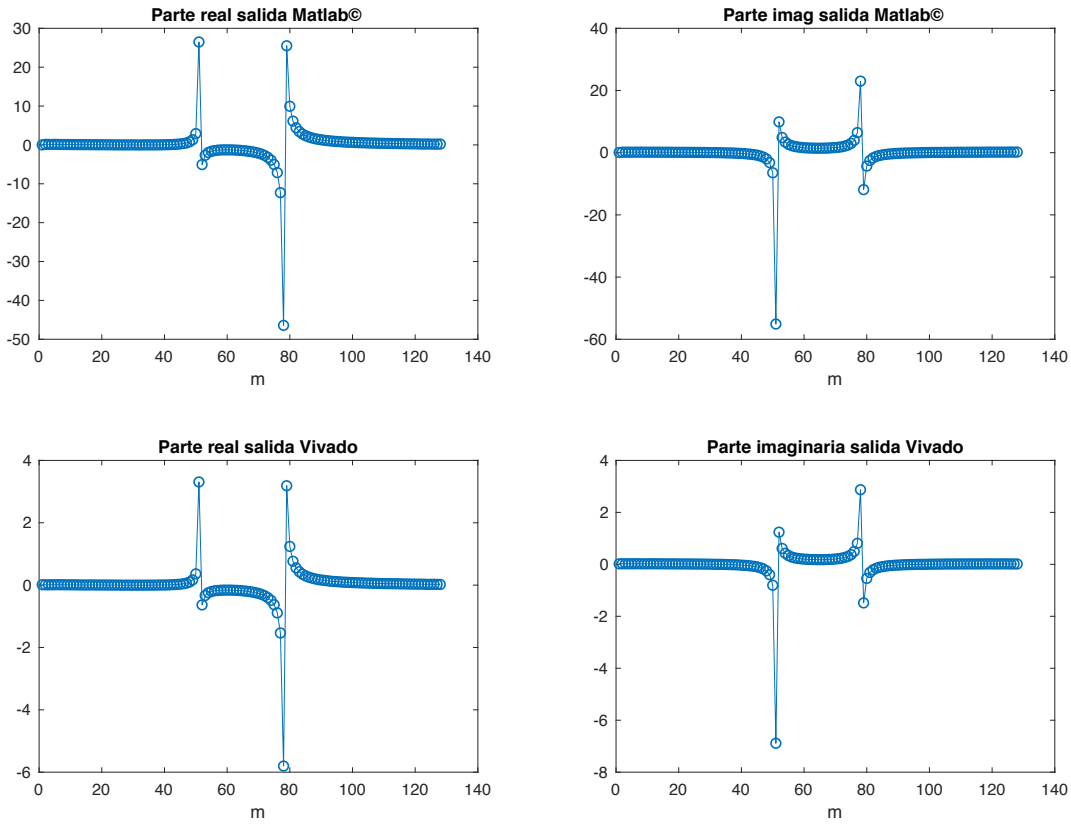

Figura 29. Representación de la salida de la FFT.

Como se puede apreciar la salida de la FFT implementada se presenta atenuada, esto es debido al uso del modo escalado presente a la hora de definir los parámetros de nuestra FFT. Los datos se presentan reducidos, debido a esta etapa, por un factor de 8 (3 bits).

4.5. Creación de una nueva entidad

Debido al factor de escala presente en la etapa anterior, se creará una nueva entidad compuesta por las etapas restantes (reordenamiento, multiplicador, restador, multiplicador y sumador). Esto se hará con el fin de poder validar la señal de salida por medio de Matlab©, para poder hacer eso la señal de entrada a los bloques debe ser idénticas en ambos modelos. Para ello se debe guardar la señal de salida de la FFT de Matlab© en un archivo, el cual se leerá en el nuevo banco de pruebas que se usará para la simulación de esta nueva entidad.

La nueva entidad, a diferencia de la entidad propia de la DCT, presenta una entrada para la parte real y otra para la parte imaginaria. Una vez validado las etapas restantes, se implementará esta nueva entidad dentro de la entidad de la DCT, formando así el sistema completo. Esta entidad se encuentra representada, a modo de esquema, en la figura 30. Tanto las entradas como la salida de esta entidad vienen definidas por 18 bits.

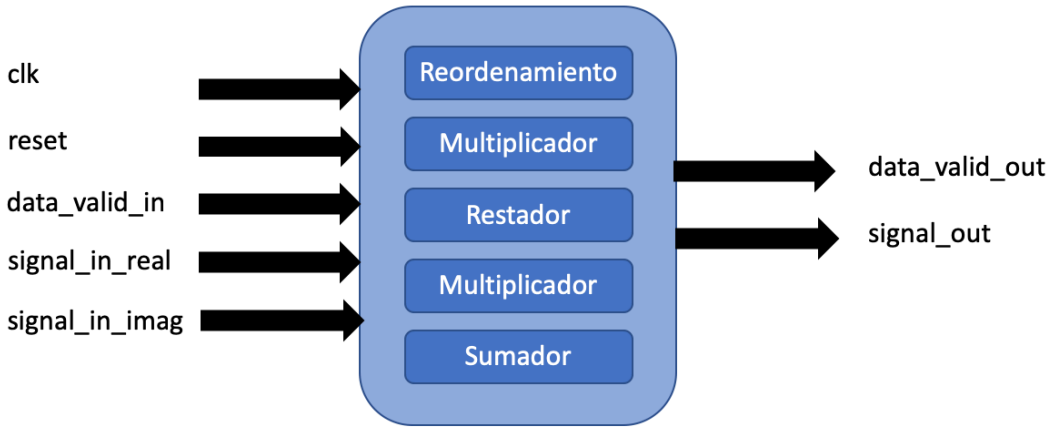

Figura 30. Nueva entidad con las etapas restantes.

4.5.1. Estudio de la tercera etapa de la DCT (Reordenamiento)

Esta etapa representa el algoritmo (4). Se trata de esquematizar el funcionamiento de este reordenamiento en la figura 31, indicando la necesidad de hacer el conjugado de la muestra reordenada con el color morado. El desarrollo del algoritmo en nuestro diseño se encuentra en la figura 32.

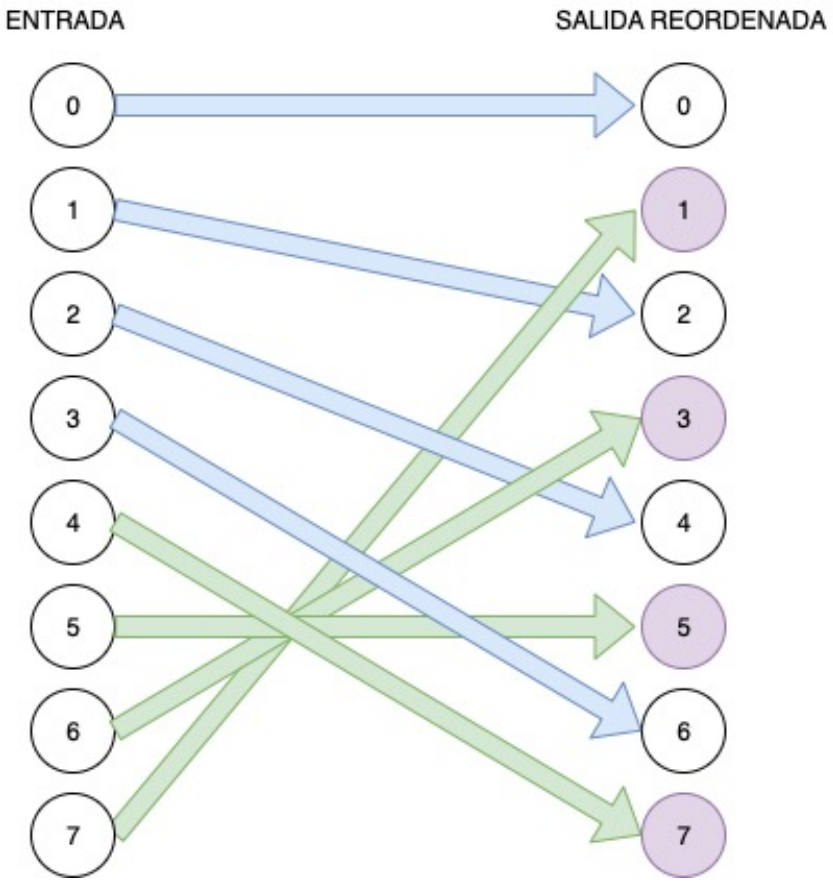

Figura 31. Esquema del algoritmo de reordenación.

```
ordena_real: for i in 0 to 64-1 generate -- Limite superior M/2
        signal_r(i+i) <= signal_in_real_ext(i);
        signal_r(i+i+1) <= signal_in_real_ext(M-1-i);
end generate ordena_real;

ordena_imag: for i in 0 to 64-1 generate -- Limite superior M/2
        signal_i(i+i) <= signal_in_imag_ext(i);
        signal_i(i+i+1) <= ( signal_in_imag_ext(M-1-i) xor
                                        "11111111111111111111111111" ) + 1;
end generate ordena_imag;
```

Figura 32. Algoritmo de ordenamiento de la señal de salida de la FFT.

Esta etapa presenta un gran retardo, debido a la necesidad de tener todas las muestras de la señal de salida de la FFT, a la entrada. Se decide extender el signo de las señales de entrada antes de realizar el reordenamiento, debido a que las señales reordenadas van a ser las entradas de la siguiente etapa, la cual hace uso de bloques multiplicadores. Debido a ello se extiende el signo de la señal de entrada pasando de 18 a 25 bits.

Se presentan dos capturas de la simulación realizada de esta etapa (figura 33 y 34), en la primera aparecen las primeras muestras de la señal de entrada y en la segunda las muestras finales de las señales de entrada junto a las salidas reordenadas.

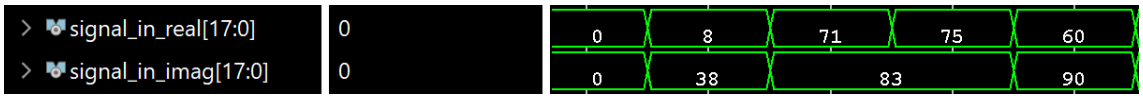

Figura 33. Primeras muestras de la señal de salida de la FFT.

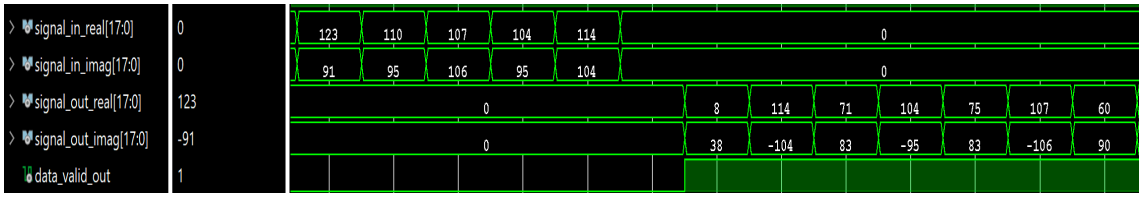

Figura 34. Últimas muestras de la salida de la FFT y señales reordenadas obtenidas en la simulación.

Una vez tenido en cuenta el retardo necesario para la correcta captura de la señal de salida, se procede a guardar dicha señal para su análisis en Matlab©. Una vez en Matlab, se debe recuperar la parte fraccionando multiplicando la señal por 2^{-9}. La parte fraccionaria esta codificada con 9 bits, de igual manera que la salida de la FFT. Los resultados obtenidos se muestran en la figura 33.

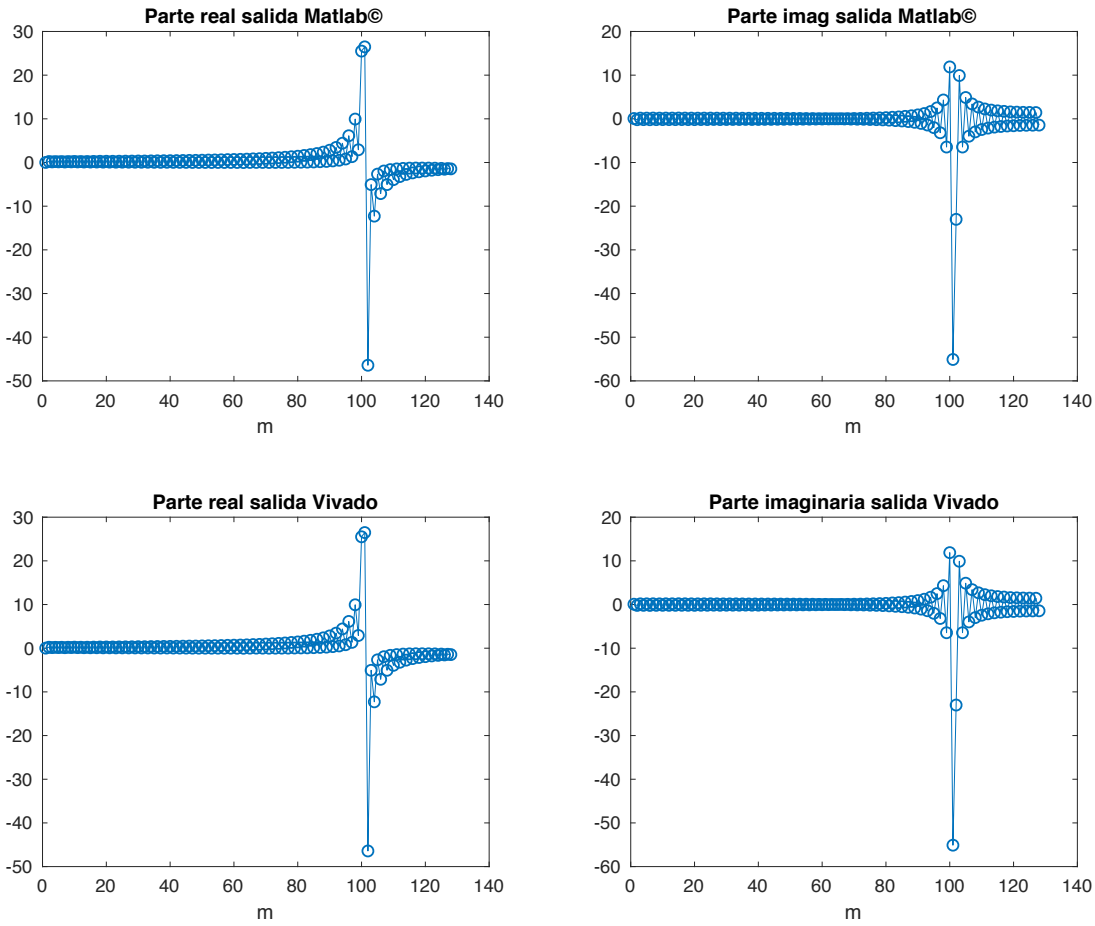

Figura 35. Señal obtenida al reordenar la salida de la FFT.

4.5.2. Estudio de la cuarta y quinta etapa de la DCT (Multiplicador y restador)

Esta etapa se encarga de multiplicar la señal reordenada con los segundos coeficientes. Una vez se tieen la señal reordenada y con el multiplicador anteriormente generado, solo faltaría tener los coeficientes que se le pasaran como entradas al bloque multiplicador. Serán son necesarios los segundos coeficientes que han sido cuantificados gracias a Matlab©, para poder ello se hace uso de la función 'num2bin' que permite convertir los valores de la señal cuantificada en una representación binaria de los mismos.

Finalmente, una vez que se tengan los coeficientes en el programa de la forma que se presenta en la figura 36 y 37.

```
type coefs_array is array (0 to M-1) of STD_LOGIC_VECTOR(17 downto 0);
constant coefs_real2: coefs_array :=(
      "011111111111111101",   "011111111111101001",
      "011111111111000010",   "011111111110000111",
      "011111111100111000",   "011111111011010101",
      "011111111001011111",   "011111110111010101",
      "011111110100110111",   "011111110010000110",
      "011111101111000001",   "011111101011101000", …
```

Figura 36. Parte real de los segundos coeficientes de la DCT para M=128.

```
constant coefs_imag2: coefs_array :=(
      "111111110011011011",   "111111011010010011",
      "111111000001001011",   "111110101000000011",
      "111110001110111101",   "111101110101110111",
      "111101011100110011",   "111101000011110001",
      "111100101010110000",   "111100010001110001",
      "111011111000110101",   "111011011111111011",…
```

Figura 37. Parte imaginaria de los segundos coeficientes de la DCT para M=128.

Al tener ambas señales valores complejos, la multiplicación de ambos no es directa. La multiplicación de dos números complejos se define en (8). Se representa en la figura 38 un esquema del modelo equivalente a la expresión mencionada.

$$(a + bi) \cdot (c + di) = (ac - bd) + (\, ad + bc) \qquad (8)$$

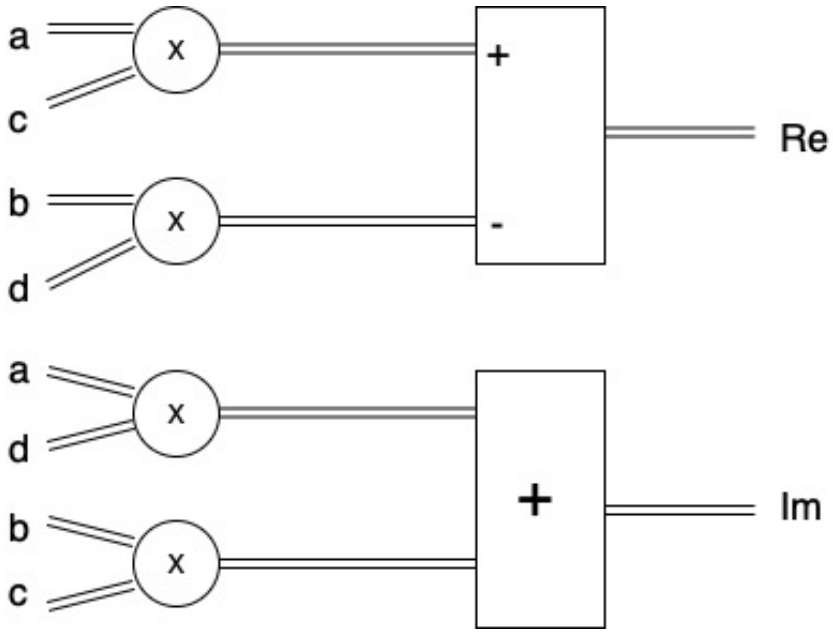

Figura 38. Esquema multiplicación de números complejos.

Al analizar (6), lo único que sería necesario para el diseño descrito sería la parte superior del esquema de la figura 38, por lo que terminarías multiplicando la parte real de la señal reordenada con la parte real de los segundos coeficientes y la parte imaginaria de la señal reordenada con la parte imaginaria de los segundos coeficientes. Al restar los valores obtenidos de los multiplicadores, se obtiene la parte real de la señal de salida, tras la multiplicación de las dos señales complejas.

Los coeficientes se pasarán por la entrada de 18 bits del multiplicador, mientras que la señal reordenada se pasará por la de 25 bits. Se instanciará tanto multiplicadores como número de puntos tenga nuestra señal (*M*=128), tanto para la multiplicación de los

valores reales como para la de los valores imaginarios. Esta instanciación y la obtención de la parte real de nuestra señal de salida se encuentra descrita en la figura 39. Hay que tener en cuenta que a la hora de restar se debe extender el bit de signo para evitar la pérdida de datos.

```
GEN_MULT: for i in 0 to M-1 generate
   Mult_r_r : Mult18x25
   PORT MAP (
     CLK => CLK,
     A => coefs_real2(i),
     B => signal_r(i),
     P => out_aux_a_c(i));
 end generate;

 GEN_MULT2: for i in 0 to M-1 generate
   Mult_i_i : Mult18x25
   PORT MAP (
     CLK => CLK,
     A => coefs_imag2(i),
     B => signal_i(i),
     P => out_aux_b_d(i));
 end generate;

 resta_parte_real: for i in 0 to M-1 generate
    real_a(i) <= (out_aux_a_c(i)(42) & out_aux_a_c(i)) - (out_aux_b_d(i)(42) &
                                                          out_aux_b_d(i));
 end generate resta_parte_real;
```

Figura 39. Instanciación de los multiplicadores y obtención de la parte real de la señal de salida.

En esta etapa se presenta un retardo en la salida de un ciclo de reloj, debido a la realización de la multiplicación. Como se puede ver en la figura 40, la señal que se obtiene en Matlab©, tras la simulación, es idéntica a la obtenida en Vivado. Se debe tener en cuenta a la hora de representar la señal que todos sus valores son enteros, por lo que se debe multiplicar por $2^{-nº\ bits\ parte\ frac.}$ para recuperar su parte fraccionaria. El número de bits de la parte fraccionaria viene definido en la tabla 1, correspondiendo un valor de 13 bits.

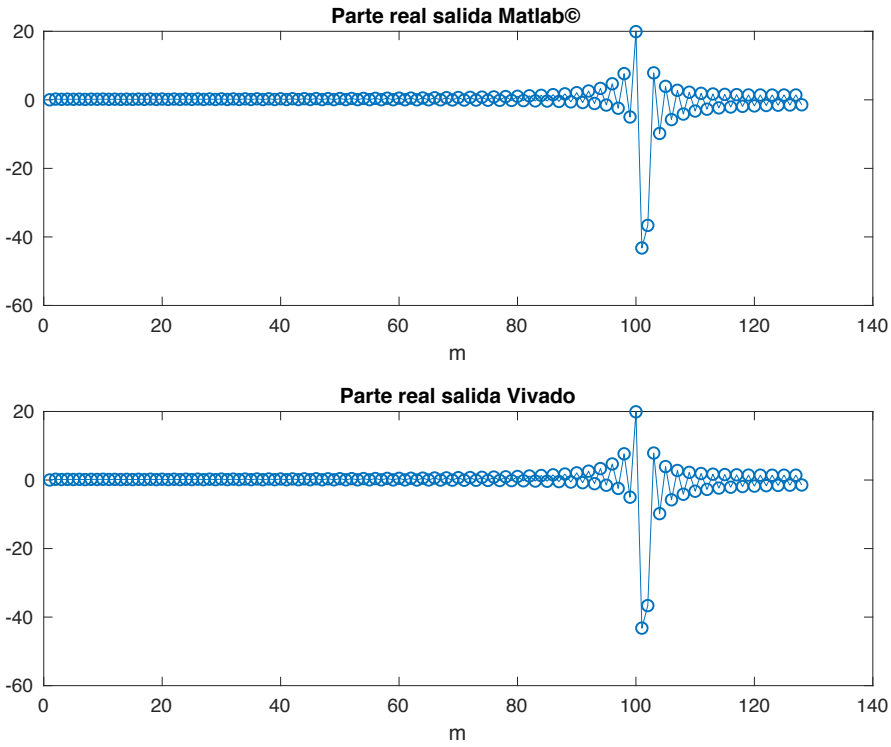

Figura 40. Parte real de la señal de salida, tras multiplicar la señal reordenada con los segundos coeficientes.

4.5.3. Estudio de la sexta etapa de la DCT (Multiplicador)

Esta etapa consiste en la multiplicación de la señal obtenida en la etapa anterior con la variable $\sqrt{2/\mathrm{M}}$. Para ello se hará uso del multiplicador generado con anterioridad, instanciándolo tantas veces como puntos tiene nuestra DCT (M=128).

Se procede a declarar con la instanciación mostrada en la figura 41, se debe realizar la declaración de la figura 42. Permanecerán los bits del 37 al 13 de la señal de salida de la etapa anterior, ya que son los que contienen toda la información del valor de las muestras.

```
GEN_MULT_RAIZ: for i in 0 to M-1 generate
  Mult_raiz : Mult18x25
  PORT MAP (
     CLK => CLK,
     A => raiz,
     B => real_a(i)(37 downto 13),
     P => signal_out_aux(i));
end generate;
```

Figura 41. Instanciación de los multiplicadores de la etapa seis.

```
signal raiz: std_logic_vector(17 downto 0):= "000000000100000000";
```

Figura 42. La constante $\sqrt{2/M}$ codificada.

Como se puede ver en la figura 43, la señal que se obtiene en Matlab©, tras la simulación, es idéntica a la obtenida en Vivado. Se debe tener en cuenta a la hora de representar la señal que todos sus valores son enteros, por lo que se debe multiplicar por 2^{-13} para recuperar su parte fraccionaria. Este número de bits de la parte fraccionaria viene definido en la tabla 1.

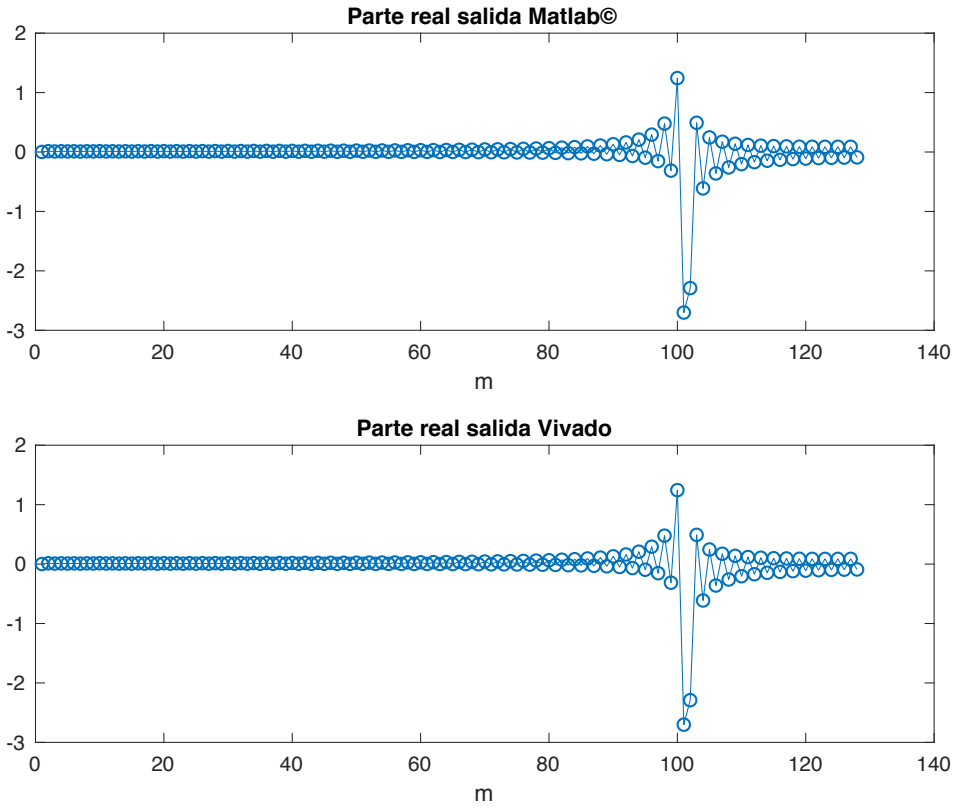

Figura 43. Señal de salida tras la multiplicación de la señal de la etapa anterior con $\sqrt{2/M}$.

Esta etapa presenta un retardo en la salida de un ciclo de reloj, debido a la realización de la multiplicación.

4.5.4. Estudio de la séptima etapa de la DCT (Sumador)

Para la obtención de la señal de salida de la DCT, lo único que faltaría es multiplicar por dos la señal obtenida de la etapa anterior o lo que es lo mismo sumarla consigo misma. Se presenta el código con dicha funcionalidad en la figura 44.

```
pk: for i in 0 to M-1 generate
  signal_out_aux(i) <= ( b_q(i)(29 downto 12) + b_q(i)(29 downto 12) );
  end generate pk;
```

Figura 44. Obtención de la señal de salida de la DCT.

Como se puede ver en la figura 45, la señal que se obtiene en Matlab©, tras la simulación, es idéntica a la obtenida en Vivado. Se debe tener en cuenta a la hora de representar la señal que todos sus valores son enteros, por lo que se debe multiplicar por 2^{-13} para recuperar su parte fraccionaria. Este número de bits de la parte fraccionaria viene definido en la tabla 1.

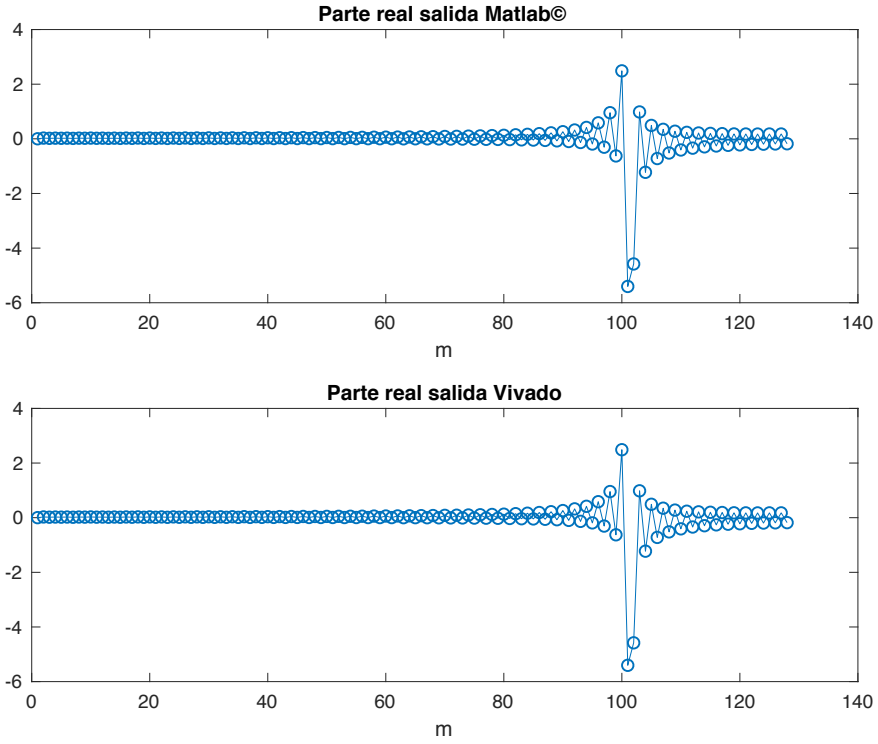

Figura 45. Estudio de la señal de salida de la DCT del módulo.

4.6. Creación de la DCT completa

Una vez creado un segundo módulo con las etapas de la tres a la siete y probado su correcto funcionamiento, se pasará a su implementación en la DCT. Para el correcto funcionamiento del sistema completo, es imprescindible saber en qué momento se produce la señal de salida de la FFT. Este instante es de gran relevancia, ya que es el momento en el cual se debe poner en marcha el segundo módulo que se creó con las etapas restantes, por lo que la validación de la señal de entrada del segundo módulo se debe activar tras 161 ciclos de reloj (1 de lectura de la señal de entrada, 1 de la multiplicación y 161 de la FFT). Los datos de los retardos obtenidos se encuentran en la tabla 4.

ETAPA	RETARDO
Multiplicador	1 ciclo
FFT	128 + 31 ciclos
Reordenamiento	128 ciclos
Multiplicador	1 ciclo
Restador	-
Multiplicador	1 ciclo
Sumador	-

Tabla 4. Valores de los retardos por etapas.

Se tiene que tener en cuenta, que las muestras de la señal de entrada se deben pasar al segundo módulo a través de un multiplexor. Gracias a este multiplexor y un contador es posible pasarle estas muestras una a una, como se muestra en la figura 46.

```
process (clk, rst)
variable count: integer range 0 to 127;
begin
if rst = '1' then
  count:= 0;
  fft_out_aux_r <= ( others =>'0' );
  fft_out_aux_i <= ( others => '0' );
elsif clk'event and clk = '1'  then
  if data_valid_fft_out = '1' then
      fft_out_aux_r <= fft_out_r(count);
      fft_out_aux_i <= fft_out_i(count);
    if count < 127 then
      count:= count + 1;
    end if;
  end if;
end if;
end process;
```

Figura 46. Generación de un multiplexor para el control de la entrada al segundo módulo.

Una vez instanciado este bloque, y sabiendo de manera correcta el retardo que presenta a la hora de tener la señal de salida, la señal de la salida de la DCT de nuestro sistema

completo debe ser igual al generado por Matlab©, pero con factor de reducción de 8 (3 bits). Para ello se procede a la simulación del sistema completo, guardando los valores en un archivo para su posterior representación en Matlab©. Antes de proceder a su representación, se debe recuperar la parte fraccionaria multiplicando por 2^{-13}.

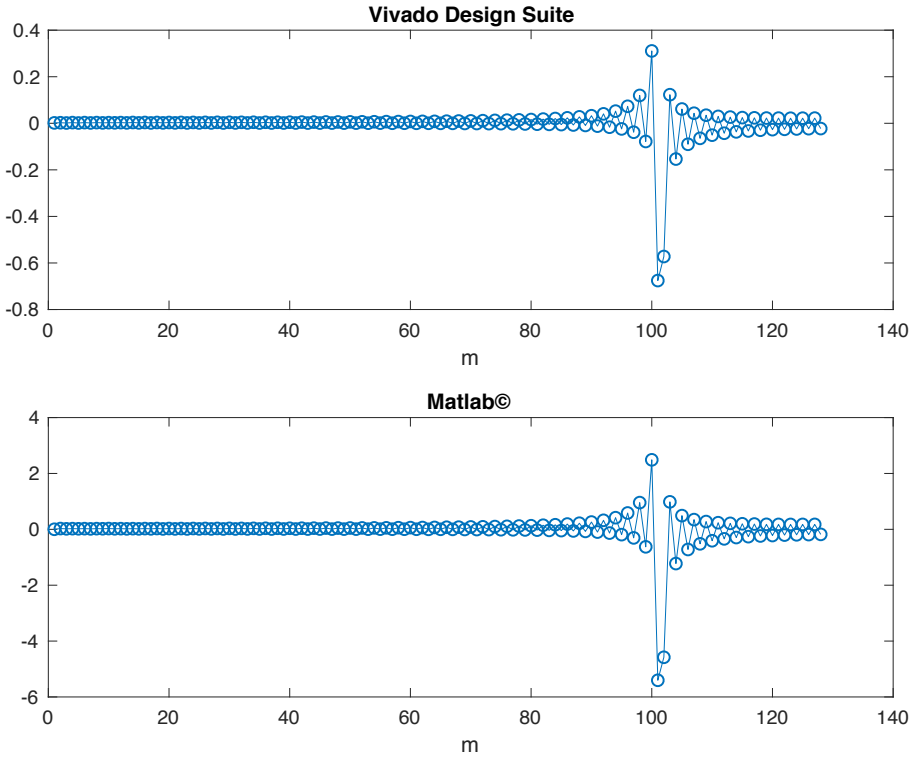

Figura 47. Señal de salida de la DCT del sistema completo.

4.7. Recursos necesarios para una DCT de *M*=128

A través de la síntesis, es posible obtener el número de recursos del que hace uso la arquitectura propuesta. Al tratarse de un número de puntos alto, el número de DSPs utilizados es considerable.

Bloque	Slice LUTs	Slice Registers	F7 Muxes	F8 Muxes	DSPs	Bonded IOB	BUFGCTRL
Global	81218	82437	1105	520	1380	53	1
Reordenamiento, multiplicador, restador, multiplicador, sumador	7469	4190	289	136	384	0	0
FFT	69490	76394	816	384	740	0	0
Multiplicador	0	0	0	0	256	0	0

Tabla 5. Utilización de recursos del sistema.

Capítulo 5: Conclusiones y Trabajos Futuros

En este apartado se analiza el trabajo realizado y se comentarán los resultados obtenidos del mismo. Se tratarán posibles problemas mencionados en los apartados, tratando de obtener conclusiones sobre nuestro diseño. En la segunda parte se tratará de explicar posibles mejoras que se podrían aplicar al diseño de la DCT, así como la posibilidad de validar el diseño con un número de puntos diferente.

5.1. Conclusiones

Este trabajo buscaba el diseño eficiente de una arquitectura para la DCT mediante VHDL. Para ello se debió de realizar un estudio previo sobre el funcionamiento propio de la DCT y más concretamente de la DCT-IV. La realización de la DCT es posible con múltiples algoritmos, por lo que se trató de elegir el algoritmo más idóneo para su adaptación al lenguaje hardware. Además, se trató de buscar una configuración que nos permitiera desarrollar una arquitectura lo más flexible posible.

Una vez se obtuvo este algoritmo, se desarrolló dicho modelo en coma flotante en Matlab©. De manera que se obtiene la base del modelo en coma fija. Una vez se obtuvo el modelo en coma fija, se podían diferenciar las diferentes etapas de las que constaría la arquitectura. Se realizó un estudio previo para determinar el número de bits idóneo para las diferentes señales del sistema.

Una vez definidas las etapas y la distribución de bits a lo largo de las mismas, se trató de replicar ese modelo en coma fija en nuestra arquitectura de VHDL. Para ello, se generó un elemento muy importante para la realización de este desarrollo, un multiplicador, que facilita en gran medida todo el desarrollo.

Se decidió, que la mejor forma de realizar era el diseño, era la realización de etapa a etapa. De esta manera se puede simular el funcionamiento de cada etapa que se vaya incorporando al diseño. Esto se vio dificultado por el escalado que presentaba la FFT, por lo que se decidió seguir con el mismo mecanismo, pero para ello se debía crear una nueva entidad con unos estímulos controlados para su correcta validación.

Para poder validar cada etapa del diseño, fue de suma importancia los modelos creados con anterioridad en Matlab©, ya que las salidas de cada etapa ya se analizaron con posterioridad en el modelo de coma fija. Esto facilita el trabajo, ya que permite analizar tu señal de salida sin necesidad de ir visualizando cada una de las muestras.

5.1. Trabajos Futuros

En este punto se tratarán ciertos puntos con relación a la mejora o correcciones derivadas del Diseño de una arquitectura eficiente para la Transformada Discreta del Coseno (DCT) en un dispositivo FPGA o analizar posibles líneas de trabajo.

- Sería interesante la comparación del número de recursos utilizados al realizar la DCT con distinto número de puntos.

- Implementación de una FFT más flexible. Tras llevar a cabo su instanciación, se vio el tiempo tan alto que lleva la misma, ya que se deben ir variando sus parámetros manualmente, lo que, para un diseño con número de puntos muy elevado, supondría muchísimo tiempo. Esto es de igual manera aplicable a cambios en la selección de su configuración, debes volver a generare el archivo e instanciarlo de nuevo a mano.

- Tratar de incorporar la arquitectura diseñada en un sistema de comunicaciones.

- Un punto interesante sería la definición de la arquitectura de la DCT mediante herramientas High-Level Synthesis (HLS), permitiendo de esa manera establecer comparaciones.

Pliego de condiciones

Se detallan los elementos hardware y software utilizados que posibilitaron la realización del TFG.

Hardware

MacBook Pro

- Procesador: 2,3 GHz Intel Core i5
- Memoria: 8 GB 2133 MHz LPDDR3
- Gráficos: Intel Iris Plus Graphics 640 1536 MB

Software

Sistemas Operativos

- macOS Mojave 10.14
- Windows 10

Software usado en el desarrollo del proyecto

- Vivado 2018.3
- Matlab© R2017b
- Paquete Office

Presupuesto

En este apartado se tratarán los costes relativos al proyecto. En la tabla 6 se representa el coste debido al software y el hardware usado para el desarrollo del proyecto.

Material/Software	Precio	Duración/Utilización	Coste en el proyecto
MacBook Pro	1800 €	3 años/6 meses	300 €
Windows 10	40 €	6 meses	40 €
Vivado 2018.3	0 €	6 meses	0 €
Matlab© R2017b	35 €	6 meses	35 €
Paquete Office	70 €	6 meses	70 €

Tabla 6. Coste del material usado.

El coste total relativo al material y al software es de 445 €. A continuación, se indicarán los costes relativos a la mano de obra en la tabla número 7.

Trabajo	Número de horas	Coste la hora	Total
Ingeniero	310 h	60 €	18600€
Redacción	120 h	15 €	1800€

Tabla 7. Coste relativo a la mano de obra.

El coste total de mano de obra es de 20400 €.

El coste total del proyecto incluyendo el IVA se muestra en la tabla 8.

Coste	Precio sin IVA	IVA	Total
Material/Software	445 €	21%	538,45 €
Mano de obra	20400 €	21%	24684 €

Tabla 8. Costes con IVA.

El coste total del proyecto, IVA incluido, es de 25.222,45 €.

Bibliografía

[1] (2012, Febrero 24). DCT Type IV Functions [Online]. Available: https://www.cs.indiana.edu/~bhimebau/CMSIS/Documentation/DSP/html/group___d_c_t4___i_d_c_t4.html.

[2] Pablo Poudereux, Álvaro Hernández, Raúl Mateos, Freddy A.Pinto-Benel, Fernando Cruz-Roldán (Noviembre 2016). Design of a filter bank multi-carrier system for broadband power line communications. *Signal Processing, 128(*2016), 57-67.

[3] Ken Cabeen and Peter Gent. Image Compression and the Discrete Cosine Transform. [Online]. Available: https://www.math.cuhk.edu.hk/~lmlui/dct.pdf.

[4] Cristian Sisterna, MSc. FIELD PROGRMMABLE GATE ARRAYS (FPGAs) – Cartelera. [Online]. Available: https://carteleras.webcindario.com/FPGA-Cisterna.pdf.

[5] José Manuel Marín de la Rosa. FIELD PROGRMMABLE GATE ARRAY (FPGA). [Online]. Available: http://bibing.us.es/proyectos/abreproy/11375/fichero/MEMORIA%252FFPGAs.pdf.

[6] Genera Soluciones Tecnológicas, S.L. (2019). Aplicaciones de las FPGA. [Online]. Available: https://www.generatecnologias.es/aplicaciones_fpga.html

Universidad de Alcalá
Escuela Politécnica Superior