Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Electrónica

Implementación de la Transformada Discreta de Fourier IFFT/FFT en un sistema embebido para automatizar un sistema de comunicación OFDMA.

Tesis presentada como requisito para obtener el título de Maestría en Ciencias de la Electrónica.

Presenta: Lic. Francisco Mora Hernández.*

Asesora de tesis: Dra. Josefina Castañeda Camacho.

Puebla, Puebla diciembre de 2015.

*Becario CONACYT.

Resumen.

Este trabajo esta dividido en 7 capítulos. En el primer capítulo se proporciona un panorama de la importancia de los sistemas de comunicación basados en OFDMA, su importancia en el mundo actual y el porque de la relevancia de la FFT e IFFT en estos sistemas.

En el segundo capítulo se profundiza en la FFT partiendo del estudio del estado del arte, su importancia en la diferentes ramas de la ciencia, su gran relevancia cuando se habla de su implementación en sistemas digitales y su gran numero de aplicaciones.

En el tercer capítulo se tratan las bases matemáticas de la FFT, junto con las diversas técnicas para su obtención y los diferentes tipos de algoritmos FFT que existen.

En el cuarto capítulo se explica el método elegido para poder obtener las diferentes FFT´s de diverso numero de muestras que serán necesarias, junto con las técnicas escogidas para poder simular por medio de Matlab y comprobar su correcto funcionamiento.

En el quinto capítulo se trasladan las ideas obtenidas en el cuarto capítulo para poder realizar la implementación en FPGA de los bloques básicos y los bloques FFT de un menor numero de muestras.

En el sexto capítulo se muestra como con los bloques obtenidos en el quinto capítulo se puede realizar la FFT/IFFT de 128 muestras con el método descrito en el cuarto capítulo.

Finalmente en el capítulo 7 se muestran las pruebas realizadas a los diversos bloques de FFT/IFFT y del bloque final FFT/IFFT de 128 muestras a partir de varias funciones ya conocidas.

En la ultima parte se tratan las conclusiones obtenidas a partir de este trabajo de tesis, junto con ideas que son propuestas para la mejora y el uso en otras aplicaciones para posteriores trabajos.

Índice

Objetivo general.

Implementar la Transformada Discreta de Fourier IFFT/FFT en un sistema embebido para un sistema de comunicaciones OFDMA.

Objetivos particulares.

1. Estudiar el estado del arte del tópico, los principios de funcionamiento de la Transformada Discreta de Fourier IFFT/FFT en un sistema de acceso OFDMA y las normas que lo regulan.
2. Analizar las metodologías propuestas en la literatura para la ejecución de la Transformada Discreta Inversa y Directa de Fourier.
3. Simular en Matlab la Transformada Discreta de Fourier IFFT/FFT bajo alguna de las metodologías propuestas en la literatura.
4. Desarrollar el programa de la Transformada Discreta de Fourier IFFT/FFT en AHDL para su posterior implementación en el FPGA.
5. Realizar pruebas del algoritmo IFFT/FFT y comprobar su correcto funcionamiento simulando secuencias de datos. -

Introducción.

Las redes móviles son uno de los avances tecnológicos más importantes en la historia del ser humano. La capacidad de establecer una llamada a través de dos terminales sin ninguna conexión, aparentemente, entre ellos era inconcebible hace un tiempo.

Hoy en día la cantidad de usuarios de comunicaciones inalámbricas ha ido en aumento y no solo eso, sino la cantidad de transferencia de datos ha crecido drásticamente haciendo de gran importancia la renovación de los sistemas de comunicaciones para poder satisfacer dichas demandas [1]. La gente está cada vez más acostumbrada a comunicarse en cualquier lugar, en cualquier momento y de cualquier forma, este patrón de comunicación va acompañada de la creciente demanda de acceso inalámbrico de banda ancha móvil. A medida que aumenta la demanda, los proveedores de acceso a la red inalámbrica y los proveedores de servicios de red están implementando sistemas de última generación que pueden soportar el flujo de datos de alta velocidad [2].

Después de ya casi dos décadas de ininterrumpido crecimiento de las comunicaciones móviles (Figura A), primero de la mano de GSM (*groupe spécial mobile / sistema global para las comunicaciones móviles*) y últimamente con el despliegue definitivo UMTS (*Universal mobile telecommunications system / Sistema universal de telecomunicaciones móviles)*, estamos en los albores de una nueva generación de comunicaciones móviles conocida como 4G (*Fourth Generation / Cuarta Generación*) [3].

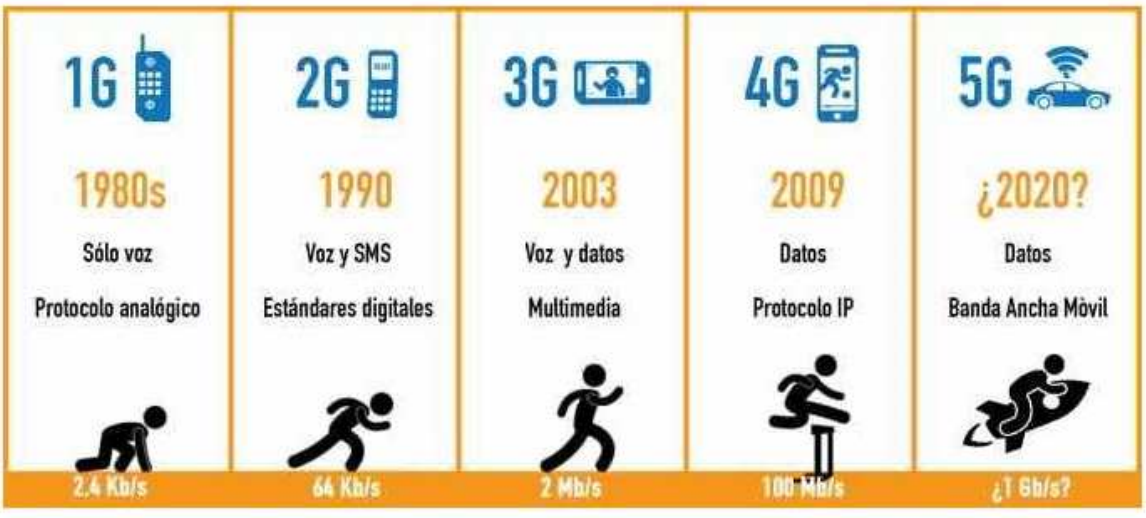

Figura A.- Evolución de las redes móviles.

La tecnología 4G está basada completamente en el protocolo IP (*Internet Protocol / Protocolo de Internet*), siendo un sistema de sistemas y una red de redes que se alcanza gracias a la convergencia entre las redes de cables e inalámbricas. Esta tecnología podrá ser usada por módems inalámbricos, smartphones y otros dispositivos móviles [3].

La principal diferencia con las generaciones predecesoras es la capacidad para proveer velocidades de acceso mayores de 100 Mbit/s en movimiento y 1 Gbit/s en reposo, manteniendo un QoS (*Quality of Service / Calidad de Servicio)* de punta a punta de alta seguridad que permitirá ofrecer servicios de cualquier clase en cualquier momento, en cualquier lugar, con el mínimo costo posible.

La tecnología 4G permite realizar transferencia de datos multimedia a alta velocidad, realizar video llamadas con imagen y sonido de alta calidad, ver televisión HD (*High Definition / Alta Definición)* y 3D (*Third Dimension / Tercera Dimensión*), efectuar videoconferencias en vivo y tener una mejor experiencia al jugar en red, entre otras ventajas [4].

Una de las tecnologías que más se ha impulsado para las comunicaciones 4G es OFDM (*Orthogonal Frequency-Division Multiplexing / Multiplexación por División de Frecuencias Ortogonales)* que permite la transmisión de grandes cantidades de datos digitales a través de una onda de radio, la cual ha tenido gran aceptación en Europa para la transmisión de televisión y radio digital pero a nivel mundial la atención hacia OFDM y la versión multiusuario de esta OFDMA (*Orthogonal Frequency-Division Multiple Access / Acceso Múltiple por División de Frecuencias Ortogonales*) ha ido en aumento, haciendo de estos claramente la tendencia a seguir para los próximos años [2].

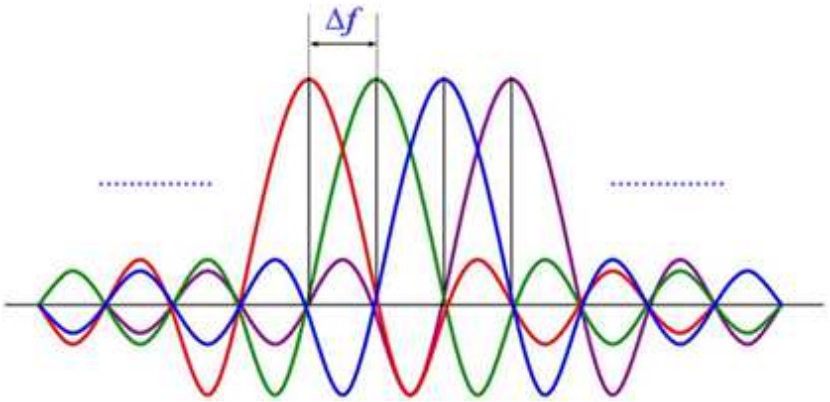

Figura B.- Representación en frecuencia de 4 subportadoras ortogonales entre sí.

La técnica de transmisión de OFDM es un mecanismo de trasmisión de múltiples portadoras que consiste en asignar un conjunto de símbolos sobre un conjunto de subportadoras. Estas soportadoras tienen una característica particular y es la de ser ortogonales entre sí como se muestra en la Figura B. Esto permite la trasmisión simultánea de todos los símbolos manteniendo la capacidad de separación de los mismos en la recepción.

Esta tecnología es conocida desde los años 60's pero su aplicación en las comunicaciones inalámbricas es mucho más reciente ya que hasta ahora se ha podido hacer frente a la complejidad de la implementación de los sistemas de transmisión y recepción [5].

Aunque existen varios metodos para poder implementar este tipo de sistemas los de mayor relevancia son OFDM, OFDMA, SOFMDA (*Scalable Orthogonal Frequency Division Multiple Access / Acceso Múltiple por División de Frecuencias Ortogonales Escalable*) y Flash-OFDM (*Fast Low Latency Access with Seamless Handoff – OFDM / Acceso Rápido de Baja Latencia con Traspaso sin Fisuras*). Aunque estructuralmente tienen ciertas diferencias existe una gran similitud entre todas y es la utilización de la FFT (*Fast Fourier Transform / Transformada Rápida de Fourier*) y la IFFT (Inverse Fast Fourier Transform / *Transformada Rápida de Fourier Inversa*) para la demodulación y modulación de la señal a transmitir, la utilización de las versiones digitales de la transformada de Fourier fue lo que logro darle viabilidad a este tipo de tecnología [4].

Además otro factor que logró darle gran impulso a esta tecnología fue la utilización de los circuitos VLSI (*Very Large Scale Integration / Muy Alta Escala de Integracion*) como los DSP's (*Digital Signal Processor / Procesadores Digitales de Señales)* y los FPGA's (*Field Programmable Gate Array / Arreglo de Compuertas Programable en Campo*) que han reducido drásticamente los costos y le han otorgado gran robustez y adaptabilidad a este tipo de sistemas. En especial los FPGA's cuya tecnología permite procesos en paralelo y el poder ser reprogramables a nivel de hardware, le da ciertas ventajas sobre otras tecnologías [6].

Capítulo I:

Sistema de comunicación OFDMA.

La demanda de las comunicaciones móviles han forzado a la utilización de múltiples técnicas que permitan lograr las tazas de transferencias propuestos por la ITU (*International Telecommunication Union / Unión Internacional de Telecomunicaciones)* para satisfacer las necesidades del mercado y poder ser consideradas tecnologías 4G. Por esto 4G no está limitada a una sola técnica de transmisión y recepción, siempre y cuando se logren las velocidades de transmisión de 100 Mbit/s para terminales en movimiento y hasta 1Gbit/s para terminales fijas se puede considerar 4G.

Una de las técnicas que ha tenido gran aceptación es OFDM, una técnica que por otro lado permite él envió simultáneo de varios símbolos sin tener interferencia entre ellos, la versión multiusuario OFDMA ha ido remplazando a OFDM gracias a las ventajas que ofrece.

1.1 Redes 4G.

Las tecnologías de la información, las comunicaciones en general y las comunicaciones móviles tienen un papel importante en el crecimiento económico, la competitividad y la mejora de la productividad. El terminal móvil ha llegado a constituir hoy en día una parte esencial en la esfera de objetos personales. En este contexto, la industria de las comunicaciones móviles ha venido aportando soluciones al mercado, en la forma de sucesivas generaciones de sistemas [1].

En el principio de las generaciones móviles, el terminal era usado solo para llamadas telefónicas, luego se añadió envío de mensajes de texto a una velocidad de 9,6 kbps y con este cambio se produjo un avance a la siguiente generación. La evolución 2G (*Second Generation / Segunda Generación)*, está caracterizada por el aumento en la velocidad para transmitir datos y la conexión a internet lo que ha producido un mayor consumo y requerimientos por parte de los clientes, por ello la velocidad fue una limitante en esta generación. 3G (Third Generation / Tercera Generación) permite solventar las deficiencias de 2G logrando aumentar la velocidad a un máximo de 4Mbps, lo que requirió el rediseño de los equipos del cliente y las arquitecturas de red para que sean compatibles a estas capacidades, sin embargo siguió limitado en la capacidad de la radio y las bandas de frecuencias, para que tanto tráfico pudiera ser abastecido a la población, por esta razón la evolución de 3G se enfocó en nuevos métodos de modulación y codificación para

la transmisión. Las nuevas características de las arquitecturas de red, también han permitido abastecer no solo al servicio móvil, sino también se puede prestar el servicio de conexión a internet a equipos fijos y notebooks, esto ha permitido competir con las conexión de banda ancha fijas [3].

Las redes 4G están basadas completamente en el protocolo IP ya que se unen redes de cable e inalámbricas. De este modo es más eficaz la transferencia de información y se pueden alcanzar los niveles establecidos por la ITU. También se incluyeron técnicas de rendimiento avanzado de radio como los sistemas MIMO (*Multiple-Input Multiple-Output / Múltiples Entradas Múltiples Salidas.)* y OFDM [4].

4G es una evolución considerable de las redes móviles con la que podremos disfrutar de altas velocidades de transmisión de información de forma completamente inalámbrica. Esta nueva era de redes móviles se estableció tras un comité del ITU. Aquí se establecieron los nuevos niveles de velocidades de transmisión de información. Estos niveles son de 100 Mbit/s para terminales en movimiento y hasta 1Gbit/s para terminales fijas [3].

Más del 50% de la población mundial tendrá acceso a Internet dentro de los próximos tres años. En efecto, según la edición de 2014 del informe “Estado de la banda ancha” del ITU, la banda ancha móvil para teléfonos móviles y tabletas es la tecnología que crece más rápidamente en la historia de la humanidad [7].

En México las compañías de telecomunicaciones inalámbricas más importantes como Telcel, Movistar, Iusacell y Nextel han comenzado desde finales del 2013 a hacer renovaciones en sus equipos de comunicaciones con el fin de satisfacer las demandas del mercado [7].

Actualmente, México cuenta con cerca de 2 millones de suscriptores 4G LTE (*Long Term Evolution / Evolución a Largo Plazo*), según datos de “*The Competitive Intelligence Unit”*. Para el año 2016, se espera que las conexiones LTE se aproximen a los 4 millones. Para 2019, las conexiones de banda ancha móvil 3G y 4G en América Latina y el Caribe rondarán el 85% del total de líneas móviles, de acuerdo con proyecciones de la consultora “*Ovum Telecoms”*. Con unos 196 millones de suscriptores, la tecnología LTE representará más del 23% de las líneas celulares en actividad [8].

1.2 Antecedentes de los sistemas OFDMA.

De manera conceptual OFDM ha existido durante décadas, pero su implementación real y costos aceptables no fue posible sino con el surgimiento y propagación de tecnologías como los microprocesadores de alta velocidad de procesamiento y los dispositivos de lógica programable para poder hacer fiable el procesamiento digital

requerido. Así se ha llegado a una tecnología muy compatible con aplicaciones de comunicaciones inalámbricas.

La principal característica para su implementación en la actualidad es la utilización de la FFT para lograr la transmisión por medio de subportadoras paralelas de manera que se elimina en lo posible la interferencia o traslape entre ellas. Por eso el dato del número de subportadoras está ligado al número de muestras que usa la FFT. Así es que, de manera general, OFDM hace referencia a la transmisión de una trama digital que requiere una elevada tasa de transferencia mediante N líneas paralelas más lentas, en subportadoras contiguas y ortogonales que transportan símbolos independientes que son producto de algún tipo de modulación digital como QPSK (Quadrature Phase Shift Keying / Modulación por Desplazamiento Cuadrafásica), 16-QAM (Quadrature Amplitude Modulation / Modulación de Amplitud en Cuadratura), 64-QAM, etc. dependiendo del sistema [2].

En contraposición a lo que vemos en las típicas comunicaciones monoportadora, donde cada símbolo se transmite en forma serial ocupando todo el ancho de banda disponible, en una modulación multiportadora se envían los símbolos paralelamente en subportadoras adyacentes, es decir, usando algún tipo de FDM (*Frequency Division Multiplexing / Multiplexación por División de Frecuencia*). Una comparación gráfica se puede observar en la Figura 1.1 donde en *a)* se observa que toda la banda asignada se utiliza para transmitir un solo símbolo a la vez, mientras que en *b)* la banda asignada es dividida en varias portadoras lo que permite enviar igual números de símbolos simultáneamente.

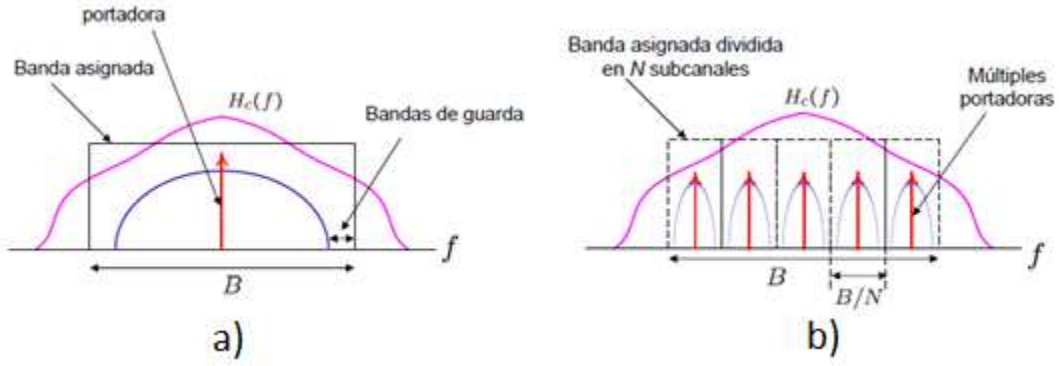

Figura 1.1.- Comparación grafica entre a) Modulación monoportadora y b) Modulación multiportadora.

Diversos métodos para FDM han sido ampliamente utilizados para canales selectivos en frecuencia, tal como lo sería un canal con multitrayectoria. El problema constante que había presentado este tipo de multiplexaje es la prevención del traslape entre subportadoras, lo que exige la colocación de una banda espectral de

separación entre ellas, la cual debía ser igual a lo que permitieran la precisión de los filtros en el receptor [5].

La práctica general de evitar la superposición espectral de subcanales se aplicó para eliminar la ICI (*Inter-Carrier Interference / Interferencia Entre Portadoras*) para lo cual era requerido una banda de guarda. Esto dio lugar a la utilización ineficiente del espectro existente. Una idea fue propuesta a mediados de la década de 1960 para hacer frente a este despilfarro. Los subcanales fueron dispuestos de modo que las bandas laterales de las portadoras individuales se superponen sin causar ICI. Para lograr esto, las portadoras deben ser matemáticamente ortogonales. De esta restricción nació la idea de OFDM.

Siendo *N* subportadoras ortogonales las que se utilizan para un sistema OFDM cualquiera, éstas estarán separadas en frecuencia justamente por el valor correspondiente a la inversa del tiempo útil del símbolo OFDM, durante este período se transmitirán *N* símbolos independientes codificados por QPSK, 16-QAM, 64-QAM o cualquier otro tipo de modulación I/Q que consiste en descomponer la señal en su componente de fase (conocida como componente I) y en cuadratura (componente Q) las cuales son ortogonales entre si [9].

El primer esquema OFDM se remonta a 1966, cuando Robert W. Chang publicó su trabajo pionero en la síntesis de señales ortogonales de banda limitada para la transmisión de datos multicanal. Posteriormente, fue emitida una patente en 1970 por su trabajo. Él presentó un nuevo esquema de transmisión de señales de forma simultánea a través de un canal de banda limitada y sin ICI e ISI (*Inter-Symbol Interference / Interferencia Entre Símbolos*). La idea principal de OFDM es dividir el canal selectivo de frecuencia en un número de subcanales de frecuencias paralelas [5].

En la Figura 1.2 se observa un esquema de implementación de OFDM con 8 frecuencias ortogonales, realizada totalmente en la etapa de RF (*Radiofrecuencia*). Se observa la dependencia del sistema de la precisión de cada oscilador y demás elementos que pueden introducir espurias como mezcladores y divisores, y debemos contar con las no linealidades de filtros y amplificadores que suponen una demodulación coherente en el receptor. Si tomamos en cuenta que normalmente se requerirían muchas más subportadoras, es fácil imaginar la complejidad y costos elevados de este tipo de dispositivos. A pesar de estas limitantes este concepto fue implementado ya en los años 60’s para usarse en radios militares de alta frecuencia [5].

Más adelante se presenta la DFT (*Discrete Fourier Transform / Transformada Discreta de Fourier*) y la IDFT (*Inverse Discrete Fourier Transform / Transformada Discreta de Fourier Inversa*) como los métodos que le darían viabilidad a OFDM en lugar de los bancos de osciladores y la inmensa y costosa circuitería de RF. El uso de estos algoritmos implica pasar el trabajo a una etapa de procesamiento digital de

señales. Por eso la implementación pudo hacerse efectiva con los avances en dispositivos VLSI y en el desarrollo de los algoritmos FFT [2].

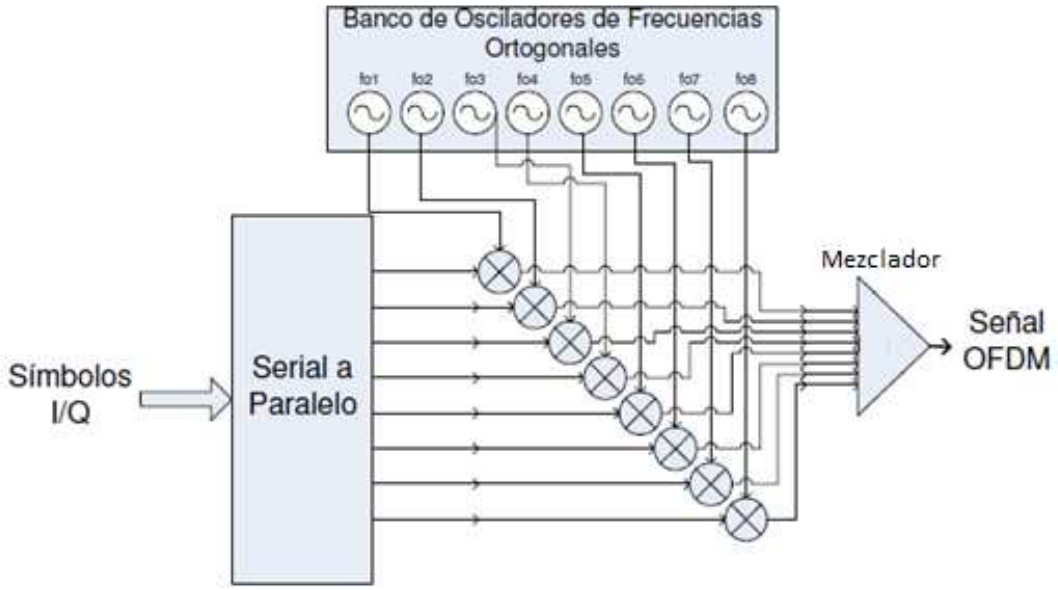

Figura 1.2.- Un sistema OFDM implementado en forma analógico de 8 subportadoras

Fue en 1971 que Weinstein y Ebert introdujeron la IFFT y la FFT para OFDM junto con el concepto de intervalo de guarda para evitar la ISI y la ICI. Desde ese entonces la técnica empezó a ocupar un sitio importante dentro de las comunicaciones; siendo posiblemente en la actualidad su aplicación más difundida y trascendental el Estándar Europeo para DVB-T (*Digital Video Broadcasting–Terrestrial / Difusión de Video Digital por Redes Terrestres*), cuya forma práctica se denomina COFDM (*Coded OFDM / OFDM Codificado*). Pero también está el estándar Europeo para DAB (*Digital Audio Broadcasting / Difusión de Audio Digital*) y otros para transmisión de datos como Cable-MODEM y ADSL (*Asymmetric Digital Subscriber Line / Línea de Abonado Digital Asimétrico*) [9].

1.3 Descripción del sistema de comunicación.

Un sistema de comunicación OFDMA está constituido como se indica en la Figura 1.3 y Figura 1.4 donde se representa en diagrama a bloques, la estructura básica de un transmisor y un receptor respectivamente.

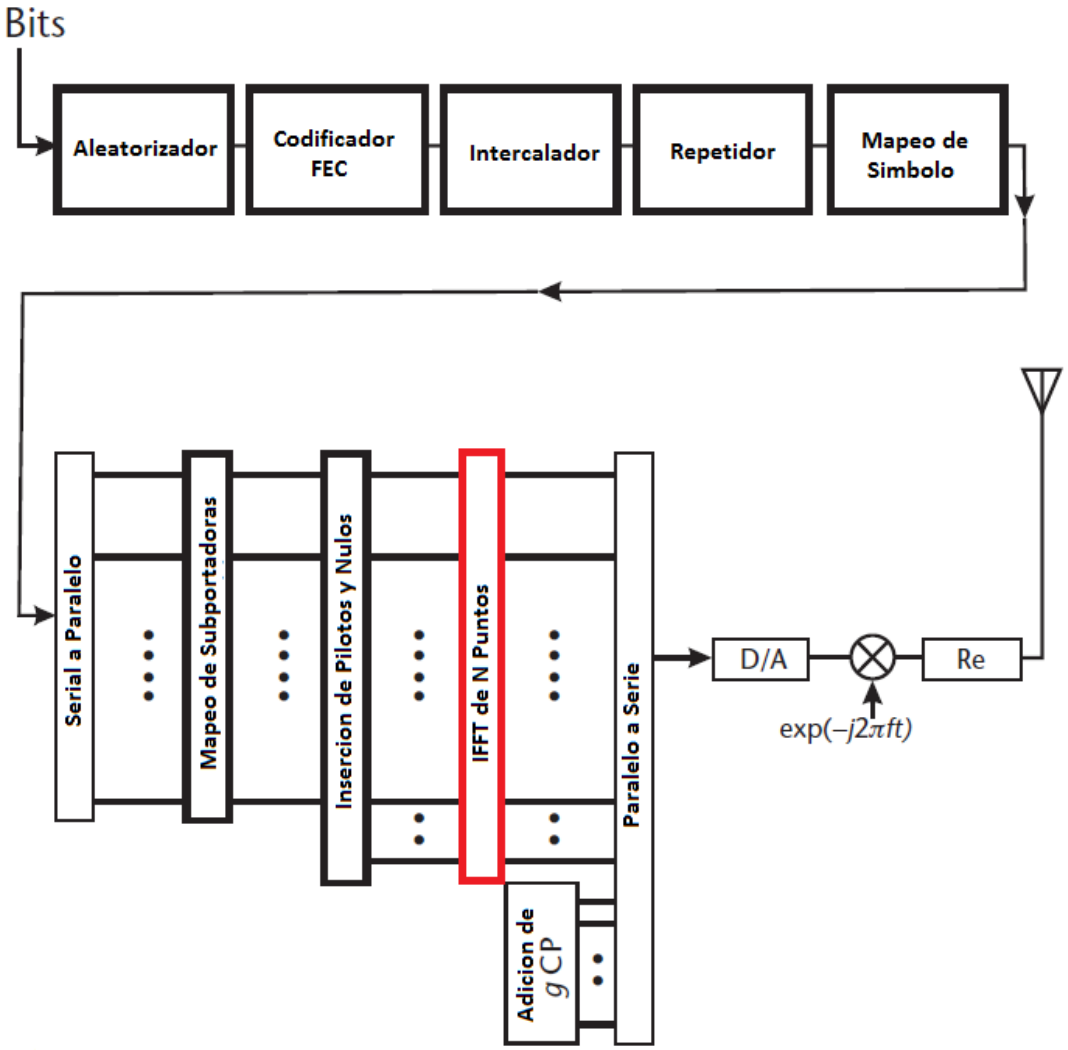

Figura 1.3 Diagrama a bloques de un transmisor OFDMA.

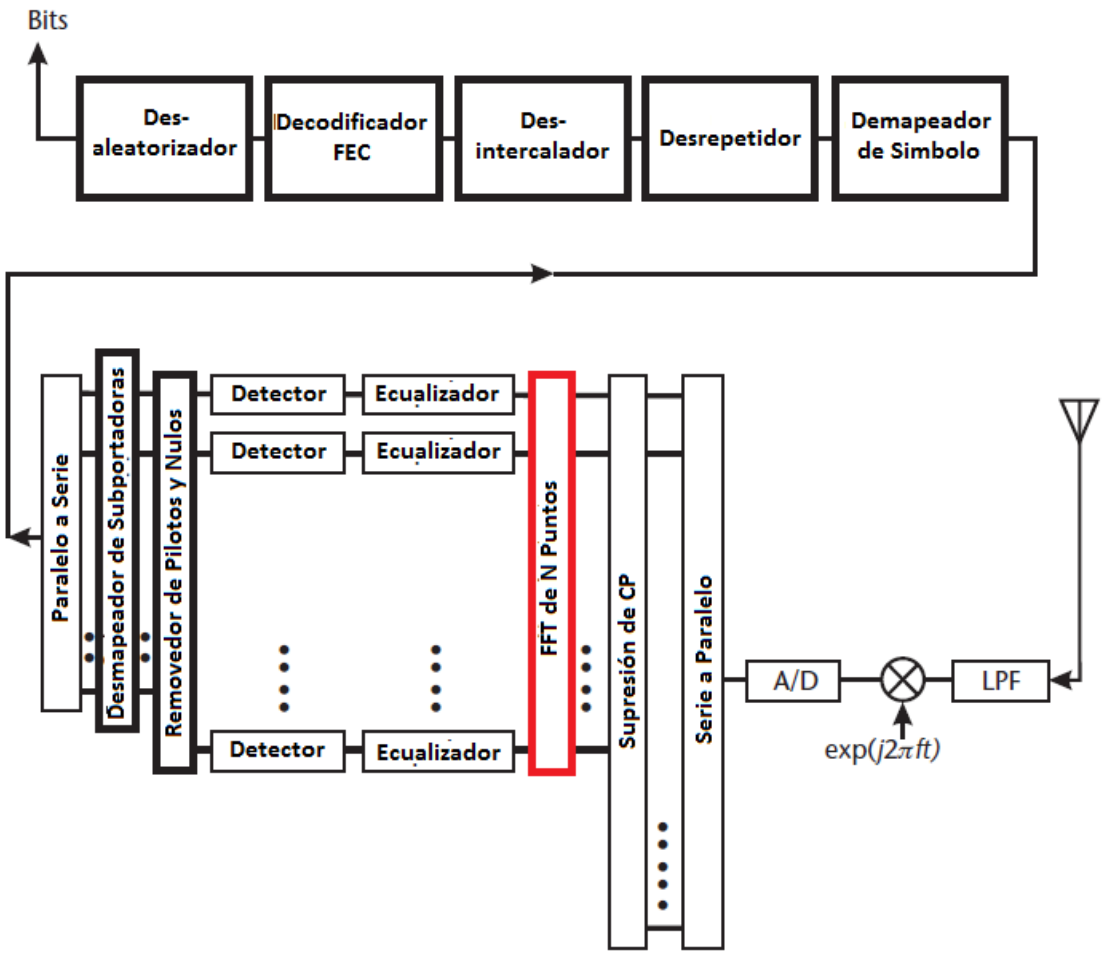

Figura 1.4.-Diagrama a bloques de un receptor de OFDMA.

Tanto en el transmisor y receptor de OFDMA existen un bloque donde se desea cambiar la señal del dominio de la frecuencia al dominio del tiempo y viceversa lo cual conlleva calcular la IDFT y DFT, ya que sus versiones IFFT y FFT dan los mismo resultados utilizando un método más eficiente, es lógico usarlo en lugar de un cálculo directo [2].

Estos bloques son la columna vertebral de este tipo de sistemas ya que gracias a ellos se realiza los procesos de modulación y demodulación en múltiples trayectorias con la certeza de ser ortogonales entre sí, característica de gran importancia que permite la transmisión simultánea de múltiples símbolos.

1.3.1 Importancia de la FFT en los sistemas OFDM.

Desde la aparición de los sistemas OFDM en los años 60's la implementación real resulto ser un gran reto por la complejidad que representaban los procesos de modulación y demodulación que en esos momentos se hacía de forma analógica usando bancos de osciladores y filtros con sus correspondientes problemas de acoplamiento, desfases y precisión de los filtros.

Poco después apareció el artículo de Cooley-Tukey sobre la FFT lo que trajo grandes cambios en muchas áreas de las ciencias, además de que las aplicaciones de este nuevo algoritmo crecieron drásticamente. En 1971 Weinstein y Ebert introdujeron la IFFT y la FFT para la modulación de OFDM con lo cual los osciladores y filtros fueron innecesarios [5].

La tendencia en comunicaciones es la transmitir mayor información en menor tiempo, por lo que cualquier proceso en un sistema de comunicación que pueda reducirse en tiempo es bienvenido, es ahí donde la FFT es de suma importancia y como se explicara más adelante es un método que reduce drásticamente el número de operaciones para obtener la DFT lo que conlleva a una modulación y demodulación mucho más rápida que si se usara un método directo [2].

Como ejemplo de la reducción de tiempo en la obtención de la DFT, si tuviéramos que obtener la DFT de $N = 2^{10}$ en el caso hipotético de que un procesador realizara diez operaciones por segundo, el cálculo directo se realizaría en 1 día y 5 horas, mientras que si se calculara con el algoritmo FFT base-2 DIT (*Decimation in Time / Dicemación en Tiempo*) el tiempo empleado seria de 8 minutos y treinta y dos segundos. Es fácil imaginar el potencial donde los sistemas embebidos logran realizar millones de operaciones en fracciones de segundo y las velocidades que pueden alcanzarse en los sistemas de comunicación a diferencia de que no se utilizara ningún algoritmo FFT [10].

1.4 Aplicaciones.

Actualmente OFDMA se usa en el modelo de movilidad del estándar IEEE 802.16, conocido comercialmente como WiMAX (*Worldwide Interoperability for Microwave Access / Interoperabilidad mundial para acceso por microondas*). OFDMA también se está usando en un enlace de descarga mejorado para 3GPP (*Third Generation Partnership Project / Proyecto de Asociación de Tercera Generación*) que se llama HSOPA (*High Speed OFDM Packet Access / Alta velocidad de acceso a paquetes OFDM*). También es uno de los candidatos para proporcionar el acceso en el IEEE 802.22, conocida como WRAN (*Wireless Regional Area Networks / Red Inalámbricas de Área Regional*), de ser así sería el primer diseño de un sistema

cognitivo de radio en las bandas bajas de VHF (*Very High Frequency / Muy Alta Frecuencia*) y las de UHF (*Ultra High Frequency / Ultra Alta Frecuencia)*, que son las bandas que comúnmente se usan para transmitir televisión.

OFDMA se utiliza en:

- El modo de movilidad del estándar IEEE 802.16 WMAN (*Wireless Metropolitan Area Network / Red de Área Metropolitana Inalámbrica*), comúnmente contemplado como WiMAX (*Worldwide Interoperability for Microwave Access / Interoperabilidad Mundial para Acceso por Microondas)*.
- El estándar IEEE 802.20 comúnmente conocida como MBWA (*Mobile Broadband Wireless Access / Acceso de Banda Ancha Movil Inalámbrica)*.
- El descendente de la 3GPP LTE estándar de cuarta generación para móviles. La interfaz de radio fue nombrada anteriormente HSOPA, ahora llamado E-UTRA (*Evolved UMTS Terrestrial Radio Access / Acceso de Radio Terrestre Envolvente UMTS*).
- El Qualcomm Flarion Technologies móvil Flash-OFDM.
- El ahora extinto proyecto de Qualcomm 3GPP2, concebido como un sucesor de CDMA2000, pero reemplazado por LTE [2].

1.5 Los FPGA's en las comunicaciones móviles e inalámbricas.

El procesamiento de señales discretas avanzo con pasos desiguales durante un largo periodo de tiempo. Hasta principios de los años cincuenta el procesamiento de señales se realizaba con circuitos electrónicos o incluso con dispositivos mecánicos.

Un gran crecimiento del hardware reconfigurable en los últimos años ha hecho posible la implementación de telecomunicaciones complejas relacionadas a algoritmos de procesamiento de señales digitales. Hay varios FPGA's disponible ahora con millones de compuertas programables, han crecido en sus densidades lógicas y se han reducido en tamaño tremendamente. Los FPGA's más recientes están hechos de millones de compuertas lógicas con módulos multiplicadores especializados para el procesamiento de señales altamente computacionales tales como OFDM. Además contienen herramientas concisas y completas para el apoyo en la implementación de OFDM. Además, hay herramientas fáciles de usar disponibles que se pueden utilizar para programar los FPGA's en muy poco tiempo. Por lo tanto, es requisito relevante para diseñar un sistema de comunicación multiportadora basado OFDM hacer pruebas en FPGA, donde varios sistemas diferentes pueden ser probados en el escenario de la vida real [6].

En los últimos años, los FPGA's han llegado a ser claves en la implementación de sistemas de procesado digital de señales de altas prestaciones, especialmente en áreas como las comunicaciones digitales, las redes, etc. Su capacidad para implementar arquitecturas altamente paralelas hace de los FPGA's la opción ideal para tareas como filtrado digital y FFT.

Sin embargo, existe poca familiaridad con el diseño hardware en general, y con los FPGA's en particular. Los ingenieros acostumbrados a desarrollar sistemas empleando DSP, suelen ser expertos programadores en C o en lenguaje ensamblador, pero carecen de experiencia en el campo del diseño digital empleando algún HDL (*Hardware Description Language / Lenguaje de Descripción de Hardware*) como *VHDL (Very High Speed Integrated Circuit HDL / HDL para Circuito Integrado de Muy Alta Velocidad*), Verilog o AHDL (*Altera HDL / HDL de Altera)* [11].

En este sentido, parece fundamental ir avanzando en la dirección del diseño hardware, adquiriendo experiencia en el desarrollo de sistemas de comunicaciones reales basados en FPGA. Por ello, se estableció como objetivo principal de este proyecto realizar un diseño orientado a una implementación en FPGA.

Aunque las computadoras digitales ya han sido utilizados en entornos de negocios y en laboratorios científicos, éstos son caros y de capacidad relativamente limitada. Uno de los primeros usos de las computadoras digitales en el procesamiento de señales fue en la prospección petrolífera. Se grababan los datos sísmicos en cintas magnéticas para su posterior procesamiento. Este tipo de tratamiento de señales no se podía realizar generalmente en tiempo real. Aunque el procesamiento de señales mediante computadoras digitales ofrecía tremendas ventajas de flexibilidad el procesado no se podía realizar en tiempo real. La aportación de Cooley y Tukey (1965) de un algoritmo eficiente para el cálculo de la transformada de Fourier aceleró el uso de la computadora digital. Muchas aplicaciones desarrolladas requerían del análisis espectral de la señal y con las nuevas transformadas rápidas se redujo en varios órdenes de magnitud el tiempo de cómputo. Además, se dieron cuenta de que el nuevo algoritmo se podría implementar en hardware digital específico, por lo que muchos algoritmos de procesamiento digital de señales que previamente eran impracticables comenzaron a verse como posibles [12] [13].

Capítulo II:

La Transformada Discreta de Fourier.

A principios del siglo XIX Jean-Baptiste Joseph Fourier a partir de sus estudios sobre la vibración de una cuerda y la propagación de calor, logra desarrollar una teoría matemática sobre la representación de funciones por medio de series trigonométricas.

Durante los siguientes 100 años se desarrolla lo conocido hoy en día como análisis de Fourier y durante ese lapso estas herramientas eran usadas dentro de la física y las matemáticas principalmente, sin tener una gran relevancia fuera de estas áreas. Todo esto cambiaria tras la publicación de un pequeño artículo en 1965 que proponía un método eficiente de obtener la DFT que además era compatible con las computadoras que en ese momento estaban mostrando el gran impacto que tendrían en el futuro [14].

2.1 Análisis de Fourier.

El análisis de Fourier surgió a partir del intento del matemático francés por hallar la solución a un problema práctico, la conducción del calor en un anillo de hierro, demostró que cualquier señal continua podía ser reconstruida a partir de una suma infinita de senos y cosenos que no son más que ondas básicas también llamados armónicos.

Existen dos casos fundamentales para aplicar el análisis de Fourier, para funciones periódicas y funciones no periódicas para las cuales se utiliza la serie de Fourier y la transformada de Fourier respectivamente [23].

En la ingeniería existen funciones básicas además del seno y el coseno que se emplean para simular señales físicas tales como señales rectangulares, diente de sierra, triangulares, etc… Estas señales tienen una característica particular que es la de repetirse en lapsos iguales de tiempo o lo que es lo mismo son funciones periódicas.

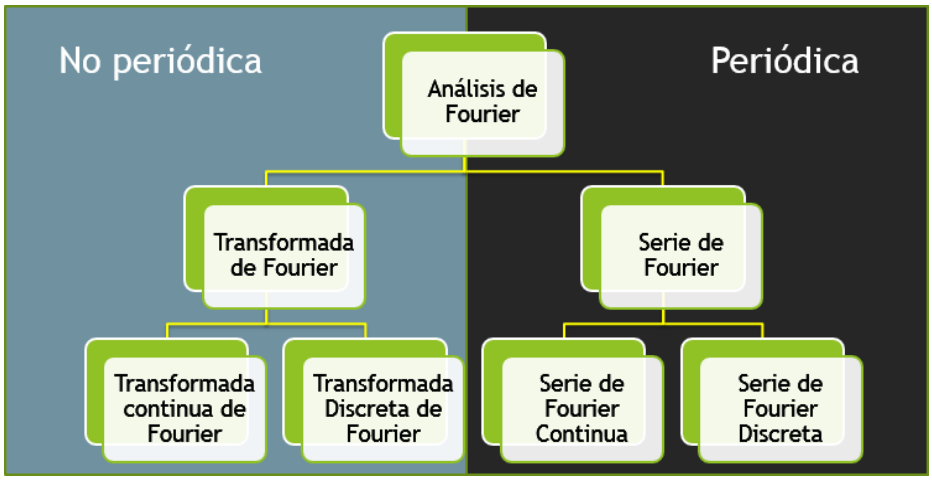

Figura 2.1.- Análisis de Fourier.

Como se ilustra en la Figura 2.1 del análisis de Fourier se desprenden diversos métodos que se ocupan para diferentes tipos de señales ya que la transformada de Fourier solo se aplica para señales no periódicas y la serie de Fourier solo sirve para señales periódicas en ambos caso se cuenta con una versión para señales continuas o discretas.

2.2 Origen de los algoritmos Cooley-Tukey.

Durante una reunión del comité de asesoría científica del entonces presidente de EE.UU. John F. Kennedy se debatía la idea de detectar pruebas nucleares en la Unión Soviética sin la necesidad de visitar las instalaciones nucleares para la ratificación de un tratado con el objetivo de eliminarlas.

Una de las ideas más sobresalientes era la de analizar los registros de sismógrafos instalados en el lecho marino, para lo cual era necesario utilizar la DFT pero debido al tamaño de los registros y el número de sismógrafos necesarios era poco viable. Fue aquí donde John W. Tukey concibió la idea básica de un algoritmo eficiente para el cálculo de la DFT la cual discutió con Richard Garwin de IBM.

Garwin deseoso de implementar estas ideas se reunió con James W. Cooley que trabaja también en IBM al cual convenció de relegar sus proyectos y dedicarse completamente a desarrollar este programa. Cooley desarrolló el programa rápidamente con el fin de sacarse este trabajo de encima y poder seguir con sus proyectos. Sin embargo, poco después de haberlo terminado empezaron a llegarle peticiones del programa e instrucciones para implementarlo por lo cual se contempló la idea de patentarlo, pero finalmente se decidió hacerlo del dominio público [15].

En Abril de 1965 se publicó en la revista *Mathematics of Computation* un pequeño artículo de apenas 5 páginas sin grandes pretensiones titulado "*An algorithm for the machine calculation of complex Fourier series*". Poco después de la publicación de este articulo la comunidad científica que trabajaba en el procesamiento digital de señales se concentró alrededor de este algoritmo lo cual produciría en muy poco tiempo una avalancha de aplicaciones para este algoritmo, consecuencia directa de acortar significativamente la brecha entre el dominio del tiempo y la frecuencia de hasta varios cientos en lo que se refiere a carga computacional, lo cual crea un nuevo termino llamado FFT aplicable a aquellos algoritmos que sugieran una forma más eficiente de obtener la DFT con respecto al método directo [16][14].

Aunque el artículo de Cooley-Tukey es un punto de inflexión en lo que a las FFT y el Procesamiento Digital de Señales se refiere, es importante señalar los trabajos previos a este.

2.3 Algoritmos desarrollados antes de Cooley-Tukey.

Los registros más antiguos de la FFT se han logrado rastrear hasta Carl Friedrich Gauss que trabajando con cálculos astronómicos deseaba obtener la interpolación de orbitas de asteroides de un conjunto de muestras equiespaciadas, concibió la idea básica de un algoritmo para el cálculo eficiente (1805) de la DFT similar a lo propuesto por Cooley-Tukey (1965) incluso antes de que Fourier publicara sus trabajos sobre la representación de funciones por medio de series trigonométricas (1807). Los trabajos de Gauss relacionados a la FFT fueron publicados post-mortem en un recopilatorio en latín, debido a esto y aunado a la notación y sobre todo a la forma trigonométrica que utilizaba Gauss propicio que no se les pudiere relacionar con la FFT [17].

La próxima referencia importante da un salto de 100 años cuando en 1903 Carl Runge formula un algoritmo para muestras igual a potencias de 2, que después generalizo para potencias de 3, este hecho sentó las bases de los PFA (*Prime Factor Algorithm / Algoritmo de Factores Primos*) los cuales fueron usados durante los años 40's difundiéndose ampliamente, sin embargo, desapareció después de la segunda Guerra Mundial.

Para 1942 Gordon C. Danielson y Cornelius Lanczos presentaron un algoritmo mostrando como reducir una transformada de largo $2N$ en dos transformadas de N puntos, utilizando solamente N operaciones de sumas [15].

Estos últimos algoritmos empezaban a tomar en cuenta las ventajas computacionales de los algoritmos de la FFT. Aunque no trascendieron mucho, una forma de explicar esto y del porque la publicación de Cooley-Tukey desbordo de atención es que estos últimos presentaron sus trabajo justo cuando el desarrollo de

la tecnología de los semiconductores comenzaba a avanzar de forma importante, permitiendo al procesamiento digital de señales avanzar a la par, lo que hacía necesario métodos que explotaran de forma eficiente los recursos computacionales con los que se contaba [10].

2.4 Algoritmos posteriores a Cooley-Tukey.

De los desarrollos posteriores de Cooley-Tukey destacan sobre todo el de Yavne que en 1968 presento un artículo de poca difusión debido a que sus ideas eran poco aplicables en ese tiempo, a pesar de que proponía el algoritmo con el menor número de multiplicaciones y sumas tanto para valores reales o complejos. Esta marca del número de operaciones necesarias se mantuvo durante mucho tiempo, además aparecieron algoritmos más simples que igualaron su eficiencia [15].

Una prueba de que se trabajaba en muchos lugares al mismo tiempo sobre cómo obtener un algoritmo más eficiente de la DFT se presentó en 1984 cuando se publicaron casi simultáneamente 4 artículos (Duhamel-Hollmann, Martens, Stasinski, Vetterli-Nussbaumer) los cuales crearon las bases de los métodos de raíces partidas que básicamente es usar un algoritmo base-2 para la parte par y un algoritmo base-4 para la parte impar que es una modificación que tomo 15 años vislumbrar a partir de los trabajos de Yavne [16].

Desde 1960 Irving John Good mostró como descomponer una DFT en varias DFT de números primos por ejemplo si $N = 240$ se podría obtener con tres DFT de tamaño 16, 3 y 5 donde $240 = (16)(3)(5)$ además en 1968 C. M. Rader demuestra que una FFT puede ser obtenida a través de una convolución circular de $N - 1$ muestras. Estos resultados no pasaron de ser anecdóticos ya que la convolución circular era mucho más complicada que la FFT [14].

Shamuel Winograd en 1976 aplica sus algoritmos que reducían el número de multiplicaciones para las convoluciónes circulares a los resultados previamente obtenidos por Good y Rader creando una nueva familia de FFT's conocida como WFTA (*Winograd Fourier Transform Algorithm / Algoritmo de la Transformada de Fourier de Winograd*) para transformadas de cierta longitud. La gran desventaja de estos algoritmos junto con los ya mencionados PFA es que se basan en conceptos matemáticos muy complicados como el Teorema Chino del Resto lo cual ocasiono un retraso para ser implementado y que finalmente provocó su relegación por nuevos algoritmos más eficientes y mucho mejor adaptados para las nuevas tecnologías que estaban surgiendo [18].

2.5 Algoritmos desarrollados en los últimos años.

Uno de los algoritmos más nuevos fue presentado por Steven G. Johnson y Matteo Frigo en 2006 que es una modificación al método de raíces partidas que logro después de 38 años batir la marca de menor número de sumas y restas impuesto por Yavne con una reducción de operaciones del 6% creando una familia de FFT's conocida como FFTW (*The Fastest Fourier Transform in the West / La más Rápida Transformada de Fourier en el Oeste)* que ha tenido gran relevancia en la comunidad científica al ser un algoritmo de software libre [19].

En 2012 investigadores del MIT (*Massachusetts Institute of Technology / Instituto tecnológico de Massachusetts*) presentaron en el ACM-SIAM Symposium on Discrete Algorithms del año 2012 un nuevo algoritmo el cual se basa en la ponderación de ciertas frecuencias las cuales pueden ser eliminadas permitiendo al algoritmo tomar ciertos atajos. Esta transformada podría tener mayor relevancia sobretodo en la compresión de imágenes y audio. Entre menos densa sea la información a tratar se puede obtener un incremento de la eficiencia para la obtención de la DFT de 10 a 100 veces mayor que la FFT, este nuevo algoritmo es conocido como SFT (*Spears Fourier Transform / Tranformada Dispersa de Fourier)* debido a su reciente descubrimiento aún no se han especificado sus alcances reales [20].

2.6 Aplicaciones de la FFT.

La publicación de Cooley-Tukey en 1965 fue el último de los factores necesarios para que el procesamiento digital de señales se impulsara en su desarrollo práctico y teórico ya que facilitó el análisis de señales en el dominio de la frecuencia y mucho más importante, al crear un algoritmo eficiente para la obtención de la DFT logró que las computadoras pudieran reducir los tiempos de esta tarea hasta en cientos de veces además de que seguía aumentado dicha eficiencia conforme se incrementara el número de muestras en comparación a como se estaba realizando hasta entonces. Adicionalmente, el uso de la FFT comenzó a aplicarse en áreas muy variadas además del procesamiento digital de señales como lo son la paleontología, geología, economía, etc.

Hablar de las aplicaciones de la FFT es hablar de un sin fin de implementaciones en un gran número de áreas que van desde cálculos astronómicos para analizar orbitas de cuerpos celestes, pasando por implementaciones en transmisores y receptores de sistemas de comunicación, codificadores de audio y video como lo

son el mp3 y MPEG-4, hasta análisis de índices bursátiles para pronósticos en la bolsa [21].

A continuación se enlistan solo algunas de estas aplicaciones:

- Análisis de antenas.
- Autocorrelación y correlación cruzada.
- Compresión de Ancho de banda.
- Convolución.
- Procesamiento de electrocardiogramas y electroencefalogramas.
- Bancos de filtros.
- Simulación de filtros.
- Espectrofotómetros IR (*Infrared radiation / Radiación Infrarroja*)-TF (*Fourier Transform / Transformada de Fourier*).
- Codificación de imágenes fractales.
- Demodulación PSK (*Phase Shift Keying / Modulación por Desplazamiento de Fase*).
- Medidas de Calidad de imagen.
- Interpolación.
- Imagen de Resonancia magnética.
- Modulación por portadoras de múltiples canales.
- Detección de múltiples frecuencias.
- Filtrado de ruido.
- Sistemas de Sonar y radar.
- Solución numérica de ecuaciones diferenciales.
- Modulación y demodulación de OFDM y OFDMA.
- Procesamiento de señales ópticas.
- Analizadores de funciones.
- Análisis espectral.
- Rotación de imágenes en 2D y 3D.

En sus principios la FFT fue planeada para su implementación en computadoras de propósito general por lo cual resultaba muy atractivo reducir el número de multiplicaciones a cualquier costo lo cual se vio reflejado en la década de los 60’s y principio de los 70’s pero con la llegada de los circuitos VLSI se empezaron a tomar en cuenta otros aspectos que resultaron ser de gran relevancia, como el área del chip y el número de entradas. Para principios de los 90’s aparecieron los primeros DSP con un módulo especializado para la FFT con lo cual se planteó la idea de que la FFT no solo podía ser mejorado por software sino también por hardware [22].

Capítulo III:

Fundamentos matemáticos de la FFT e IFFT.

La DFT es un caso del análisis de Fourier donde se requiere que una función sea no periódica y este discretizada, estas dos características son muy importantes de tomar en cuenta ya que casi todas las funciones que existen en la naturaleza son no periódicas y el uso de computadoras para almacenar datos y analizarlos obliga a que estas funciones sean discretizadas. Un análisis directo de la DFT es un modo poco eficiente para obtener el espectro de una señal y que exige grandes recursos en los sistemas computacionales por lo cual se opta por otros métodos como la FFT que obtiene el mismo resultado pero de una forma más eficiente el cual hace posible el ahorro de grandes recursos a la hora de su implementación en sistemas digitales. La DFT y la FFT tienen varias características matemáticas que pueden ser explotadas para poder reducir el número de operaciones necesarias para su obtención.

3.1 La Serie de Fourier.

Si una señal $x(t)$ es periódica y satisface ciertas condiciones, se puede representar en el dominio de la frecuencia mediante un número infinito de componentes sinusoidales relacionadas armónicamente con la frecuencia fundamental. La amplitud y fase de cada armónica se especifican mediante la Serie de Fourier [23].

La representación de una función $x(t)$ donde T es el periodo de la señal, mediante su serie de Fourier está dado por:

$$x(t) = a_0 + 2\sum_{n=1}^{\infty}[a_n \cos(2\pi n f_0 t) + b_n sen(2\pi n f_0 t)] \qquad (3.1)$$

Donde

$f_o = 1/T$ es la frecuencia fundamental.

$$a_0 = 1/T \int_{-T/2}^{T/2} x(t)dt \qquad (3.2)$$

Que corresponde a la componente continua.

$$a_n = 1/T \int_{-T/2}^{T/2} x(t)\cos(2\pi f_0 t)dt \tag{3.3}$$

$$b_n = 1/T \int_{-T/2}^{T/2} x(t)\mathrm{sen}(2\pi f_0 t)dt \tag{3.4}$$

a_0, a_n, b_n son conocidos como coeficientes de Fourier [10].

3.2 La Transformada de Fourier.

Es obvio que no todas las funciones son periódicas, en la práctica estas constituyen un número muy reducido, existen aquellas cuyo periodo es infinito es aquí donde entra el concepto de transformada de Fourier que se utiliza para este último tipo de señales.

La Transformada de Fourier para una función $x(t)$ con $T = \infty$ está definida como:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi tf}\,dt \tag{3.5}$$

Donde

$X(f)$ es la transformada de Fourier de $x(t)$.

Dicho proceso puede invertirse por medio de:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(f)e^{j2\pi tf}\,df \tag{3.6}$$

Este par de integrales por lo general son de difícil resolución en forma analítica pero el uso de métodos numéricos con ayuda de computadoras digitales ha llevado a la creación de una versión discreta de la transformada de Fourier [23].

3.3 La Transformada Discreta de Fourier (DFT).

Si tenemos una función continúa en el tiempo y es muestreada a intervalos de tiempo iguales, podemos obtener una versión discreta para el análisis de Fourier el cual generara una función discreta del espectro de la señal, limitada en frecuencia.

Cabe señalar que este análisis es válido tanto para una señal continua muestreada como para una secuencia de números.

Las fórmulas para la DFT e IDFT son [10]:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\left(-j\frac{2\pi}{N}nk\right)}, \qquad k = 0, 1, \dots, N-1 . \tag{3.7}$$

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k) e^{\left(j\frac{2\pi}{N}nk\right)}, \qquad n = 0, 1, \dots, N-1 . \tag{3.8}$$

Muchas veces después de aplicada la DFT y hecho su correspondiente análisis en frecuencia se requiere volver a el dominio del tiempo por lo cual es necesario utilizar la IDFT. Como se puede ver el procedimiento es similar al utilizado para la DFT solo difieren en el signo del exponente del exponencial y el factor de escalamiento $1/N$. Los procesamientos computacionales para evaluar la ecuación 3.8 son enteramente similares a los necesarios para evaluar la ecuación 3.7 [24].

3.3.1 Computo eficiente de la Transformada discreta de Fourier.

La ecuación 3.7 puede ser reescrita de la siguiente forma:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \qquad k = 0, 1, \dots, N-1 \tag{3.9}$$

Donde W es denominado como factor de rotación y cumple con:

$$W_N^k = e^{-j\frac{2\pi}{N}nk} \tag{3.10}$$

Si desarrollamos de forma directa la ecuación 3.9 obtendremos:

$$X(0) = x[0]W^0 + x[1]W^0 + \cdots + x[N-1]W^0$$
$$X(1) = x[0]W^0 + x[1]W^1 + \cdots + x[N-1]W^{N-1}$$
$$\vdots$$
$$\vdots$$
$$X(N-1) = x[0]W^0 + x[1]W^{\boldsymbol{N-1}} + \cdots + x[N-1]W^{(N-1)(N-1)}$$

Como se observa para la obtención de la DFT de N muestras es necesario hacer $(N-1)(N)$ sumas complejas y $(N)(N)$ multiplicaciones complejas. Esta forma de obtener la DFT no es eficiente, ya que cuando $nk = 0$ se realiza una multiplicación por 1, por lo tanto no es necesario realizar dicha multiplicación

Por otro lado W tiene propiedades de simetría y periodicidad que hacen posible la reducción de multiplicaciones, también hay una estrecha relación entre la DFT y la convolución que ayuda a reducir la complejidad de la DFT, aquellos métodos que logran obtener de una forma eficiente la DFT son conocidos como FFT [21].

3.3.2 Periodicidad y simetría de los factores de Rotación.

Existen dos propiedades de los factores de rotación que hacen más eficiente la obtención de la DFT, estas son la simetría y periodicidad las cuales quedan de manifiesto en las siguientes ecuaciones [21]:

$$W_N^{nk} = W_N^{nk+N} \qquad 3.11$$

$$W^k = W_N^{-nk+N/2} \qquad 3.12$$

La ecuación 3.11 nos dice que W se repetirá cada N veces y en 3.12 se denota que los factores separados por $N/2$ veces solo cambiaran de signo en la parte imaginaria. Como ejemplo y para mejor visualización presentaremos los factores de rotación para $N = 8$.

En la Figura 3.1 podemos ver la propiedad de periodicidad de los factores de rotación los cuales se repiten cada N veces.

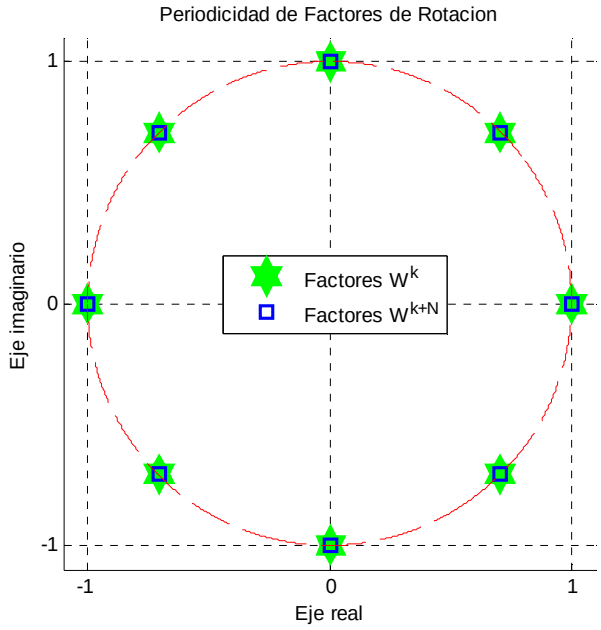

Figura 3.1 .- Periodicidad de los Factores de Rotación para $N = 8$.

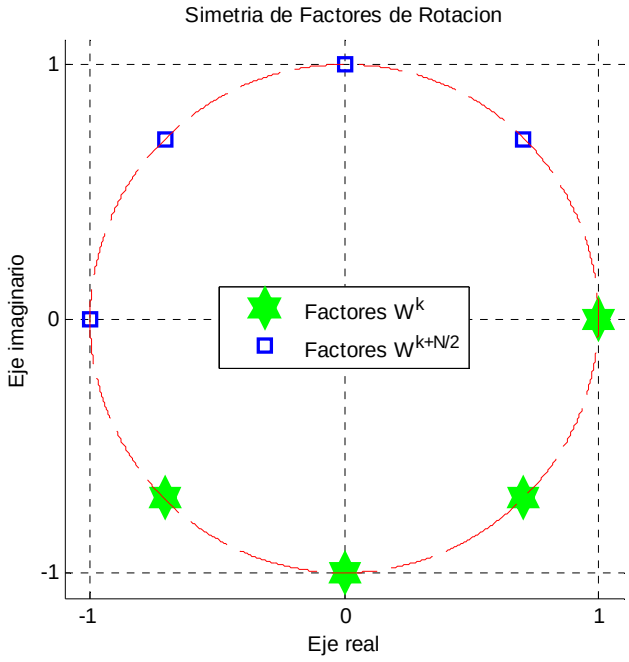

Figura 3.2.- Simetría de los Factores de Rotación para $N = 8$.

En la Figura 3.2 se puede observar la propiedad de simetría de los factores de rotación. Los primeros 4 factores de rotación están en azul y sus simétricos se encuentran en verde el cual se localiza $N/2$ veces del factor original, por lo cual queda de manifiesto que solo es necesario calcular los primeros 4 factores de rotación [21].

Los algoritmos del tipo Cooley-Tukey explotan estas características de los factores de rotación logrando una gran eficiencia en la reducción de operaciones necesarias para obtener la DFT.

3.4 La Transformada Rápida de Fourier (FFT).

El desarrollo de algoritmos rápidos por lo general consiste en utilizar las propiedades especiales del algoritmo de interés para eliminar operaciones redundantes o innecesarios derivadas de una aplicación directa. El cálculo directo del DFT no resulta eficiente pero gracias a la periodicidad, simetría y ortogonalidad que presentan algunas FFT y su relación especial con la convolución aumentan la eficiencia aritmética para la obtención de la DFT.

La FFT puede ser el algoritmo numérico más importante en la ciencia, la ingeniería y las matemáticas aplicadas. Nuevos resultados teóricos siguen apareciendo, los avances en las computadoras y hardware formulan nuevas ideas y enfoques diferentes y las nuevas aplicaciones abren nuevas áreas de investigación [24].

3.4.1 Clasificación de las FFT.

Por motivos de clasificación y para poder centrarse en aquellas FFT que son de mayor relevancia para este trabajo separaremos los diferentes grupos de FFT. Este trabajo no es fácil ya que existen muchas y muy variadas formas de diferenciarlas sin embargo lo haremos de acuerdo al el número de muestras que pueden manejar.

Existen dos grupos principalmente:

- Algoritmos Cooley-Tukey para $N = m^l$.

- Algoritmos de Factores Primos para $N = (a)(b)$.

Donde *N* es el número de muestras, *m* y *l* son enteros positivos y *a, b* son números primos entre sí, positivos y enteros.

Los primeros son conocidos como algoritmos Cooley-Tukey, estos son los más usados, conocidos y documentados ya que dos de ellos ofrecen grandes ventajas para los sistemas computacionales, que son los algoritmos base-2 y base-4. Cabe

aclarar que a manera de ejemplo en el artículo de Cooley-Tukey se presentó un algoritmo base-2 por las ventajas computacionales que ofrecía [22].

Existen muchas técnicas para implementar un algoritmo de este tipo como lo son la DIT, DIF (*Decimation in Frequency / Dicemación en Frecuencia*) y el cómputo en el lugar que se utiliza para ahorrar uso de memoria. Además de otros métodos necesarios para la indización y la obtención de los coeficientes de rotación, estos últimos solo se dan en este tipo de FFT lo cual representa su principal desventaja ante los algoritmos de factores primos [20].

Un caso especial es el conocido como algoritmo de raíces partidas el cual logra una eficiencia mucho mayor al conjuntar los algoritmos de base-2 y base-4 que de forma aislada [15].

Los PFA son usados para su implementación en computadoras de uso general de los cuales el más conocido es el WFTA y el Algoritmo de Goertzel muy utilizado para filtros digitales.

Estos algoritmos se sustentan en que una FFT puede ser obtenida mediante una convolución circular que junto con métodos eficientes para calcular dichas convoluciónes logrando una gran eficiencia en este tipo de FFT's [18].

Se analizaran los dos grupos de algoritmos para determinar las ventajas y desventajas que tendrían para su uso en un sistema OFDMA y su implementación en un sistema embebido.

3.4.2 Algoritmos Cooley-Tukey.

Los algoritmos Cooley-Tukey requieren que el número de muestras a transformar pueda ser expresado como potencia de un número entero positivo.

La idea básica de este tipo de algoritmos radica en poder descomponer una DFT en varias DFT de tamaño *m*.

Lo cual provoca que un mismo bloque pueda ser reutilizada un determinado número de veces logrando reducir las operaciones necesarias de N^2 a $Nlog_n(N)$ entiéndase como operación en este caso al conjunto de una multiplicación compleja y una suma compleja [11].

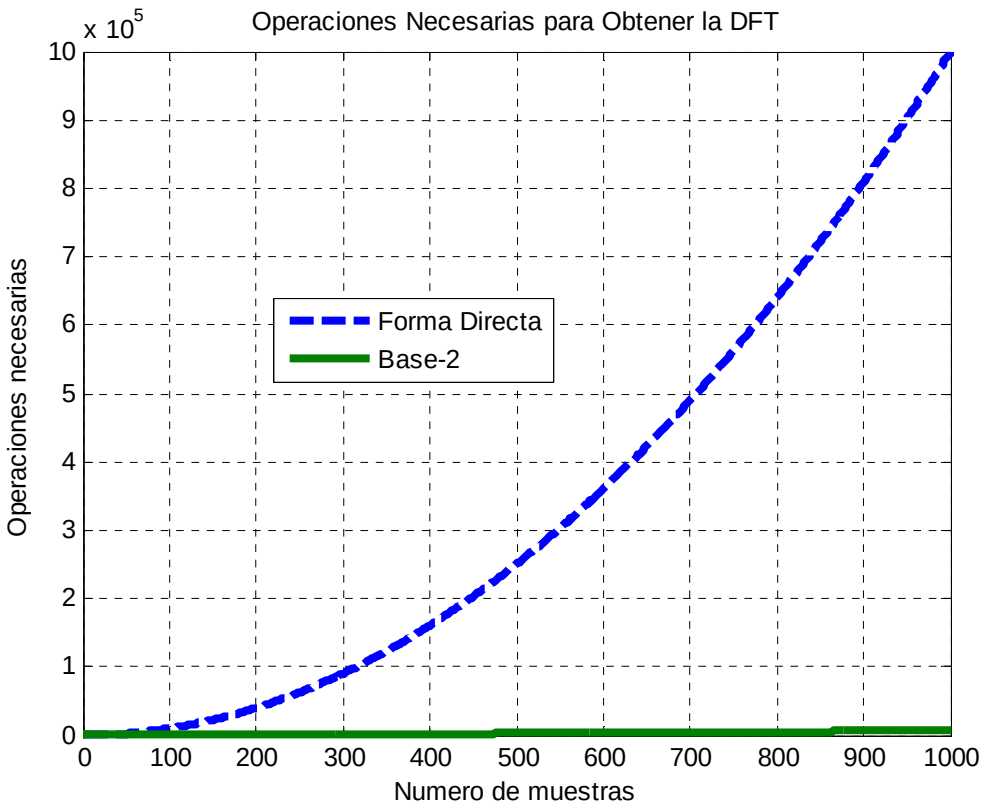

Figura 3.3.- Comparativa del número de operaciones entre el cálculo directo y FFT base-2.

Con la ayuda de la Figura 3.3 podemos ver el porqué de la importancia de los algoritmos FFT y la eficiencia que estos logran. En azul se observa las operaciones necesarias derivadas de un cálculo directo y en verde las operaciones requeridas usando el algoritmo base-2, como se puede ver conforme el número de muestras aumenta el número de operaciones necesarias en comparación con el método directo es mucho más eficiente.

3.4.2.1 Decimación en tiempo y frecuencia.

El cálculo de la DFT se puede hacer mucho más eficiente computando mayor cantidad de DFT's de menor longitud ya que se aprovecha las propiedades de simetría y periodicidad de los factores de rotación. Los algoritmos en los que la señal temporal $x(n)$ se descompone sucesivamente en bloques más pequeños se denominan algoritmos DIT [24].

Como ejemplo ilustrativo tomaremos el caso donde $N = 8$:

Partimos de la idea de que *N* es un entero positivo par por lo cual la DFT puede dividirse en dos DFT de tamaño $N/2$ formadas por los puntos pares de $x(n)$ y los puntos impares de $x(n)$.Si $X(k)$ está dada por:

$$XF(k) = \sum_{n=0}^{N-1} xt(n) e^{\left(-j\frac{2\pi}{N}nk\right)}, \qquad k = 0, 1, \dots, N-1 \qquad 3.13$$

Dividiendo $x(n)$ en las muestras pares e impares se obtiene:

$$XF(k) = \sum_{n\ par} x\,t(n) e^{\left(-j\frac{2\pi}{N}\right)nk} + \sum_{n\ impar} xt(n) e^{\left(-j\frac{2\pi}{N}\right)nk} \qquad 3.14$$

Sustituyendo $n = 2r$ para los n par y $n = 2r + 1$ para los n impares:

$$XF(k) = \sum_{r=0}^{\frac{N}{2}-1} xt(2r) e^{(-j\frac{2\pi}{N})2rk} + \sum_{r=0}^{\frac{N}{2}-1} xt(2r+1) e^{\left(-j\frac{2\pi}{N}\right)(2r+1)k}, \quad r = 0, \dots, \frac{N}{2} - 1. \qquad 3.15$$

$$XF(k) = \sum_{r=0}^{\frac{N}{2}-1} xt(2r) e^{(-j\frac{2\pi}{N})2kr} + \sum_{r=0}^{\frac{N}{2}-1} xt(2r+1) e^{\left(-j\frac{2\pi}{N}\right)2kr}, \qquad r = 0, \dots, \frac{N}{2} - 1. \qquad 3.16$$

Teniendo en cuenta que $e^{(-j\frac{2\pi}{N})2rk} = e^{\left(-j\frac{2\pi}{N/2}\right)rk}$ puede reescribirse la última ecuación como:

$$XF(k) = \sum_{r=0}^{\frac{N}{2}-1} xt(2r) e^{(-j\frac{2\pi}{N/2})kr} + e^{-j\frac{2\pi}{N}k} \sum_{r=0}^{\frac{N}{2}-1} xt(2r+1) e^{\left(-j\frac{2\pi}{\frac{N}{2}}\right)kr}, r = 0, \dots, \frac{N}{2} - 1. \qquad 3.17$$

Podemos reducir la expresión como:

$$XF(k) = G(k) + e^{-j\frac{2\pi}{N}k} H(k) \quad , \qquad k = 0, 1, 2 \dots, \frac{N}{2} - 1. \qquad 3.18$$

Como se puede ver la DFT original ha sido dividida en dos DFT de tamaño $N/2$ el primer factor representa la DFT para las muestras pares y el segundo representa la DFT para las muestras impares. A pesar de que k toma valores de 0 a $N-1$ las sumas se calculan solo para los valores de 0 a $N/2-1$ ya que tanto $G(k)$ como $H(k)$ son periódicas de periodo $N/2$. El factor $e^{-j\frac{2\pi}{N}}$ es un factor muy común por lo tanto se le representa como W_N^{nk}.Quedando finalmente la expresión como [22]:

$$X(k) = G(k) + W_N^{nk}H(k) \quad , \quad k = 0, 1, 2 \ldots, \frac{N}{2} - 1. \qquad 3.19$$

Puesto que esta representación matemática resulta muy abstracta y difícil de entender se muestra la Figura 3.4 donde se representa una DFT para $N = 8$ que se obtiene a partir de dos DFT de tamaño $\frac{N}{2} = 4$ una conformada por las muestras pares y otra de las muestras impares.´

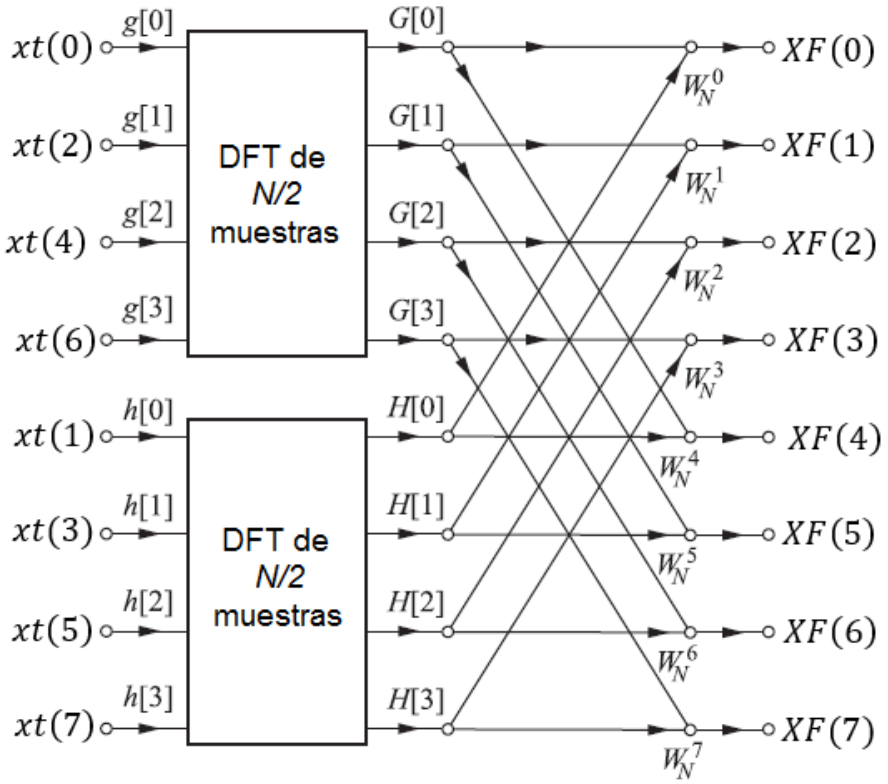

Figura 3.4.- Descomposición de una DFT de $N = 8$ muestras en dos DFT de $N/2$ muestras decimación en tiempo.

Obsérvese que una vez que tenemos dos DFT de tamaño $N/2$, nada nos impide el aplicar nuevamente el procedimiento anterior a las dos DFT obtenidas. Por lo cual la DFT de $N/2$ muestras puede descomponerse en dos DFT de $\frac{N/2}{2}$ muestras así hasta obtener solo DFT's de tamaño 2 como se muestra en la Figura 3.5 [24].

Este método fue propuesto por Cooley-Tukey en su artículo de 1965 por sus características particulares que inferían ciertas ventajas para su implementación en computadoras de propósito general [14].

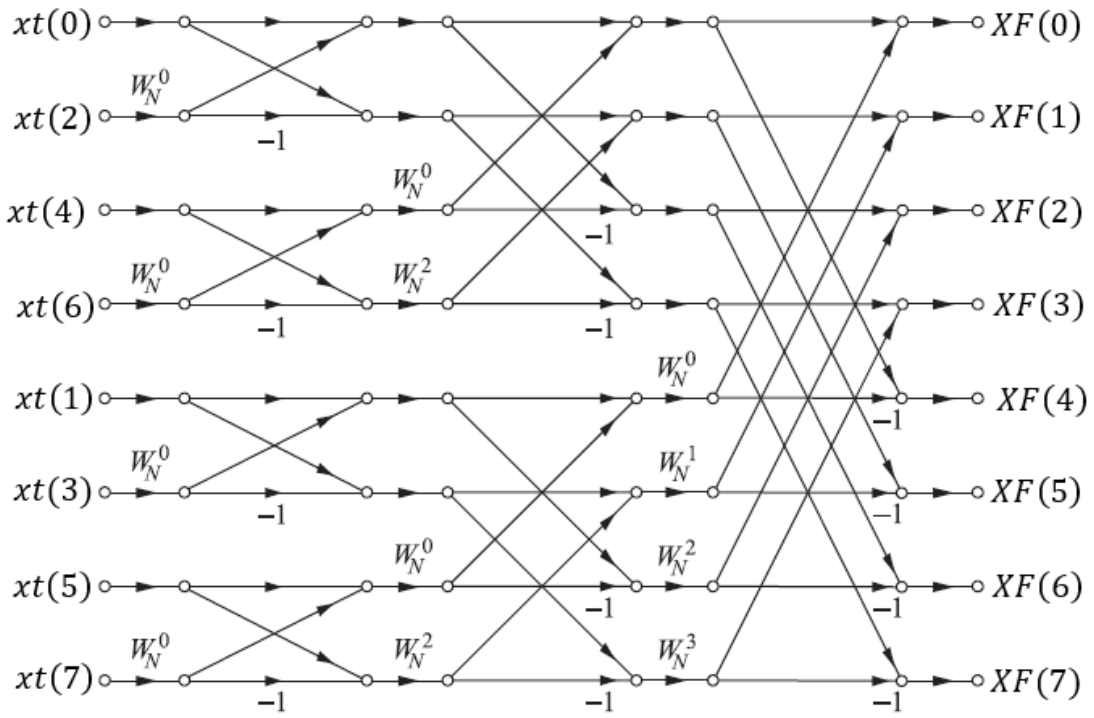

Figura 3.5.- Algoritmo FFT base-2 para $N = 8$ muestras de dicemación en tiempo completo.

En posteriores documentos se dieron varias mejoras al algoritmo propuesto por Cooley-Tukey uno de estos fue la DIF que con un enfoque similar al analizado se logra establecer que al dividir la secuencia $X(k)$ en varias DFT de menor tamaño se obtienen los mismos resultados que con la DIT en cuanto al total de operaciones requeridas para calcular la DFT el mismo algoritmo presentado en la Figura 3.4 de DIT se presenta en la Figura 3.5 con DIF [15].

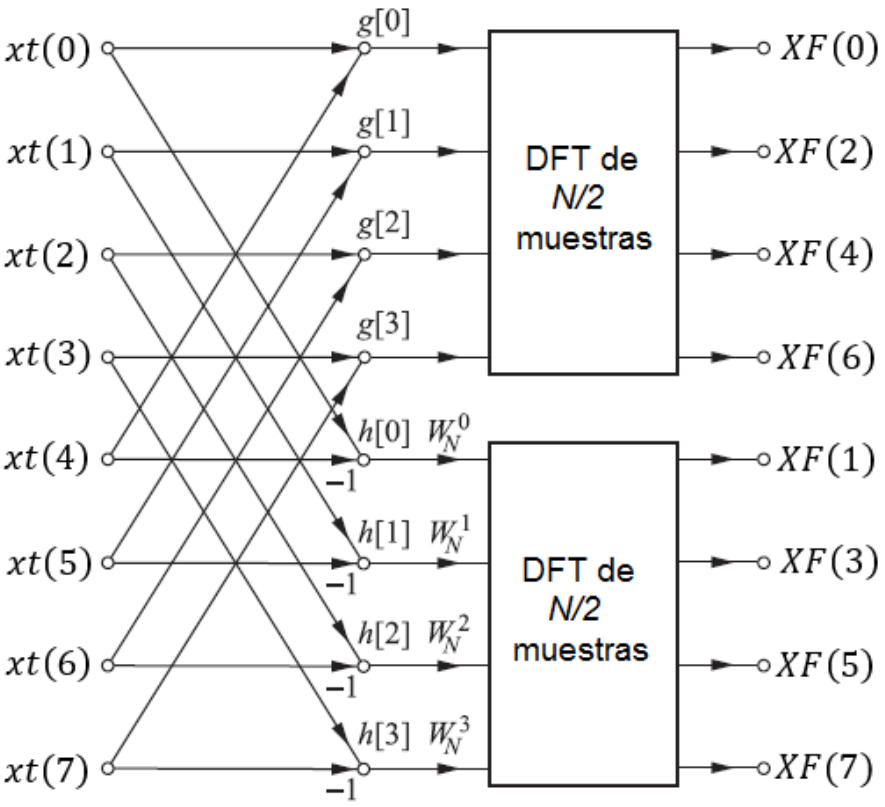

Figura 3.6.- Descomposición de una DFT de $N = 8$ muestras en dos DFT de $N/2$ muestras utilizando decimación en frecuencia.

Estos dos métodos emplean el mismo número de operaciones para el cálculo de la DFT pero cada uno tiene ciertas ventajas que pueden ser de gran relevancia a la hora de seleccionar un algoritmo para un propósito específico el más sobresaliente a simple vista es el orden de las muestras de entrada con respecto al de las salidas [21].

3.4.2.2 Inversión de bit.

Si se observó con detenimiento las Figura 3.4 y Figura 3.6, se puede notar que en ambos casos el orden de la secuencia a la entrada no coincidía con el orden a la salida, esto se debe a la separación que se hace de la DFT en dos DFT donde se agrupan las muestras pares y las impares.

Si se sigue con la decimación completa, el orden de las muestras se ve alterado mucho más, como se muestra a continuación:

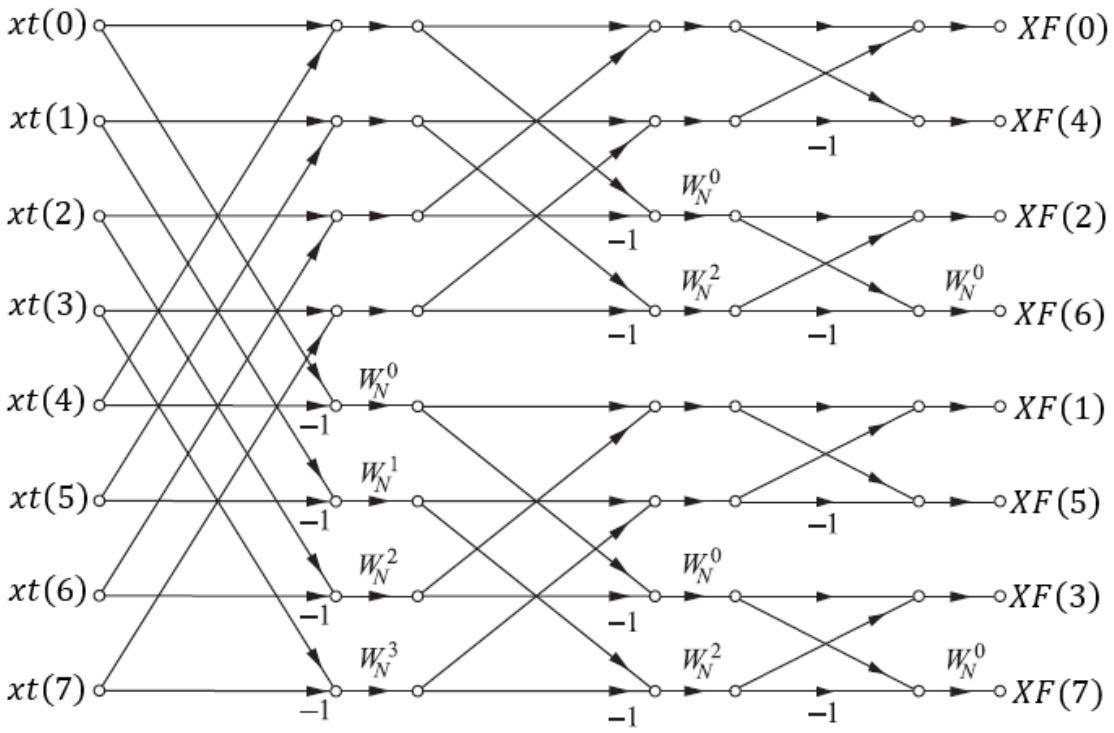

Figura 3.7.- Algoritmo FFT base-2 para N=8 muestras de decimación en frecuencia completo.

Obsérvese que si la secuencia de entrada se encuentra en orden, la salida estará en forma desordenada por lo cual es necesario hacer un reordenamiento de las muestras. Esto resulta relativamente fácil si se toma una representación binaria de la posición de la muestras como se presenta en la Tabla 1:

Tabla 1.- Inversión de bit para 8 muestras.

Posición a la salida	Representación binaria	Posición que le corresponde	Representación binaria
$XF(0)$	000	$XF(0)$	000
$XF(1)$	001	$XF(4)$	100
$XF(2)$	010	$XF(2)$	010
$XF(3)$	011	$XF(6)$	110
$XF(4)$	100	$XF(1)$	001
$XF(5)$	101	$XF(5)$	101
$XF(6)$	110	$XF(3)$	011
$XF(7)$	111	$XF(7)$	111

En este caso para hacer el ordenamiento correcto solo es necesario cambiar el bit más significativo por el menos significativo y viceversa, esto se conoce como inversión de bit dado que se invierte el orden de lectura de la palabra en binario lo cual aplica para cualquier número de muestras que sea potencia de 2 [10].

3.4.2.3 Más allá de base-2.

Aunque todos los ejemplos que se han puesto hasta ahora se han referido a la FFT base-2, lo antes mencionado es aplicable para otras bases como lo son base-4 y base-8. En la práctica se requiere que algunos sistemas de comunicación OFDMA como lo son LTE y 4G trabajen con un numero de muestras que puede ser de 1200 o 2400 las cuales no son potencias de ninguna de las bases mencionadas, por lo que se ha visto la necesidad de utilizar algoritmos FFT base-3 y base-5 que se habían relegado debido a su complejidad y menor adaptabilidad a los sistemas computacionales [25].

Las FFT's base-3 y base-5 se encuentran escasamente documentados y aún menos implementados en sistemas computacionales ya que estos últimos rompen con la tendencia que se ha seguido de adaptar el algoritmo FFT a los sistemas computacionales.

3.4.3 Algoritmos de Factores Primos.

Los algoritmos de factores primos necesitan poder expresar N como una multiplicación de dos números enteros y primos entres si, luego se crean DFT's de menor tamaño las cuales tendrán como base los números primos obtenidos, los cuales serán utilizados una sola vez, a diferencia del método anterior donde la DFT base-m era utilizada tantas veces como fuera necesario.

Este tipo de FFT se basa en conceptos matemáticos muy avanzados como el teorema chino del resto. Para su implementación se requieren realizar DFT's relativamente pequeñas en menos de N operaciones lo cual se logró al desarrollar una DFT de N muestras en una convolución circular de $N-1$ muestras [23].

Como se puede notar este método requiere de más conocimientos matemáticos para su comprensión. Por lo que es más difícil traducir los resultados obtenidos en la teoría a un código de computadora.

Estos algoritmos presentaban resultados teóricos bastantes prometedores, sin embargo una vez implementados no alcanzaban las expectativas esperadas lo cual

condujo a una nueva definición de complejidad que dio más relevancia a las características físicas de las computadoras, como se ilustra en la Figura 3.8 donde podemos observar la metodología para obtener una FFT de factores primos implica varios procesos complejos , además de que los algoritmos base son complejos también [24].

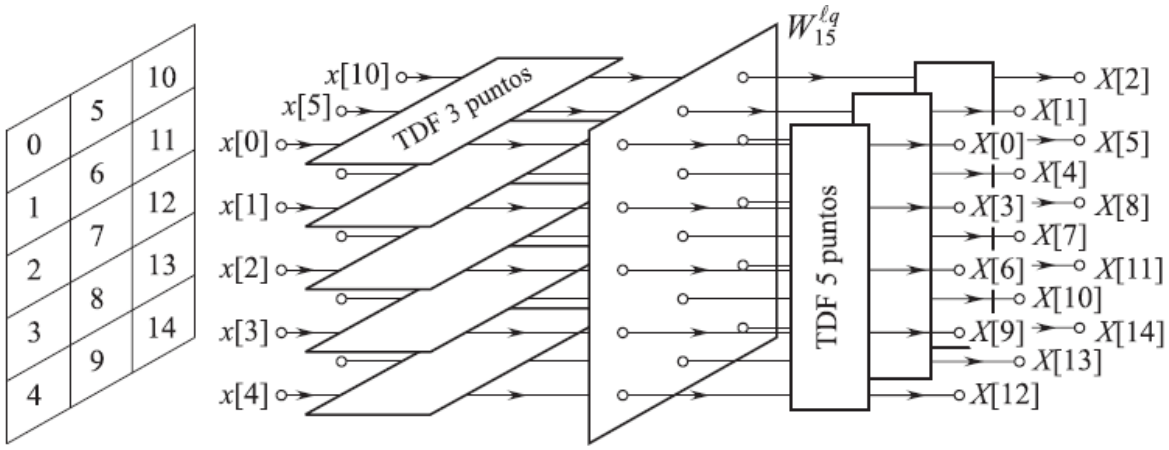

Figura 3.8.- Esquema de la metodología para obtener la FFT para N=15 muestras a partir de los algoritmos base-5 y base-3.

No todo el trabajo realizado en este tipo de algoritmos fue en vano ya que se logró desarrollar una nueva familia de algoritmos basados en DFT de números primos como lo son WFTA y los algoritmos de Goertz [21].

Se logró también la adquisición de nuevos enfoques y conocimientos teóricos que proporcionaron una nueva perspectiva para los algoritmos Cooley-Tukey que podían ser de gran provecho en las nuevas tecnologías que se estaban desarrollando [15].

3.4.3.1 Algoritmo de Winograd.

Este algoritmo le debe su nombre a Shamuel Winograd que propuso manejar la DFT en función de convoluciones o multiplicaciones de polinomios que básicamente es descomponer la DFT original en varias DFT de menor longitud, donde el orden de cada DFT es un número primo [15].

Con el algoritmo de Winograd el número de multiplicaciones necesarias para una DFT de N puntos se reduce de $Nlog_2N$ a N. Aunque este resultado puede parecer muy prometedor esta drástica reducción se logra a partir del incremento significativo

del número de sumas. Sin embargo, en los procesadores modernos muchas veces las sumas y multiplicaciones se ejecutan en los mismos ciclos de reloj, donde son más convenientes los algoritmos Cooley-Tukey [12].

Capítulo IV:

Simulaciones en Matlab.

Para realizar las simulaciones en Matlab utilizaremos las funciones más básicas con el objetivo de que puedan ser reproducidas fácilmente en HDL, aunque ambos estilos de programación son un tanto diferentes ya que Matlab es un lenguaje secuencial y AHDL es concurrente. Matlab será utilizado para confirmar que los algoritmos escogidos son correctos y que la metodología seleccionada es la ideal, además de permitirnos hacer modificaciones para ver la respuesta de los algoritmos, comprobar las propiedades matemáticas de ellos y saber si es necesario modificarlos o no.

4.1 Metodología del algoritmo FFT/IFFT para N=128.

La idea básica es la de descomponer la FFT/IFFT de 128 en FFT/IFFT de menor tamaño, ya que esto nos permitirá reutilizar un mismo bloque varias veces. Partimos de la obtención de la DFT de forma directa utilizando la ecuación 3.7

$$XF(k) = \sum_{n=0}^{127} xt(n) e^{\left(-j\frac{2\pi}{128}nk\right)}, \qquad k = 0, 1, \dots, 126, 127 . \tag{4.1}$$

Desarrollada de forma completa 4.1 obtendríamos:

$$XF(0) = xt(0)W_{128}^{(0)(0)} + xt(1)W_{128}^{(0)(1)} + \cdots + xt(126)W_{128}^{(0)(126)} + xt(127)W_{128}^{(0)(127)}$$
$$XF(1) = xt(0)W_{128}^{(1)(0)} + xt(1)W_{128}^{(1)(1)} + \cdots + xt(126)W_{128}^{(1)(126)} + xt(127)W_{128}^{(1)(127)}$$
$$\vdots$$
$$\vdots$$
$$XF(126) = xt(0)W_{128}^{(126)(0)} + xt(1)W_{128}^{(126)(1)} + \cdots + xt(126)W_{128}^{(126)(126)} + xt(127)W_{128}^{(126)(127)}$$
$$XF(127) = xt(0)W_{128}^{(127)(0)} + xt(1)W_{128}^{(127)(1)} + \cdots + xt(126)W_{128}^{(127)(126)} + xt(127)W_{128}^{(127)(127)}$$

Este es una forma ineficiente de obtener la DFT ya que requiere de 16384 sumas complejas y mismo número de multiplicaciones complejas. Si utilizamos un algoritmo FFT base-2 podemos descomponer la ecuación 4.1 en dos sumatorias de 64 elementos cada una como se muestra:

Teniendo en cuenta que $e^{(-j\frac{2\pi}{N})2rk} = e^{\left(-j\frac{2\pi}{N/2}\right)rk}$ por la propiedad de periodicidad de los factores de rotación puede reescribirse la última ecuación como dos sumatorias una formada por los elementos pares y otra por los elementos impares de $xt(n)$:

$$X(k) = \sum_{r=0}^{63} x(2r)e^{(-j\frac{2\pi}{128/2})kr} + e^{-j\frac{2\pi}{128}k} \sum_{r=0}^{63} x(2r+1)e^{(-j\frac{2\pi}{128/2})kr}, r = 0, \dots \frac{N}{2} - 1 \qquad 4.2$$

Podemos reducir la expresión como:

$$X(k) = G(k) + e^{-j\frac{2\pi}{128}k}H(k) \qquad k = 0, 1, 2 \dots, 63. \qquad 4.3$$

Su representación en un algoritmo de flujo de datos se observa en la Figura 4.1:

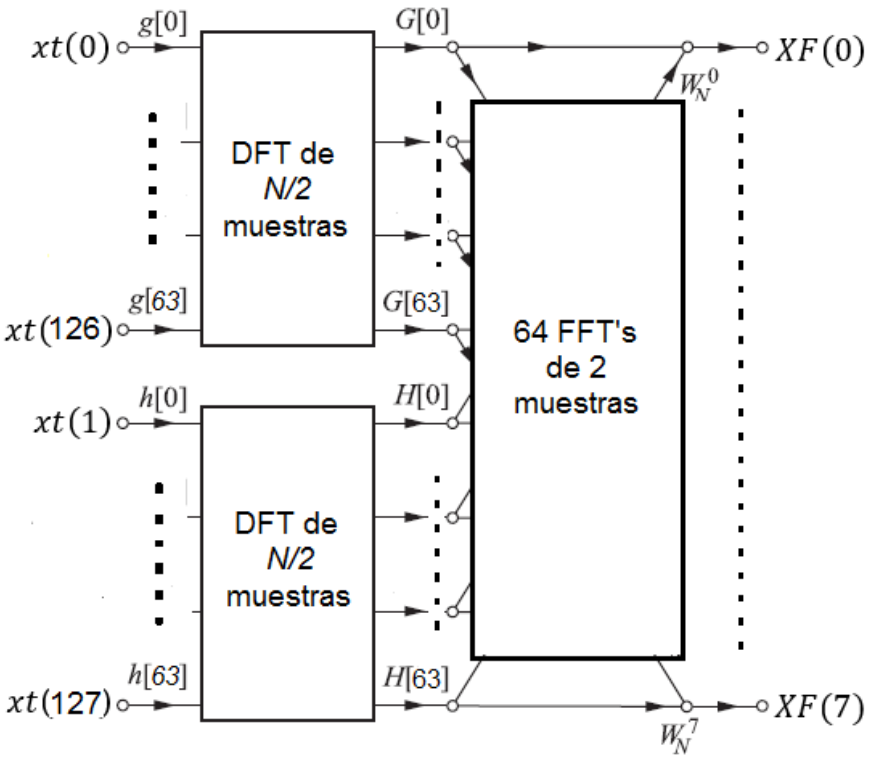

Figura 4.1.- Diagrama a bloques del algoritmo FFT para N=128 base-2 utilizando dicemación en frecuencia.

Con esto hemos reducido la FFT de 128 muestras a 64 FFT's de 2 muestras y 2 FFT's de 64 muestras reduciendo el número de sumas y multiplicaciones complejas a 4160 que es una reducción de poco menos del 26% de operaciones aunque este no es el punto importante lo importante es ver que la FFT puede dividirse en FFT de menor número de muestras según la base que se utilice.

Si utilizáramos la base 8 para la FFT de 128 muestras obtendríamos la siguiente ecuación:

$$XF(k) = \sum_{r=0}^{15} xt(8r)e^{(-j\frac{2\pi}{128/2})kr} + e^{-j\frac{2\pi}{128}k}\sum_{r=0}^{63} xt(8r+1)e^{(-j\frac{2\pi}{128/2})kr} + \cdots + e^{-j\frac{2\pi}{128}k}\sum_{r=0}^{15} xt(8r+6)e^{\left(-j\frac{2\pi}{128/2}\right)kr} + e^{-j\frac{2\pi}{128}k}\sum_{r=0}^{15} xt(8r+7)e^{(-j\frac{2\pi}{128/2})kr}, r = 0, \ldots 15 \quad (4.5)$$

Podemos reducir la expresión como:

$$X(k) = G(k) + e^{-j\frac{2\pi}{128}k}H(k) + \cdots + e^{-j\frac{2\pi}{128}k}M(k) + e^{-j\frac{2\pi}{128}k}N(k) \quad , \quad k = 0, 1, 2 \ldots, 63. \quad (3.18)$$

Con esto hemos reducido la FFT de 128 muestras a 16 FFT's de 8 muestras y 8 FFT's de 16 muestras. Su representación en un algoritmo de flujo de datos se observa en la Figura 4.2. Como se puede ver se ha reducido la FFT de 128 muestras en FFT's de 8 y 16 muestras, esta reorganización de la FFT es la más conveniente ya que se usan las FFT's del menor número de muestras posibles.

Como se puede deducir una vez separada la FFT de 128 muestras en FFT's de menor tamaño, a estas nuevas FFT's también puede aplicarse una reducción similar, en el caso de la FFT de 8 muestras esta se puede dividir en FFT's de 4 puntos y FFT's de 2 puntos. Para el caso de la FFT de 16 muestras puede dividirse en FFT's de 4 muestras todo esto con la obvia reducción de operaciones.

Todo este desarrollo matemático es muy difícil de comprender y además existen otros métodos de obtener los mismos resultados de reducción de la FFT de 128 muestras que son más fáciles de comprender los cuales son mostrados a continuación.

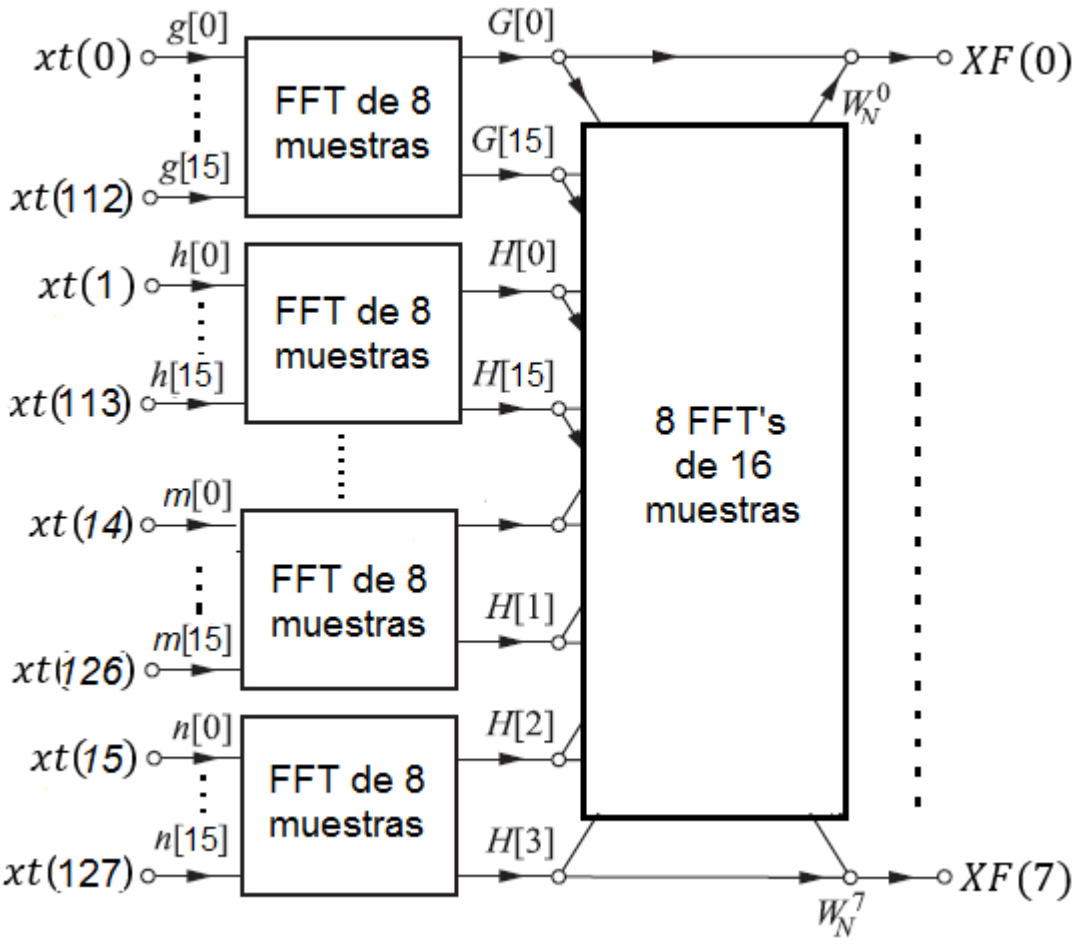

Figura 4.2.- Diagrama a bloques del algoritmo FFT para N=128 base-8 utilizando dicemación en tiempo.

4.2 Algoritmos Cooley-Tukey.

En principio se programó el algoritmo FFT base-2 de la Figura 4.3, ya que es el más simple y es la base para realizar los demás algoritmos cuando $N = 2^l$ (2, 4, 8, 16,…) siendo N el número de muestras a procesar.

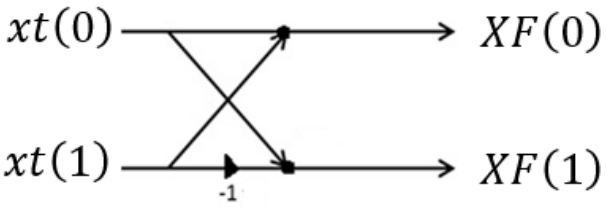

Figura 4.3.- Algoritmo FFT base-2 para N=2.

Se ha seleccionado el algoritmo DIT ya que con este método no es necesario ordenar de forma particular la entrada, cosa que no sucede con la DIF, pero será necesario reordenar el resultado obtenido, ya que el orden no corresponde al deseado, lo cual se lograra con el método conocido como inversión de bit el cual fue explicado en el capítulo III [7].

Si tomamos en cuenta que cada nodo es un punto de suma, que los triángulos son puntos de multiplicación y sabiendo que siempre iremos de izquierda a derecha entonces podemos expresar este algoritmo como:

$$XF(0) = xt(0) + xt(1) \qquad 5.1$$

$$XF(1) = xt(0) - xt(1) \qquad 5.2$$

.

Es importante mencionar que para obtener la IFFT base-2 de 2 puntos se utilizan las mismas formulas obtenidas para la FFT, ya que en este momento no intervienen los factores de rotación por lo cual ambos algoritmos son similares. El factor de escalamiento $\frac{1}{N}$ no será tomado en cuenta en este momento.

Este algoritmo resulta muy sencillo de programar en Matlab como se muestra en el apéndice A.

Se ha creado una función con el nombre “base_2” la cual requiere de dos entradas en este caso $xt(n)$ que debe ser un vector de dos elementos, a su vez regresa un vector de dos elementos donde $XF(k)$ es nuestra FFT.

Ahora tomando como referencia el algoritmo de la Figura 4.4 podemos realizar el script para la FFT base-2 para $N = 4$.

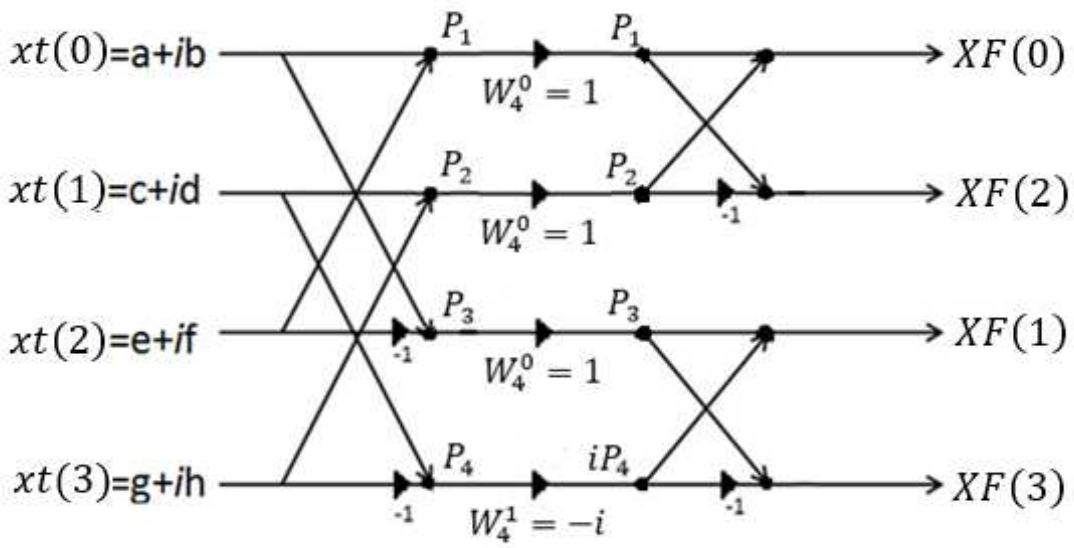

Figura 4.4.- Algoritmo FFT base-2 para N=4.

Siguiendo la metodología marcada por el algoritmo de la Figura 4.4 y representando cada elemento de $x(n)$ en su parte real e imaginaria en la primera etapa obtenemos los puntos P_x los cuales son:

$$P_1 = xt(0) + xt(2) \qquad 5.3$$

$$P_2 = xt(1) + xt(3) \qquad 5.4$$

$$P_3 = xt(0) - xt(2) \qquad 5.5$$

$$P_4 = xt(1) - xt(3) \qquad 5.6$$

Como se puede ver 5.3 y 5.5 cumplen con las ecuaciones del algoritmo base-2, lo mismo pasa con 5.4 y 5.6.

El próximo paso es multiplicar los puntos P_x por los factores de rotación, como se observa en la Figura 4.4 el único factor de rotación diferente de 1 es el correspondiente a P_4 por lo cual es necesario multiplicar este término por i. Los valores de cada factor de rotación son obtenidos de 3.10.

Para obtener los valores finales de $XF(k)$ tenemos que:

$$XF(0) = P_1 + P_2 \qquad 5.7$$

$$XF(2) = P_1 - P_2 \qquad 5.8$$

$$XF(1) = P_3 + iP_4 \qquad 5.9$$

$$XF(3) = P_3 - iP_4 \qquad 5.10$$

Nuevamente $XF(0)$ y $XF(2)$ al igual que $XF(1)$ y $XF(3)$ cumplen con el algoritmo base-2. Si desarrollamos estos términos en términos de la parte real e imaginaria de $xt(n)$ tenemos:

$$XF(0) = (a + ib) + (c + id) + (e + if) + (g + ih) \qquad 5.11$$

$$XF(2) = (a + ib) - (c + id) + (e + if) - (g + ih) \qquad 5.12$$

$$XF(1) = (a + ib) + (c + id) - (e + if) - (g + ih) \qquad 5.13$$

$$XF(3) = (a + ib) - (e + if) - i[(c + id) - (g + ih)] \qquad 5.14$$

Si desarrollamos 5.11, 5.12, 5.13 y 5.14 y separamos las partes reales e imaginarias obtenemos:

$$XF(0) = (a + c + e + g) + i(b + d + f + h) \qquad 5.15$$

$$XF(2) = ((a + e) - (c + g)) + i((b + f) - (d + h)) \qquad 5.16$$

$$XF(1) = ((a + h) - (e + d)) + i((b + c) - (f + g)) \qquad 5.17$$

$$XF(3) = ((a + d) - (e + h)) + i((b + g) - (f + c)) \qquad 5.18$$

Esta última representación tiene una gran ventaja ya que todos los términos de $X(k)$ tanto su parte real e imaginaria se obtienen solamente con sumas y restas.

De manera similar podemos obtener las ecuaciones para IFFT. La única diferencia es en el término de rotación W_4^1 que en este caso toma el valor de i.

El grupo de ecuaciones para obtener la IFFT son:

$$invXF(0) = (a + c + e + g) + i(b + d + f + h) \qquad 5.19$$

$$invXF(2) = ((a + e) - (c + g)) + i((b + f) - (d + h)) \qquad 5.20$$

$$invXF(1) = ((a + d) - (e + h)) + i((b + g) - (f + c)) \qquad 5.21$$

$$invXF(3) = ((a + h) - (e + d)) + i((b + c) - (f + g)) \qquad 5.22$$

De 5.15 a 5.22 podemos ver que $XF(0) = invXF(0)$ y $XF(2) = invXF(2)$. $XF(1)$ solo difiere de $invXF(1)$ en la ubicación de cuatro términos en este caso se intercambian h por d y c por g, lo mismo sucede para $XF(3)$ y $invXF(3)$, se intercambian las mismas variables por lo cual la diferencia entre realizar la FFT y la IFFT se resuelve con una sentencia IF que nos ubicara las variables c, d, g y h donde sean necesarios.

Hasta ahora se han realizado los algoritmos base-2 para $N = 2$ y el algoritmo base-4 para $N = 4$ y una de las características más importantes que podemos resaltar hasta ahora es que no se ha utilizado ninguna multiplicación, esto es muy importante debido a que las multiplicaciones son los recursos más escasos y más importantes de tomar en cuenta como se mencionara más adelante, ya que estas ideas se trasladaran directamente al FPGA.

Además confirmamos una parte muy importante de este trabajo, con la misma estructura se puede obtener la FFT e IFFT. Es importante señalar que este algoritmo no da el resultado correcto de la IFFT ya que carece del factor de escalamiento

$1/N$, esto es omitido debido a que este factor de escalamiento debe ser utilizado en el resultado final y es variable conforme el número de muestras que se estén procesando, ya que será usado en otras funciones no es conveniente agregarlo en este momento, esto facilitara su reutilización en FFT's de mayor tamaño.

Una vez obtenido el algoritmo base-2 de 2 puntos y el algoritmo base-4 de 4 puntos se usaran para realizar el algoritmo base-2 de 8 puntos de la Figura 4.10. Siguiendo una secuencia de pasos podremos a partir de los algoritmos bases poder realizar FFT's de mayor numero de datos.

La metodología para obtener la FFT para $N = 8$ o las subsecuentes de un mayor número de datos se puede dividir en tres etapas, en primer lugar se deben acomodar los elementos de $xt(n)$ en forma matricial como se muestra en la Figura 4.5.

$xt(0)$	$xt(4)$
$xt(1)$	$xt(5)$
$xt(2)$	$xt(6)$
$xt(3)$	$xt(7)$

Figura 4.5.- Forma matricial de xt(n) para la FFT de N=8 base-2.

Esta representación nos permite ver con mayor claridad el manejo de los datos. En la primera etapa a cada fila de la matriz de la Figura 4.5 se le aplica la FFT base-2 de lo cual obtenemos las siguientes ecuaciones:

$$P_1 = xt(0) + xt(4) \quad 5.23$$

$$P_2 = xt(1) + xt(5) \quad 5.24$$

$$P_3 = xt(2) + xt(6) \quad 5.25$$

$$P_4 = xt(3) + xt(7) \quad 5.26$$

$$P_5 = xt(0) - xt(4) \quad 5.27$$

$$P_6 = xt(1) - xt(5) \quad 5.28$$

$$P_7 = xt(2) - xt(6) \quad 5.29$$

$$P_8 = xt(3) - xt(7) \quad 5.30$$

De estas ecuaciones obtenemos la siguiente matriz:

P_1	P_4
P_2	P_5
P_3	P_6
P_4	P_7

Figura 4.6.- Matriz resultante de la primera tapa de la metodología para obtener el algoritmo FFT de 8 muestras.

Para la segunda etapa necesitamos generar una segunda matriz conformada por los factores de rotación W_8^k donde el factor k estará dado de acuerdo a la ubicación del elemento en la matriz. El valor de k se obtiene con:

$$k = (\,fila\, - 1)(columna - 1) \quad 3.31$$

La matriz de factores de rotación queda de la siguiente forma:

W_8^0	W_8^0
W_8^0	W_8^1
W_8^0	W_8^2
W_8^0	W_8^3

Figura 4.7.- Matriz de los factores de rotación de la metodología para obtener el algoritmo FFT de 8 muestras.

Los valores de W_N^k estarán dados por la ecuación 3.10, como se puede observar si el valor $k = 0$ el factor de rotación tendrá el valor de 1 lo cual siempre ocurre en todos los elementos de la primera fila y la primera columna, si se desea obtener la IFFT, el valor de k será el mismo solo que con signo negativo esto hará que se cumpla la ecuación 3.8 correspondiente a la IDFT.

Ahora se multiplicara cada elemento de la matriz resultante de la primera etapa Figura 4.5 con su correspondiente de acuerdo a su ubicación, con la matriz de factores de rotación Figura 4.7 con lo cual obtenemos:

$$Q_1 = P_1 \quad 5.32$$

$$Q_2 = P_2 \quad 5.33$$

$$Q_3 = P_3 \quad 5.34$$

$$Q_4 = P_4 \quad 5.35$$

$$R_1 = P_5 \quad 5.36$$

$$R_2 = W_8^1 P_6 \quad 5.37$$

$$R_3 = W_8^2 P_7 \quad 5.38$$

$$R_4 = W_8^3 P_8 \quad 5.39$$

La matriz resultante de las multiplicaciones queda representada en la Figura 4.8:

Q_1	R_1
Q_2	R_2
Q_3	R_3
Q_4	R_4

Figura 4.8.- Matriz resultante de la segunda etapa de la metodología para obtener el algoritmo FFT de 8 muestras.

El siguiente paso consiste en aplicar una FFT base-4 de 4 puntos a cada columna de la matriz de la Figura 4.8 para lo cual usamos las ecuaciones 5.7, 5.8, 5.9 y 5.10 lo cual nos generaran las ecuaciones:

$$XF(0) = Q_1 + Q_2 + Q_3 + Q_4 \quad 5.32$$

$$XF(1) = R_1 + R_2 + R_3 + R_4 \quad 5.33$$

$$XF(2) = Q_1 - iQ_2 - Q_3 + Q_4 \quad 5.34$$

$$XF(3) = R_1 - iR_2 - R_3 + R_4 \quad 5.35$$

$$XF(4) = Q_1 - Q_2 + Q_3 - Q_4 \quad 5.36$$

$$XF(5) = R_1 - R_2 + R_3 - R_4 \quad 5.37$$

$$XF(6) = Q_1 + iQ_2 - Q_3 - Q_4 \quad 5.38$$

$$XF(7) = R_1 + iR_2 - R_3 - iR_4 \quad 5.39$$

Con esto obtenemos el resultado final que en forma matricial queda de la siguiente forma:

$XF(0)$	$XF(1)$
$XF(4)$	$XF(5)$
$XF(2)$	$XF(3)$
$XF(6)$	$XF(7)$

Figura 4.9.- Matriz final de la metodología para obtener el algoritmo FFT de 8 muestras.

Si se desea representar todo este procedimiento en un diagrama de flujo de datos obtenemos la Figura 4.10:

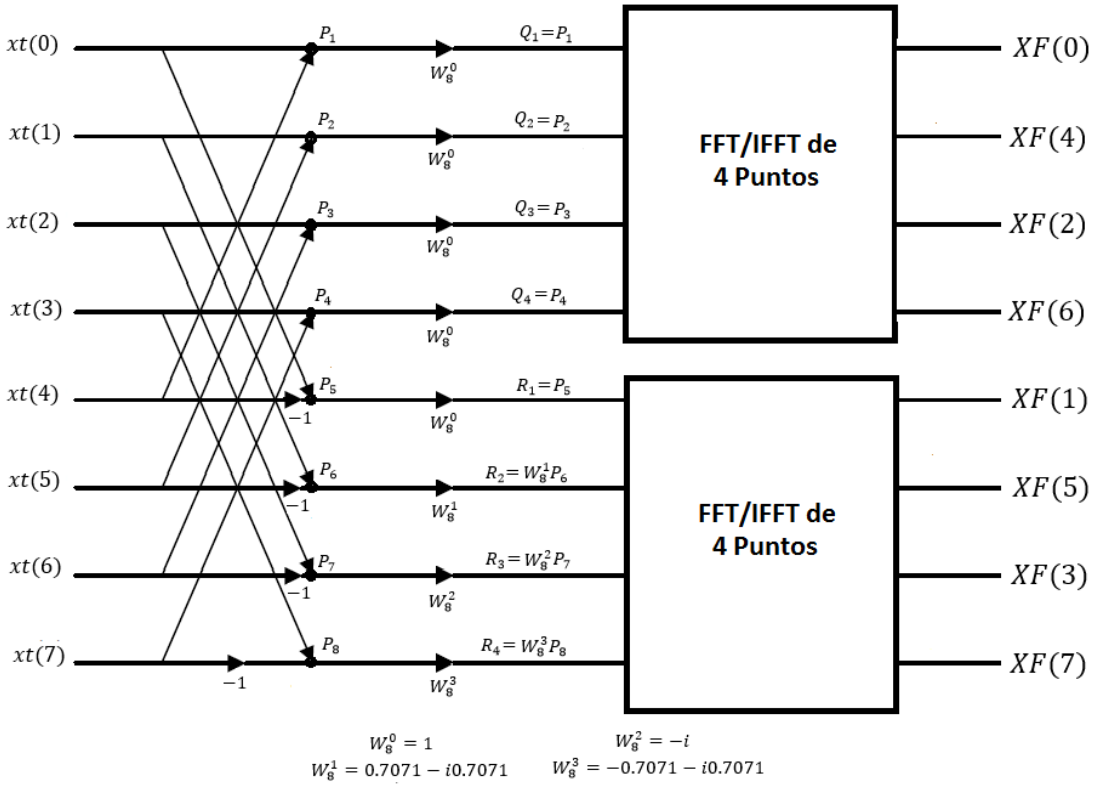

Figura 4.10.- Algoritmo de flujo de datos de la FFT/IFFT de 8 puntos base-2 y base-4.

Esta metodología nos permite dividir la FFT de 8 puntos en FFT's de 2 y 4 puntos lo cual simplificara la programación en Matlab como la implementación en el FPGA.

Para realizar la FFT de 16 puntos utilizaremos la misma metodología que para la FFT de 8 puntos, si utilizáramos la FFT base-2 tendríamos que generar una matriz de 8x2, pero en su lugar utilizaremos base-4, lo cual nos permite generar una matriz 4x4 como se muestras en la siguiente figura:

$xt(0)$	$xt(4)$	$xt(8)$	$xt(12)$
$xt(1)$	$xt(5)$	$xt(9)$	$xt(13)$
$xt(2)$	$xt(6)$	$xt(10)$	$xt(14)$
$xt(3)$	$xt(7)$	$xt(11)$	$xt(15)$

Figura 4.11.- Forma matricial de $x(n)$ para obtener el algoritmo de la FFT/IFFT para N=16 base-4.

Con esto dividimos la FFT de 16 puntos en solo FFT's de 4 puntos, esta forma de acomodar los elementos de la matriz nos permite descartar el uso del algoritmo base-2 de 2 puntos. Siguiendo los mismos pasos que para la FFT de 8 puntos podemos obtener la FFT de 16 puntos.

Con los algoritmos para la FFT de 8 y 16 muestras podemos obtener la FFT de 128 ya que $(8)(16) = 128$, esto se puede lograr usando la misma metodología que se ha ido aplicando para FFT's de mayor número de datos a los algoritmos base. La base de este algoritmo al igual que los demás es la forma matricial de $x(n)$ que en este caso será una matriz de 16 filas por 8 columnas, aunque también es muy válido utilizar una matriz de 8 filas por 16 columnas ya que a la hora de programar en Matlab este cambio no supones diferencias relevantes.

Pero el hecho de seleccionar una matriz de 16x8 u 8x16, si genera grandes diferencias al implementarlo en el FPGA por lo cual se elige el arreglo de 16x8 los motivos se explicaran detalladamente en el capítulo correspondiente.

La forma matricial de $x(n)$ para FFT de 128 muestras queda de la forma que se muestra en la Figura 4.12:

$xt(0)$	$xt(16)$	.....	$xt(96)$	$xt(112)$
$xt(1)$	$xt(17)$	.....	$xt(97)$	$xt(113)$
⋮	⋮	⋱	⋮	⋮
$xt(14)$	$xt(30)$	.....	$xt(110)$	$xt(126)$
$xt(15)$	$xt(31)$	.....	$xt(111)$	$xt(127)$

Figura 4.12.- Forma matricial de $x(n)$ para obtener el algoritmo de la FFT/IFFT para N=128 base-8 y base-16.

Este arreglo matricial como en los casos anteriores nos permite dividir la FFT en FFT's de menor tamaño que en este caso con FFT's de 8 muestras y FFT's de 16 muestras.

El método utilizado podemos resumirlos en 5 pasos:

Paso 1: Se reorganiza la secuencia $xt(n)$ en un arreglo matricial de nxm.

Paso 2: Se realizan las transformadas de tamaño n de los elementos de cada fila del arreglo matricial.

Paso 3: Se realiza la multiplicación elemento a elemento entre la matriz resultante del paso dos y una matriz de factores de rotación.

Paso 4: Se realizan las transformadas de tamaño m de los elementos de cada columna del arreglo matricial obtenido del paso 3.

Paso 5: Se reorganizan nuevamente los elementos de la matriz en una secuencia $XF(k)$.

4.3 Algoritmos de Factores Primos.

Los algoritmos de factores primos siguen una metodología diferente, pero tienen las mismas propiedades ya mencionadas, además de que son diseñados para utilizar el menor número de multiplicaciones posible a cambio de aumentar el número de sumas, esto puede resultar hasta cierto punto una ventaja ya que cuando hablamos de la implementación en el FPGA es sabido que una suma ocupa menos recursos que una multiplicación.

La implementación de los algoritmos base-3 y base-5 no representa gran problema solo es seguir detenidamente la ruta necesaria para cada dato.

4.3.1 FFT/IFFT base-3.

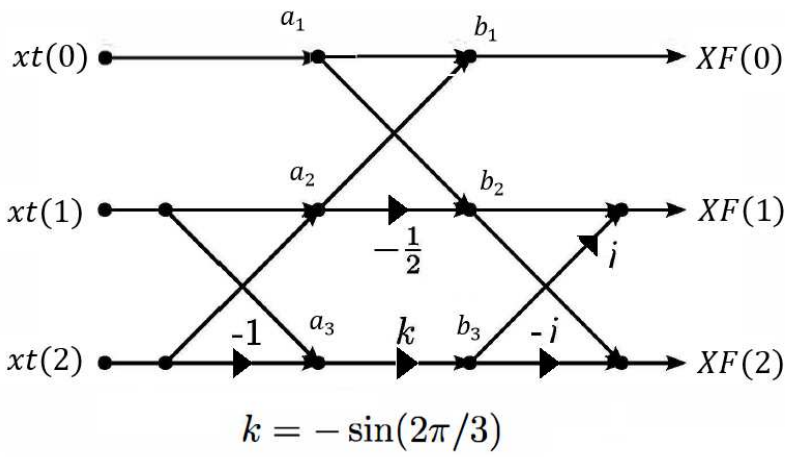

Figura 4.13.- Algoritmo de la FFT base 3.

Para obtener los a_x seguimos las siguientes ecuaciones:

$$a_1 = x(0) \qquad 5.40$$

$$a_2 = x(1) + x(2) \qquad 5.41$$

$$a_3 = x(1) - x(2) \qquad 5.42$$

Con los valores de a_x y realizando las multiplicaciones que marca el algoritmo podemos obtener b_x los cuales son:

$$b_1 = a_1 + a_2 \quad 5.43$$

$$b_2 = a_1 - \frac{1}{2}a_2 \quad 5.44$$

$$b_3 = ka_3 \quad 5.45$$

Finalmente podemos obtener $XF(k)$:

$$XF(0) = b_1 \quad 5.46$$

$$XF(1) = b_2 + ib_3 \quad 5.47$$

$$X(2) = b_2 - ib_3 \quad 5.48$$

Para el caso de la IFFT en el algoritmo base-3 solo se necesita cambiar el signo de los factores i y $-i$.

4.3.2 FFT/IFFT base-5.

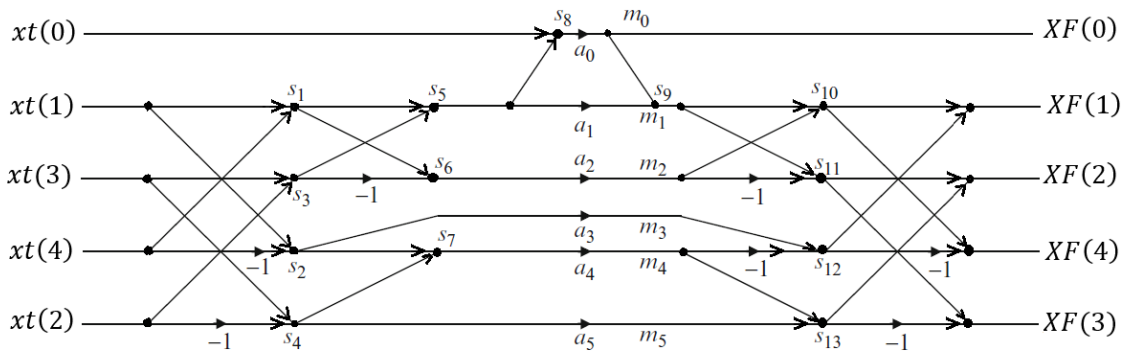

Figura 4.14.- Algoritmo de la FFT base-5 de 5 puntos.

Para realizar el algoritmo base-5 en primer lugar realizamos las primeras sumas que son:

$$S_1 = xt(1) + xt(4) \quad 5.49$$

$$S_2 = xt(1) - xt(4) \quad 5.50$$

$$S_3 = xt(3) + xt(2) \quad 5.51$$

$$S_4 = xt(3) - xt(2) \quad 5.52$$

$$S_5 = xt(1) + xt(3) \quad 5.53$$

$$S_6 = xt(1) - xt(3) \quad 5.54$$

$$S_7 = xt(2) + xt(4) \quad 5.55$$

$$S_8 = S_5 + xt(0) \quad 5.56$$

Ahora necesitamos calcular los valores de a_x para lo cual debemos de saber que $u = \frac{-2\pi}{5}$ entonces tenemos:

$$a_0 = 1 \quad 5.57$$

$$a_1 = \frac{\cos(u)+\cos(2u)}{2} - 1 \quad 5.58$$

$$a_2 = \frac{cos(u)+cos(2u)}{2} \quad 5.59$$

$$a_3 = i(\text{sen}(u) + sen(2u)) \quad 5.60$$

$$a_4 = i\text{sen}(2u) \quad 5.61$$

$$a_2 = i(sen(u) - sen(2u)) \quad 5.62$$

Con los valores de a_x podemos obtener los puntos de multiplicación m_x generando:

$$m_0 = a_0 \cdot S_8 \quad 5.63$$

$$m_1 = a_1 \cdot S_5 \quad 5.64$$

$$m_2 = a_2 \cdot S_6 \quad 5.65$$

$$m_3 = a_3 \cdot S_2 \quad 5.66$$

$$m_4 = a_4 \cdot S_7 \quad 5.67$$

$$m_5 = a_5 \cdot S_4 \quad 5.68$$

Con los valores de m_x podemos realizar el segundo grupo de sumas para lo cual obtenemos:

$$S_9 = m_0 + m_1 \quad 5.69$$

$$S_{10} = S_9 + m_2 \quad 5.70$$

$$s_{11} = s_9 - m_2 \quad 5.71$$

$$S_{12} = m_3 - m_4 \quad 5.72$$

$$S_{13} = m_4 + m_5 \quad 5.73$$

Finalmente podemos obtener $XF(k)$ con las siguientes ecuaciones:

$$XF(0) = m_0 \quad 5.74$$

$$XF(1) = S_{10} + S_{12} \quad 5.75$$

$$XF(2) = S_{11} + S_{13} \quad 5.76$$

$$XF(3) = S_{11} - S_{13} \quad 5.77$$

$$XF(4) = S_{10} - S_{12} \quad 5.78$$

Para el caso del algoritmo base-5 hay 5 multiplicaciones realizadas por los valores a_x para la obtención de la IFFT en este caso solo es necesario cambiar de signo la parte imaginaria de cada uno.

Estos algoritmos aunque resultan ser muy fácil de programar en Matlab generan muchos problemas a la hora de implementarlo en un FPGA por lo cual su utilización es descartada desde este momento, ya que a pesar de que algunos algoritmos hacen uso de estos cuando el número de muestras no es potencia de 2 para el caso particular de nuestro transmisor esto no se cumple para $N = 128$ para lo cual es necesario utilizar algún algoritmo base-2.

Capítulo V:

Desarrollo del Firmware de los bloques básicos.

Ya que para las simulaciones en Matlab se optó por no utilizar funciones preestablecidas y utilizar solo sumas, restas y multiplicaciones resulta fácil trasladar las ideas obtenidas a un diagrama digital. Es importante mencionar que los algoritmos anteriores por lo regular están representados en forma concurrente en la literatura, esto genera una ventaja más a la hora de implementar dichos algoritmos en un FPGA ya que para esto es necesario un lenguaje de descripción de hardware que también trabaja de forma concurrente.

5.1 Especificaciones requeridas.

Se cuenta con un sistema de transmisión OFDM diseñado con anterioridad, el cual es el motivo principal de la realización de esta tesis. Este sistema cuenta con ciertas características que deben de respetarse al momento de diseñar la FFT e IFFT ya que nuestro algoritmo deberá ser compatible con la señal generada por el transmisor.

Las características que debe cumplir nuestro módulo FFT e IFFT son las siguientes:

- La FFT será de 128 puntos ya que el estándar usado por el transmisor establece que 96 subportadoras son de datos, 10 subportadoras son pilotos y hay 22 subportadoras nulas.
- La extensión de la palabra es de 20 bits siendo el bit más significativo el bit de signo.
- Un dato está integrado por dos palabras donde una representa la parte real y otra la parte imaginaria.
- Los datos serán ingresados de forma serial y serán entregados de forma serial.

Tomando en cuenta estas características podemos afirmar que el módulo de la FFT/IFFT será capaz de trabajar con el transmisor previamente diseñado y con el receptor que será diseñado más adelante.

Los datos están integrados por 20 bits y están distribuidos como se ilustra en la Figura 5.1:

Figura 5.1.- Formato de la distribución de bits de la palabra a utilizar.

Esta asignación es abstracta debido a que el sistema lo tratara como un número entero, esto facilitara en gran medida el procesamiento de los datos y sobre todo reducirá los recursos necesarios para implementar este algoritmo que si se emplearan otras técnicas como la representación de números por punto flotante que elevan en gran medida el número de elementos lógicos necesarios para realizar cualquier operación básica sobre todo para la multiplicación.

Para representar un número en el formato requerido es necesario aplicar la siguiente formula:

$$Num_2 = Num_{10} * 1024 \qquad 6.1$$

Este escalamiento tiene su origen en el hecho de que es necesario recorrer el punto decimal, debido a que se desea que la representación quede solo en números enteros binarios para facilitar las operaciones realizadas.

Cuando sea necesario un número negativo será representado con su complemento a dos, ya que así es como trabajan los elementos básicos de suma, resta y multiplicación.

Esta distribución de bits nos permite tener las siguientes características:

Podremos representar números en el rango de $-2^{19} a\ (2^{19} - 1)$ en binario, si deseamos saber su representación en decimal seria de $-\frac{2^{19}}{1024}\ a\ \frac{2^{19}-1}{1024}$. La precisión que se puede alcanzar con esta representación es de $\frac{1}{1024}$.

Es muy importante tomar en cuenta precisión de lo contrario no se puede garantizar la reversibilidad de la transformada. Por otro lado es necesario remarcar que un número binario siempre tendrá cierto error debido al truncamiento y redondeo, ya que no es posible en ciertos casos representar con exactitud un número decimal fraccionario lo que hace necesario representar dicho número en el binario más cercano generando un error.

La estrategia de diseño que se utilizo fue bottom-up (de abajo a arriba) donde las partes individuales se diseñan con detalle y luego se enlazan para formar componentes más grandes, que a su vez se enlazan hasta que se forma el sistema completo.

Para el desarrollo de los bloques básicos se utilizó la herramienta MegaWizard Plug-Ins, que consiste en una herramienta de parametrización que ayudan a integrar megafunciónes en los diseños sin requerir de terceros. Se puede utilizar esta función en el Quartus II y MAX + PLUS II (versión 8.2 y superior) proporciona la máxima flexibilidad, que permite personalizar megafunciónes sin cambiar el código fuente del diseño. Se puede integrar una megafunción parametrizada en cualquier HDL [13].

Como se ilustra en la Figura 5.2 los bloques que se diseñaron con esta herramienta son los siguientes:

- Sumador de 20 bits con signo.
- Restados de 20 bits con signo.
- Multiplicador de 20 bits con signo.

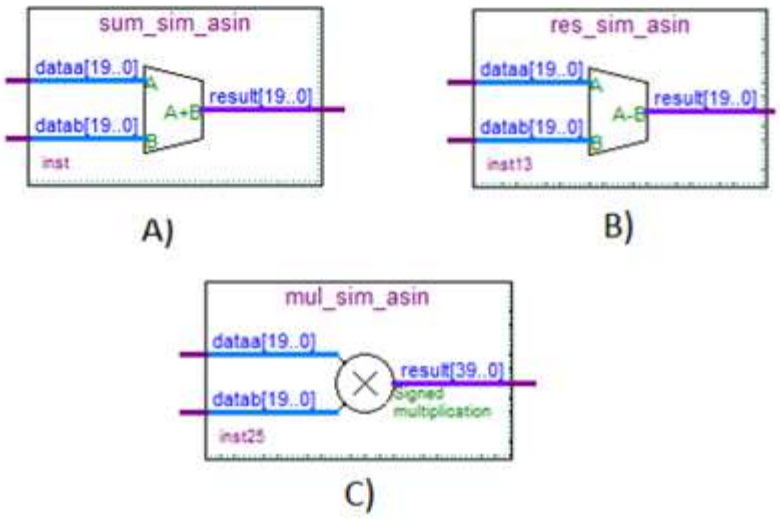

Figura 5.2.- Módulos creados con MegaWizard Plug-Ins: A) Sumador de 20 bits, B) Restador de 20 bits y C) Multiplicador de 20 bits.

Para el correcto funcionamiento de los bloques se debe tomar en cuenta que el bit más significativo representa el signo, 0 indica que el número es positivo y 1 que es negativo. Adicionalmente los números negativos están representados en complemento a dos. Para el multiplicador de 20 bits sabemos que dos números de 20 bits generan un resultado de 40 bits por lo cual más adelante será necesario generar un bloque que adapte este dato a nuestro estándar utilizado.

5.2.1 Construcción del firmware de los algoritmos.

Antes de la construcción de los algoritmos base es necesario diseñar un módulo más, que es un multiplicador de dos números complejos. Si multiplicamos dos números complejos obtenemos:

$$(a + ib)(c + id) = (ac - db) + i(ad + bc) \qquad 6.2$$

Como se observa se puede realizar con 4 multiplicaciones, una suma y una resta. Dado que la parta real e imaginaria del dato a procesar están separadas, solo es cuestión de entrelazar correctamente los bloques para poder obtener el resultado correcto como se muestra en la Figura 5.3.

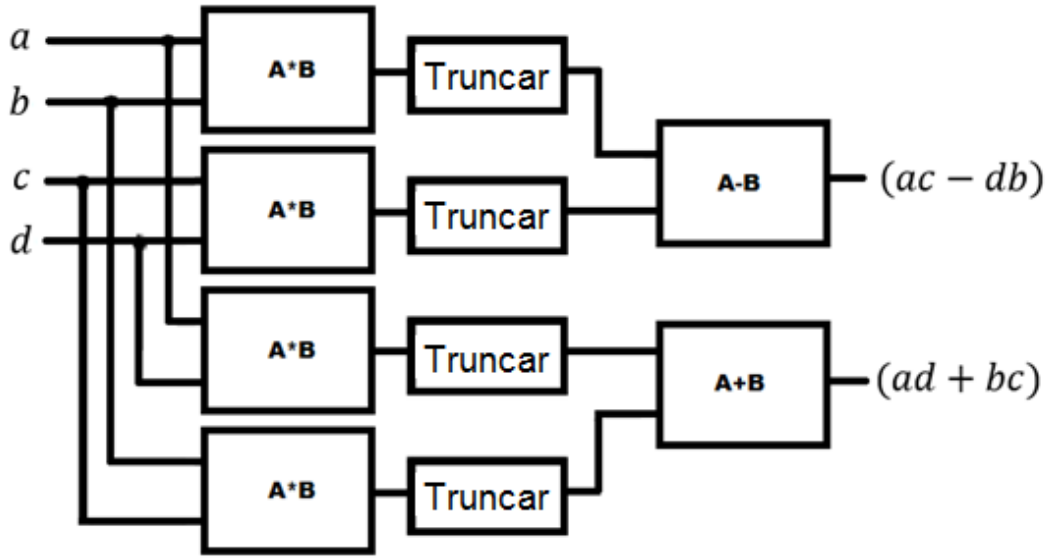

Figura 5.3.- Diagrama a bloques del multiplicador de números complejos que trabaja en paralelo.

Sin embargo un inconveniente es que la multiplicación genera un número de 40 bits pero el formato de nuestra palabra debe ser de 20 bits. Por lo tanto debemos de realizar un truncamiento, esta operación es la que agregara mayor error a nuestro diseño ya que acorta la exactitud de nuestro dato. El truncamiento se realizara eliminando los 10 bits más significativos ya que no se tendrán valores tan grandes y los 10 bits menos significativos debido a que valores tan pequeños están fuera de la resolución mínima que requerimos.

Si observamos el diagrama de la Figura 5.3 un solo multiplicador complejo implementado de esta forma necesita 4 multiplicadores de 20 bits, 4 módulos de truncamiento, un sumador de 20 bits y un restador de 20 bits.

Hay que poner especial atención en el número de multiplicadores necesarios ya que el FPGA tiene una cantidad limitada de multiplicadores 9x9 lo cual solo nos permite implementar 3 multiplicadores de 20x20. Aunque es posible implementar multiplicadores con elementos lógicos esto conlleva utilizar el 10% de los elementos lógicos disponibles por cada multiplicador, realizarlo de esta forma obliga a utilizar recursos en exceso que hace imposible la implementación completa de la FFT en el FPGA de la tarjeta hija con la que se cuenta. Por ello se plantearan varias ideas que ayudaran a reducir las operaciones necesarias con el fin de reducir los recursos necesarios y lograr usar la tarjeta hija. Estas ideas solo pueden hacerse presentes a la hora de implementarlo en un FPGA debido a las diferencias existentes entre un lenguaje de programación que trabaja secuencialmente y un HDL que trabaja concurrentemente.

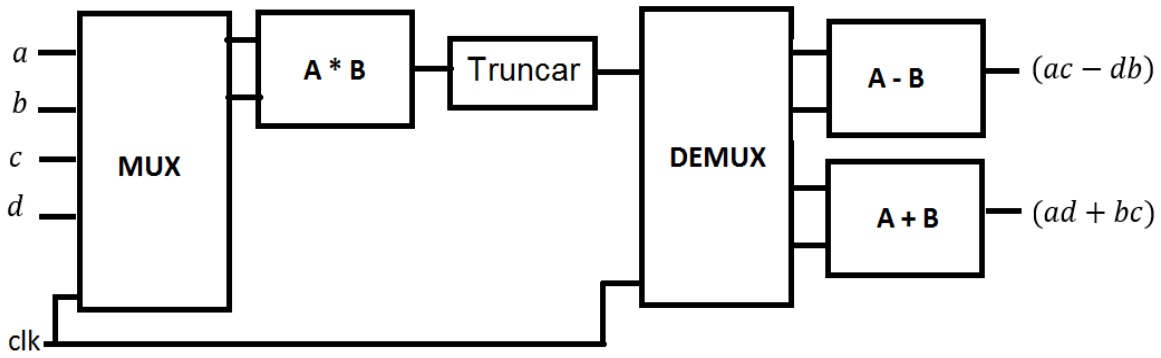

Figura 5.4.- Diagrama a bloques del multiplicador de números complejo que trabaja en forma multiplexada.

Se utilizó una técnica de multiplexación con el fin de poder realizar la multiplicación compleja con un solo multiplicador y un solo módulo de truncamiento esto es posible ya que se utiliza el mismo multiplicador en diferentes lapsos de tiempo, por lo cual

el firmware de la Figura 5.3 y el de la Figura 5.4 entregan el mismo resultado, con la diferencia que el primero lo realiza en un ciclo de reloj, mientras que el segundo toma 10 ciclos, por lo cual se puede observar que a cambio de reducir el número de recursos necesarios para realizar la multiplicación compleja se incrementa el tiempo en el que esta es realizada.

Aquí empiezan a surgir ciertas características que es necesario tomar en cuenta para poder utilizar la técnica de multiplexado, las más sobresalientes son la utilización de una señal de reloj que es necesaria para poder sincronizar los diferentes bloques y la necesidad de utilizar registros para poder utilizar los datos en diferentes momentos sin que se pierdan.

5.2.2 Los algoritmos base.

Empezamos por descartar el uso de un algoritmo base-2 de 2 puntos con factor de rotación ya que al tener por separado la FFT base-2 y la multiplicación del factor de rotación, nos dará mayor control sobre el número de multiplicaciones necesarias para la obtención de la FFT, como se observara en los algoritmos de 8 y 16 puntos. Por lo cual solo se utilizara el firmware de la Figura 5.5 que como se puede observar solo consta de dos sumadores y dos restadores de 20 bits.

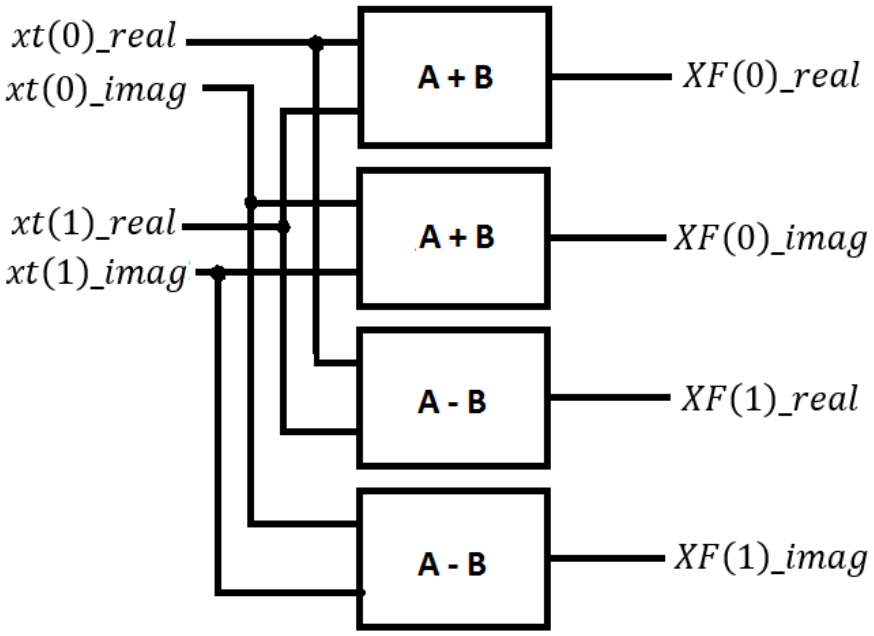

Figura 5.5.- Diagrama a bloques del firmware del algoritmo FFT base-2.

Para el firmware de la FFT base-4, según lo marca el algoritmo de la Figura 4.4 se tendría que realizar una multiplicación compleja por $-i$ en el caso de la FFT e i para IFFT, pero este es un caso especial que nos permite implementar la FFT/IFFT sin la necesidad de utilizar ni una sola multiplicación, como se observó mientas realizamos el desarrollo del algoritmo de la FFT base-4 de 4 puntos.

Se estableció a partir de las ecuaciones 5.15 a 5.22 que $XF(0) = invXF(0)$ y $XF(2) = invXF(2)$, para que $XF(1)$ sea igual a $invXF(1)$ solo hay que intercambiar h por d y c por g, y finalmente para $X(3)$ sea igual a $invXF(3)$ hay que intercambiar d por h y c por g.

Las multiplicaciones por i y $-i$ pueden ser evitadas y sustituidas por multiplexores con lo cual se obtiene el resultado correcto con la gran ventaja de no utilizar ni una sola multiplicación lo cual es uno de los objetivos principales. Este firmware será de gran relevancia para poder realizar los módulos de los algoritmos de 8 y 16 puntos con las ventajas que conlleva el no utilizar ninguna multiplicación para la FFT/IFFT de 2 y 4 puntos.

En la Figura 5.6 se puede observar que no ha sido necesario ningún bloque de multiplicación a cambio se han agregado multiplexores que con una señal de control llamada FFT_IFFT nos indica cual es la tarea deseada, ya sea la FFT si esta en bajo o si se desea realizar la IFFT para lo cual debe permanecer en alto.

Teniendo listo los algoritmos base podremos realizar las FFT/IFFT's de un mayor número de muestras que como se ha mostrado en el capítulo III pueden dividirse en módulos de FFT base-2 o base-4, de ahí la importancia de haber implementado estos bloques sin la necesidad de utilizar multiplicadores ya que podemos utilizar la cantidad necesaria de estos bloques sin consumir ni uno de los tres multiplicadores disponibles en el FPGA.

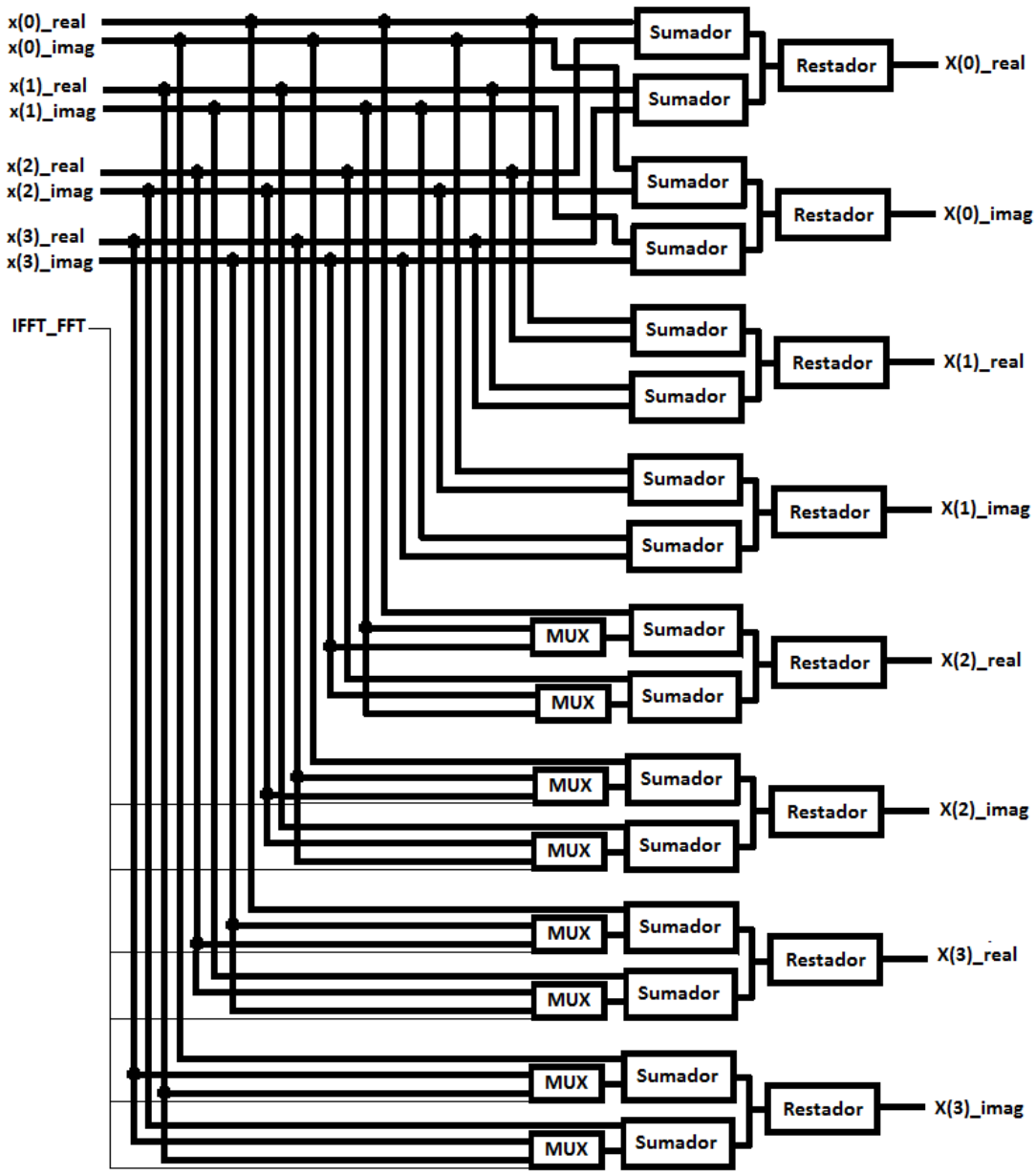

Figura 5.6.- Diagrama a bloques del firmware del algoritmo FFT/IFFT base-4.

5.2.3 Algoritmos base 2 y 4 para FFT de 8 y 16 muestras.

Una vez obtenido el firmware de los algoritmos base-2 y base-4 podemos usarlo para poder realizar el algoritmo de 8 puntos de la Figura 5.7. La teoría dice que una FFT donde N puede expresarse como una potencia de 2 puede dividirse en dos FFT's de tamaño $N/2$ donde se agrupan las muestras impares y las pares para las

FFT's de menor tamaño. Siguiendo esta metodología para la transformada de 8 muestras, se separaran las muestras pares y las impares en dos grupos a las cuales se les aplicara el algoritmo base-2 de 2 puntos y al resultado se le trata como una FFT de 4 puntos, con las cuales utilizaremos las funciones obtenidas con anterioridad. Esta es otra forma de ver el mismo procedimiento ya realizado donde la secuencia $x(n)$ era reorganizada como una matriz.

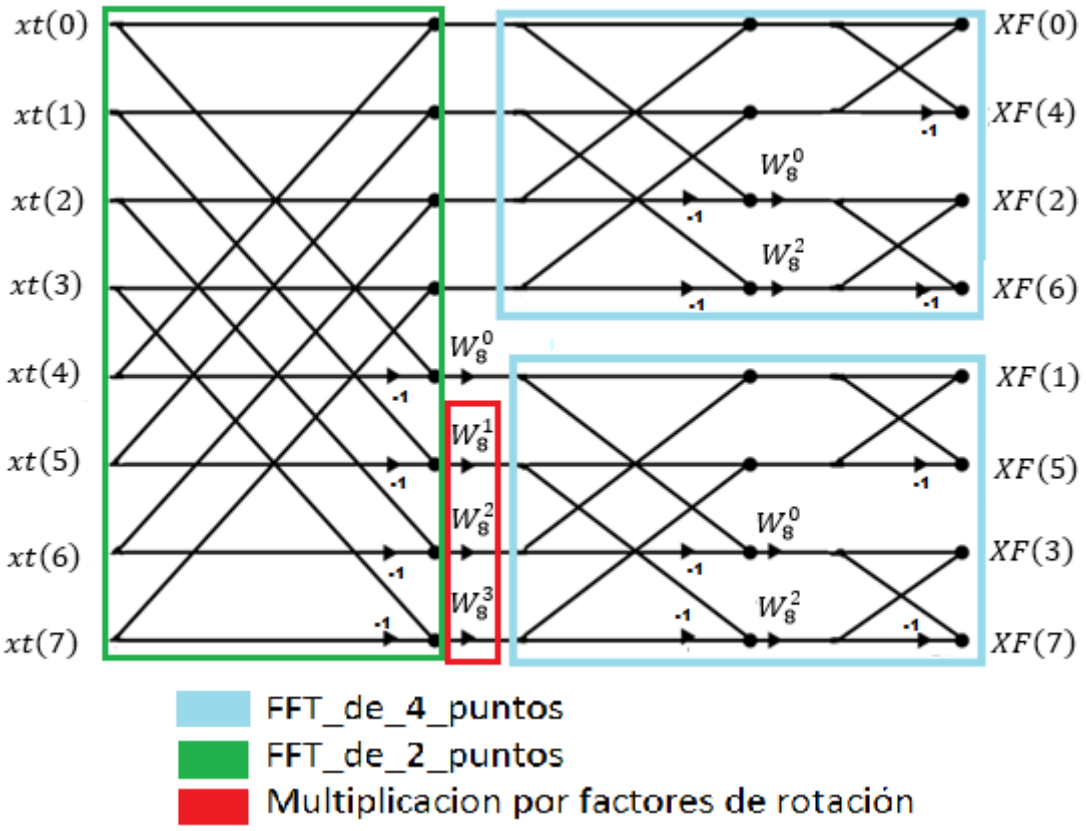

Figura 5.7.- Separación en secciones del algoritmo FFT/IFFT para N=8 base-2 y base-4.

Como podemos ver separamos el algoritmo de la Figura 5.7 en tres partes, en verde se realizan cuatro FFT's de 2 muestras, en cian están marcadas 2 FFT's de 4 muestras y en rojo los factores de rotación W_8^0, W_8^1, W_8^2, W_8^3 por lo cual será necesario realizar cuatro multiplicaciones complejas. El valor correspondiente de W_8^k podemos obtenerlo de la ecuación 3.10.

Como se observa si $k = 0$ entonces $W_8^0 = 1$ por lo cual no será necesaria realizar esta multiplicación. Otro punto importante a notar es que los valores de W siempre estarán localizados sobre el círculo unitario por lo tantos las multiplicaciones siempre serán por un número menor o igual a 1.

Para realizar la IFFT de 8 muestras se utiliza el mismo algoritmo y por lo tanto la misma metodología antes descrita, la única diferencia radicara en los factores de rotación que ahora serán obtenidos con la ecuación 5.12.

Esta diferencia entre las ecuaciones 3.10 y 5.12 se refleja como un cambio de signo en la parte imaginaria de los factores de rotación, por lo tanto, si se desea realizar la FFT o la IFFT solo se tendrá que realizar un cambio de signo en la parte imaginaria de cada factor de rotación.

En la Figura 5.7 se muestra el algoritmo para obtener la FFT/IFFT de 8 puntos que se ha dividido en tres partes, en verde se muestran las cuatro FFT de 2 puntos necesarias para la primera etapa, en rojo la segunda etapa, correspondiente a las multiplicaciones complejas necesarias por los factores de rotación que en este caso son 3 y finalmente se observa que en la tercera etapa se realizan dos FFT de 4 puntos. Utilizando la técnica de multiplexación podemos implementar este algoritmo utilizando solo un módulo FFT base-2, un multiplicador complejo y un módulo FFT/IFFT base-4 como se muestra en la Figura 5.8.

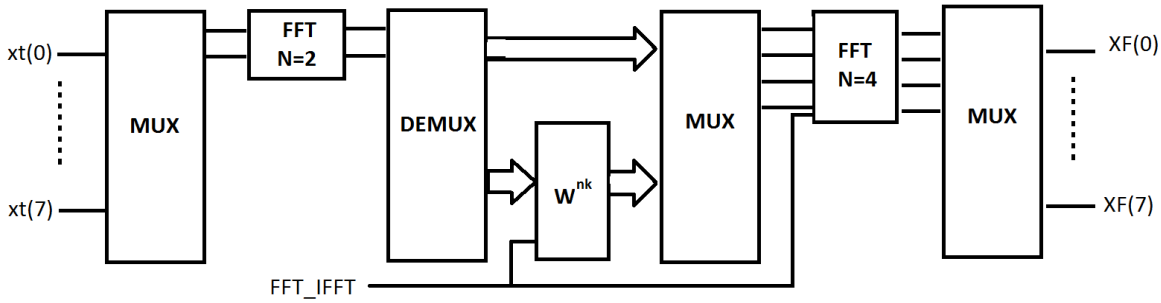

Figura 5.8.- Diagrama a bloques del firmware del algoritmo FFT/IFFT para N=8 base-2 y base-4.

Con esta técnica se ha reducido en gran medida los recursos necesarios para la implementación de la FFT/IFFT para $N = 8$ en comparación con una versión que trabajaba en paralelo, solo fue necesario un solo multiplicador complejo el cual es usado en diferentes lapsos de tiempo para realizar las tres multiplicaciones complejas necesarias para los factores de rotación.

El cambio entre los factores de rotación necesarios para realizar la FFT o la IFFT se realiza a partir de una señal llamada FFT_IFFT con la cual realiza la FFT si esta en bajo y la IFFT si esta en alto.

Para obtener la FFT/IFFT de 16 puntos, no desarrollamos el algoritmo base-2 que como se vio en el capítulo IV requería generar un matriz de 2x8, ya que este algoritmo implica la utilización de un mayor número de bloques básicos y más

etapas para el procesamiento, en su lugar utilizamos el algoritmo base-4 de 16 puntos de la Figura 5.9 debido a que este se realiza en menos etapas, lo cual nos permite dividirlo en tres partes como en el caso de la FFT/IFFT de 8 muestras.

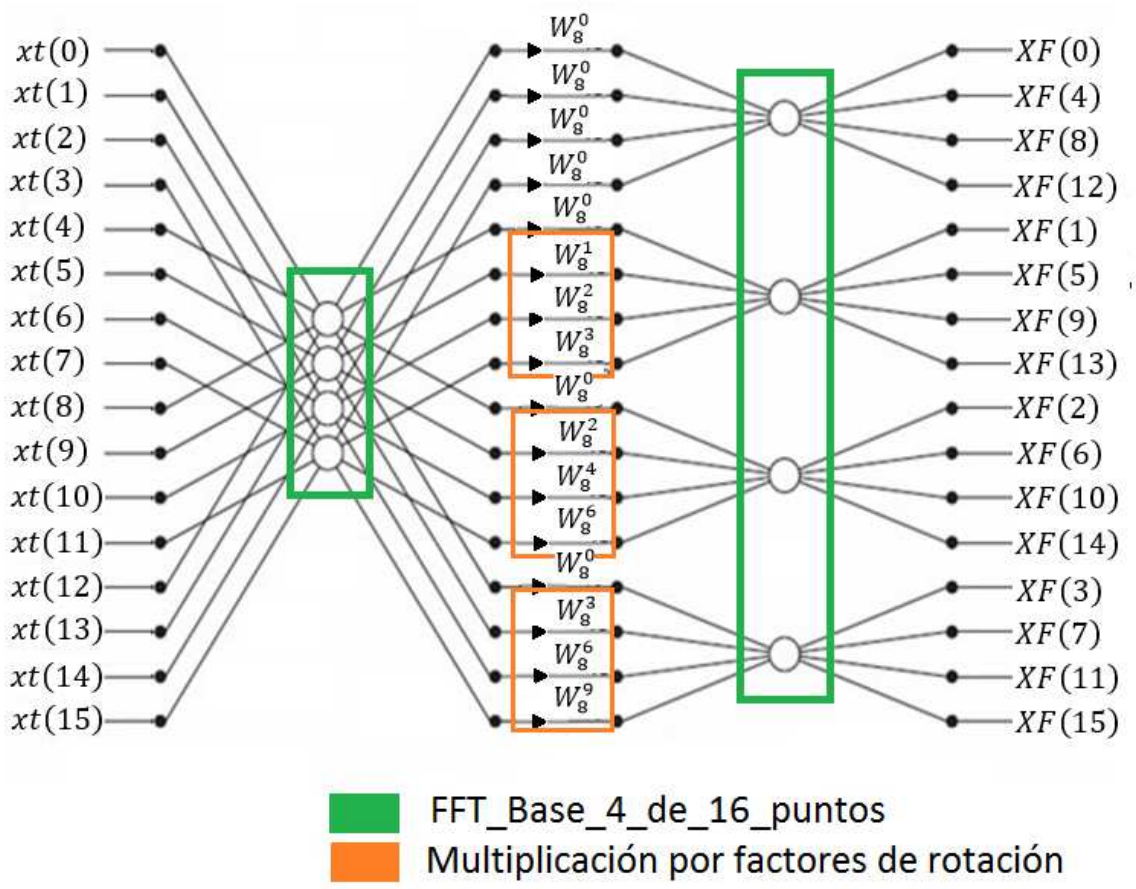

Figura 5.9.- Separación en secciones del algoritmo FFT/IFFT para N= 16 base-4.

En verde se muestran las cuatro FFT/IFFT de 4 puntos necesarias para la primera etapa, en naranja se marca la segunda etapa correspondiente a las multiplicaciones complejas que en este caso son 9 y finalmente se observa que en la tercera etapa nuevamente se realizan 4 FFT's de 4 puntos, por lo cual, la primera y tercera etapa puede realizarse con el mismo módulo FFT/IFFT de 4 muestras, ya que solo difieren en el orden en que se ingresan los datos, cosa que pude ser fácilmente implementado en los multiplexores necesarios.

Utilizando la técnica de multiplexación todo el algoritmo de 16 puntos puede ser implementado con un solo módulo FFT/IFFT de 4 puntos y un solo multiplicador complejo, como se muestra en el diagrama a bloques de la Figura 5.10. Se observa que usar solo un módulo FFT/IFFT de 4 puntos complica el firmware haciendo

necesario utilizar más multiplexores y registros, el aumento en el uso de estos recursos no impacta de forma significativa en nuestro diseño ya que los bloques de memoria son de los menos utilizados y más abundantes en el FPGA.

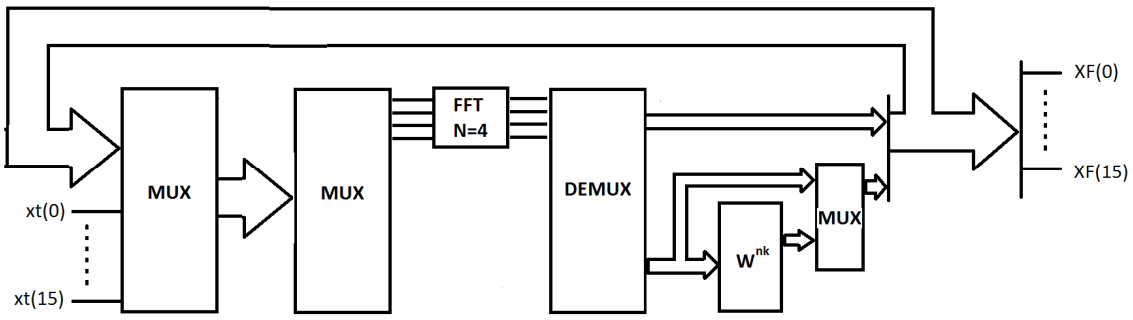

Figura 5.10.- Diagrama a bloques del firmware del algoritmo FFT/IFFT para N=16 base-4.

Con los bloques de la FFT/IFFT de 8 y 16 puntos es posible realizar la FFT/IFFT de 128 muestras, como se mostró en las simulaciones de Matlab, además de estos bloques es necesario diseñar otros módulos necesarios para ordenar correctamente los datos según se requiera, cosa que se mostrara en el siguiente capítulo.

5.3 Firmware de los Algoritmos de Factores Primos.

Los algoritmo de factores primos tienen una metodología diferente como ya se ha mencionado, por lo cual tienen ciertas diferencias en su estructura, ya que en estos algoritmos se intenta reducir al mínimo el número de multiplicaciones requeridas para realizar la FFT, una característica importante que deben de cumplir es la ecuación $N = (a)(b)$ que dice que la FFT de N muestras puede ser obtenida a partir de dos FFT, una de tamaño a y otra de tamaño b, donde a y b deben de ser números primos entre sí, ya que de lo contrario estaríamos ante un algoritmo del tipo Cooley-Tukey. La metodología para utilizar este tipo de algoritmos para una FFT de mayor número de muestras es un tanto diferente a lo seguido en los de tipo Cooley-Tukey por lo cual solo se limitara en este caso a los algoritmos base.

Como se observa estos algoritmos no necesitan de multiplicaciones complejas por lo cual se usan multiplicadores simples.

Aunque algunos de estos algoritmos son usados en ocasiones donde el número de muestras no es una potencia de 2, existen ciertas técnicas que pueden utilizarse que permiten el uso de los algoritmos tipo Cooley-Tukey, sin agregar gran complejidad al diseño original cosa que puede compensar en sobre medida, si es que se desea utilizar algún algoritmo de factor primo.

Capítulo VI:

Firmware de la FFT/IFFT de 128 muestras.

Además de los bloques realizados con anterioridad son necesarios otros módulos para implementar la transformada de 128 puntos estos tienen que ver con la organización de los datos de acuerdo a la metodología que se ha ido utilizando para FFT's de un número mayor de puntos al de los algoritmos bases.

El firmware final lo podremos dividir en seis etapas correspondientes a la metodología elegida para realizar la FFT/IFFT de 128 muestras las cuales se describen con más detalles a continuación.

6.1 Implementación del firmware de la FFT/IFFT de 128 muestras.

Las entradas de nuestro modulo son una entrada de reloj de 100 MHz, una señal que restablece todos los contadores y registros y una entrada de 20 bits correspondiente a la entrada serial de nuestro sistema.

Siguiendo la metodología marcada en la Figura 3.8 podemos dividir nuestro modulo en seis etapas encargadas de realizar tareas muy específicas para poder realizar la FFT/IFFT de 128 muestras cada una de estas etapas es descrita con detalle.

En la **primera etapa** se necesita organizar a partir de una entrada serial de 20 bits una matriz de datos de 16 filas y 8 columnas en el orden que se indica en la Figura 4.12, este arreglo se escogió sobre el de 8 filas y 16 columnas por la ventaja que conlleva realizar en primer lugar las FFT/IFFT's de 8 puntos en lugar de la segunda opción que implica realizar primero las de 16 puntos.

Para realizar el almacenamiento de los datos en el orden deseado se utilizaron 16 memorias FIFO (First Input, First Output / Primera Entrada, Primera Salida) de 20 bits x 16 palabras esto nos permite controlar el orden de como los datos se irán entregando, el cual no corresponde al orden de entrada de los datos en la entrada serial ya que así lo pide el método elegido para realizar la FFT/IFFT de 128 muestras. En la Figura 6.1 se muestra que este arreglo entrega 8 salidas correspondiente a las entradas necesarias para poder realizar las FFT/IFFT de 8 puntos correspondientes a cada fila de la matriz de 16x8, con la ayuda de las mismas memorias estos datos se mantienen el tiempo necesario que necesite la segunda etapa, tercera etapa y cuarta etapa.

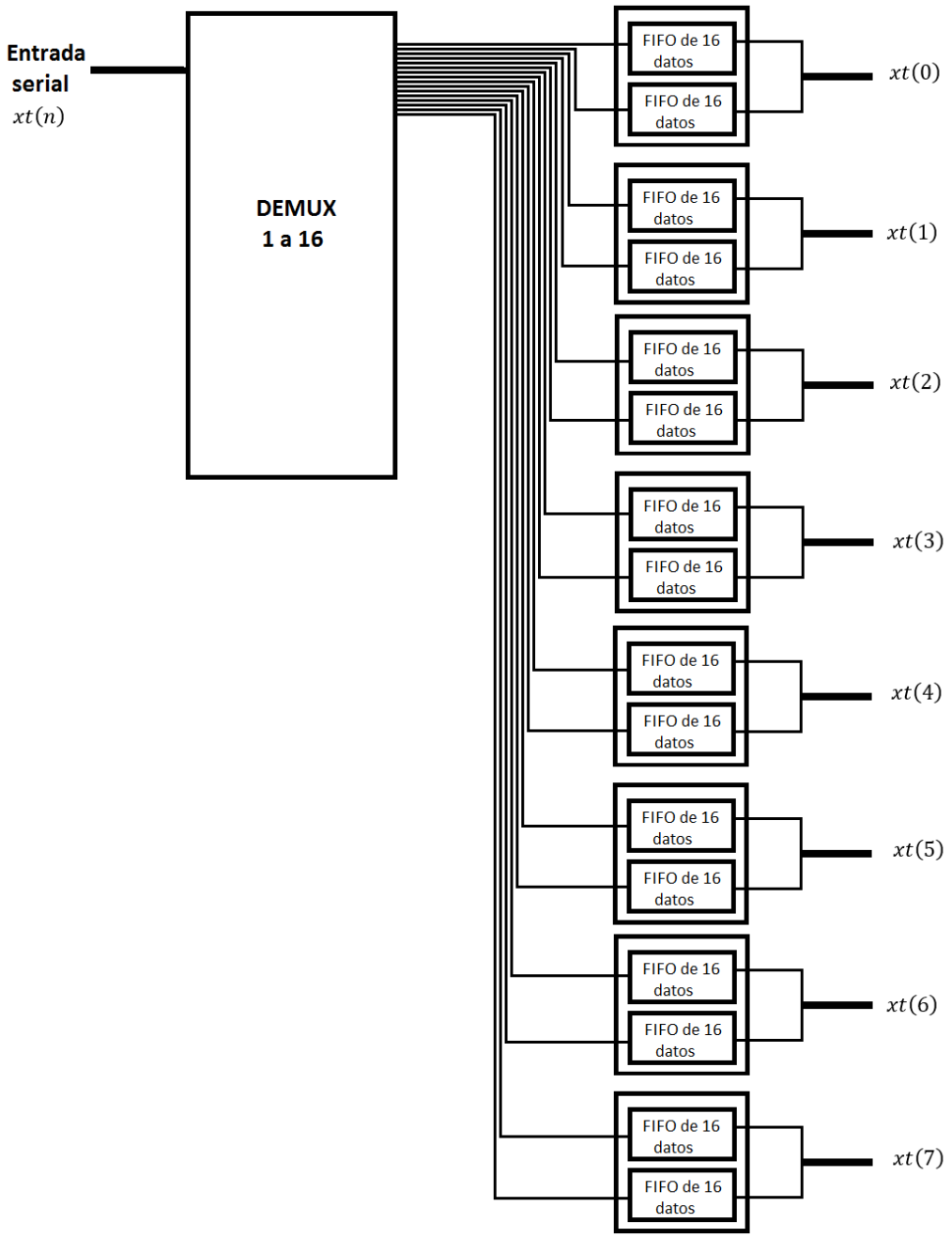

Figura 6.1.- Diagrama a bloques del arreglo de memorias FIFO correspondiente a la primera etapa del firmware FFT/IFFT de 128 muestras.

En la **segunda etapa** se realizan 16 FFT/IFFT's de 8 muestras por lo que se utilizó el bloque previamente diseñado, la primera etapa es el encargado de entregarnos los datos correspondientes y mantenerlos, por lo que no es necesario hacer ninguna

modificación a este bloque más que el de conectar correctamente las entradas necesarias.

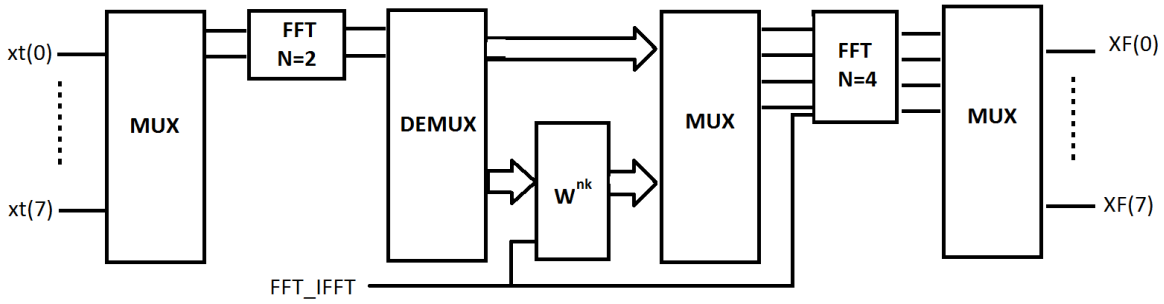

Figura 6.2.- Diagrama a bloques del firmware del algoritmo FFT/IFFT para N=8 correspondiente a la segunda etapa.

La **tercera etapa** corresponde a las multiplicaciones complejas necesarias entre la matriz generada por la primera etapa y la matriz de los factores de rotación, como se ha realizado en el capítulo III, esta es la principal razón por la cual escogimos realizar primero las FFT/IFFT's de 8 puntos, esto conlleva realizar 16 veces 7 multiplicaciones complejas que de haber realizado primero las FFT/IFFT's de 16 muestras esto implicaba hacer 8 veces 15 multiplicaciones complejas, donde este segundo caso implica una mayor complejidad y aunque se realice en menos ocasiones, el diseño de este módulo sería mucho más complejo.

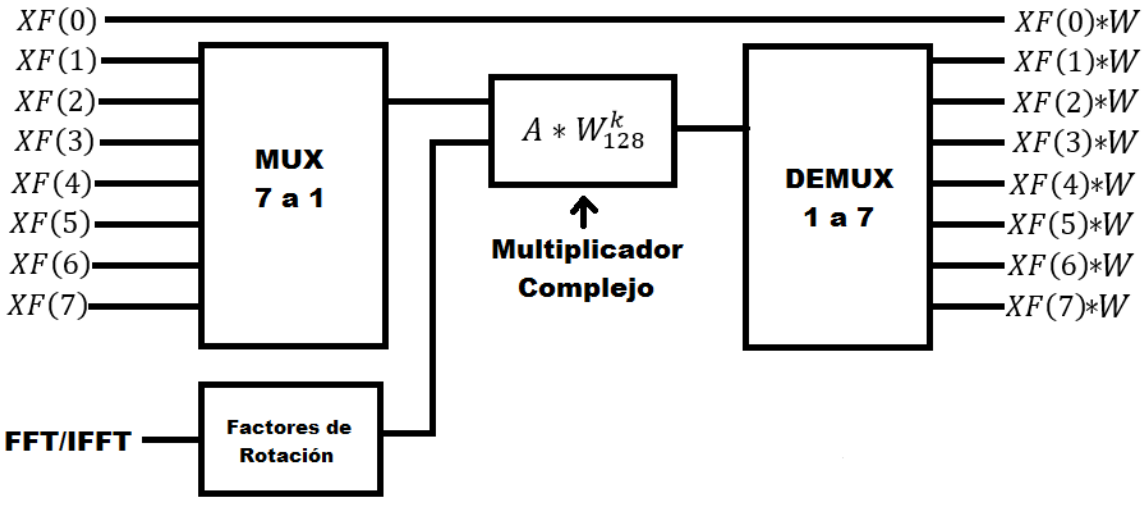

Figura 6.3.- Multiplicador de los factores de rotación correspondiente a la tercera etapa.

Para realizar las multiplicaciones complejas utilizaremos un multiplicador de 20 bits y haciendo el recuento en total se han utilizados tres, uno para la FFT/IFFT de 8 muestras, uno para la FFT/IFFT de 16 muestras y finalmente uno para la matriz de factores de rotación con esto hemos utilizado los tres multiplicadores disponibles en el FPGA.

La **cuarta etapa** es nuevamente un arreglo de memorias FIFO donde se tiene que alterar el orden de los datos ya que este no coincide con el necesario para poder realizar la siguiente etapa, además, esta vez se tienen que entregar 16 datos en paralelo como se muestras en la Figura 6.2.

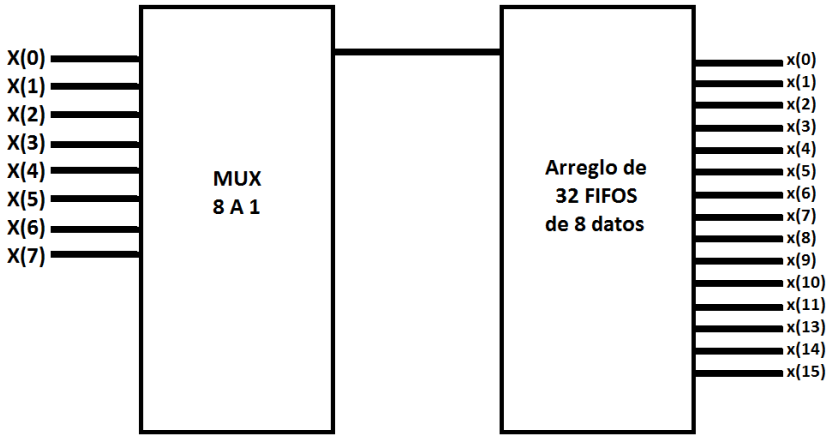

Figura 6.4.- Diagrama a bloques del arreglo de memorias FIFO correspondiente a la cuarta etapa del firmware del algoritmo FFT/IFFT de 128 muestras.

La **quinta etapa** corresponde a la realización de 8 FFT/IFFT de 16 muestras por lo cual solo es necesario utilizar el bloque ya antes diseñado.

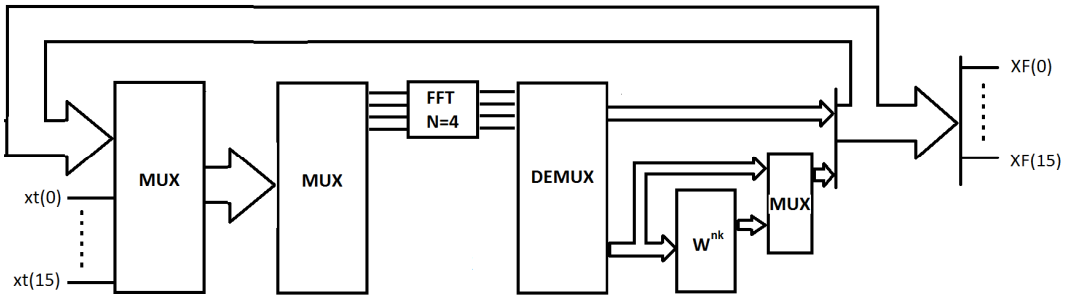

Figura 6.5.- Diagrama a bloques del firmware del algoritmo FFT/IFFT para N=16 correspondiente a la quinta etapa.

Finalmente la **sexta etapa** corresponde al arreglo de FIFO's de la Figura 6.6 que almacenan los datos entregados por el módulo FFT/IFFT de 16 muestras, también tiene que ordenar los datos en su secuencia final para entregar la FFT/IFFT de 128 muestras, todos estos datos almacenados tiene que ser entregados de forma serial.

Este arreglo es el mismo que en la primera etapa y podría reutilizarse, pero esto elevaría notablemente la complejidad de firmware, ya que los bloques de memoria son los recursos más abundantes y el menos usado, podemos implementar por separado otro arreglo de memorias FIFO's, esto nos facilitara la conversión de paralelo a serial para entregar el resultado final.

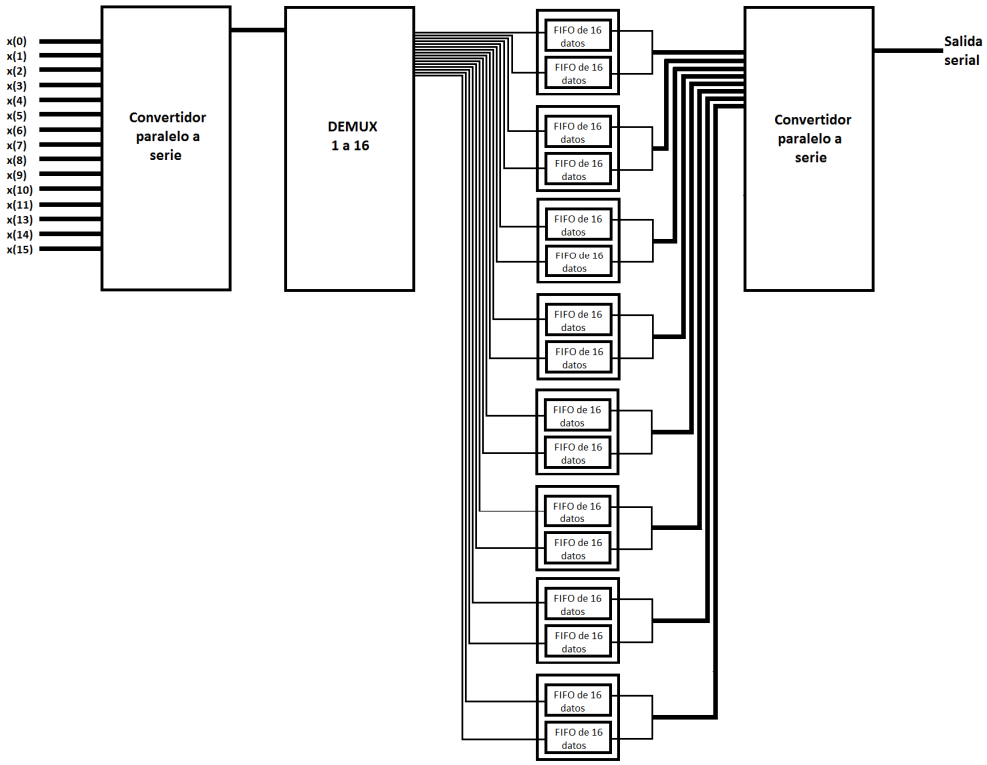

Figura 6.6.- Diagrama a bloques del arreglo de memorias FIFO correspondiente a la sexta etapa el firmware del algoritmo FFT/IFFT de 128 muestras.

Se agregó una etapa extra a la de la metodología escogida encargada de realizar el factor de escalamiento $1/N$ que es requerido para la IFFT solamente, N representa el número de muestras a procesar que en este caso es de 128, pero este número lo podemos representar como 2^7, ya que estamos trabajando con números binarios, sabemos que una división entre un factor que sea potencia de dos se puede realizar con un corrimiento a la derecha, por lo cual esta última etapa se

encarga de recorrer a la derecha siete veces el resultado si es que se está realizando la IFFT de lo contrario no realiza ninguna acción.

El diagrama general de las 7 etapas conjuntas se muestra en la Figura 6.7 con las 7 etapas interconectadas necesarias para realizar la FFT/IFFT de 128 muestras, como se puede observar todas las etapas cumplen una tarea específica de la metodología seleccionada, tres de las seis etapas están destinadas al almacenamiento y reordenamiento de los datos cosa muy importante ya que de ello depende obtener o no un resultado correcto.

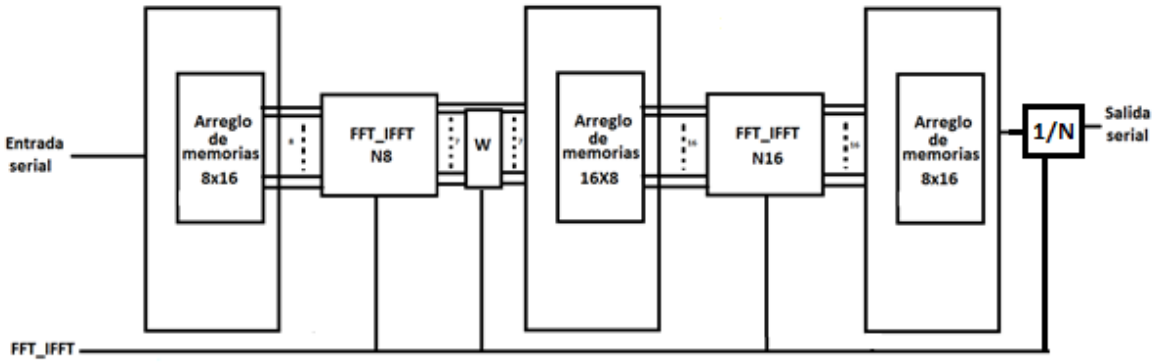

Figura 6.7.- Diagrama general a bloques de la FFT/IFFT de 128 muestras.

Para conocer si el firmware implementado funciona correctamente se realizaron diferentes simulaciones y pruebas una vez implementado en el FPGA sobre todo para los módulos que realizaban las diferentes FFT/IFFT's de diferentes número de muestras ya que en ellos recaía gran parte del procesamiento y sobre todo los módulos de truncamiento de la multiplicación eran los que más error podrían generar, si este error excedía cierto límite no se podía garantizar que una vez realizada la FFT se pudiera regresar a los datos originales aplicando la IFFT y por lo tanto el modulo no cumpliría con las características necesarias para su utilización en el sistema de comunicaciones.

6.2 Simulaciones del Firmware.

Las simulaciones están hechas para observar en primer lugar si el firmware realiza la tarea deseada, que es obtener la FFT de una secuencia de números obtenidos aleatoriamente con la ayuda de Matlab con la función rand, ya que así se desea poder observar errores particulares o excepciones en donde no se realice correctamente la FFT/IFFT.

Por otra parte se desea observar si nuestro diseño es capaz de realizar la FFT y la IFFT utilizando la misma estructura, que como lo marca la teoría con la diferencia de variar los factores de rotación según 3.7 y 3.8 según los valores requeridos para cada caso dependiendo de una señal de control nombrada IFFT_FFT.

La prueba más importante en estas simulaciones es conocer cuál es la diferencia entre el dato original y el dato obtenido una vez se le haya aplicado la FFT y luego la IFFT, ya que por motivos de truncamiento y redondeo en los factores de rotación y en las multiplicaciones se generan ciertos errores que deben ser medidos para establecer si este error es relevante o puede ser despreciable.

Él esquema de simulación se presenta en la Figura 6.8 como podemos ver existen dos módulos FFT/IFFT con la diferencia en la señal de control IFFT_FFT que esta invertida con lo cual el proceso del primer bloque es invertido, con esto se obtiene aproximadamente la secuencia de entrada.

Con el restador podemos ver la diferencia entre el dato original y el de salida.

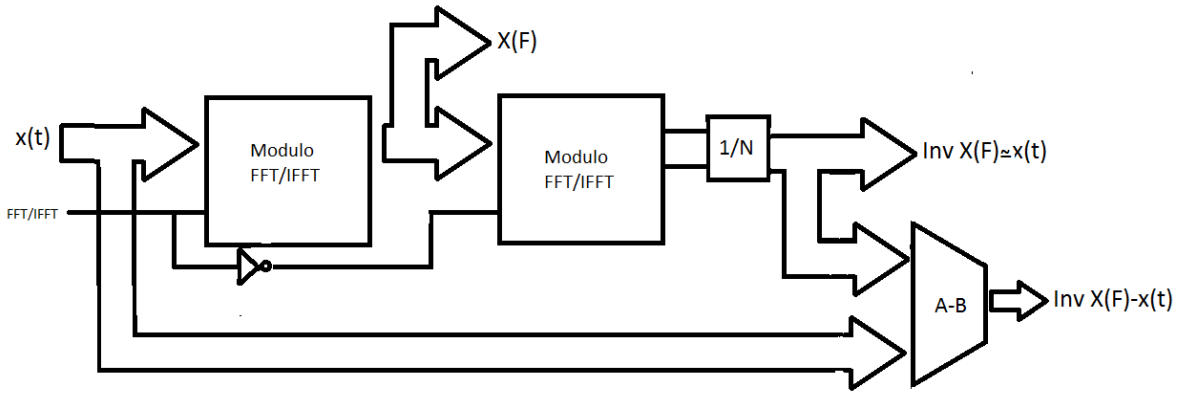

Figura 6.8.- Esquema de las simulaciones realizadas en Matlab y Quartus II.

Una vez programado el FPGA se siguió un proceso muy similar para comprobar que el firmware realizara la tarea correcta y que el error generado estuviera dentro de la tolerancia requerida.

Capítulo VII:

Resultados.

Generamos el firmware para 4 algoritmos que se implementaron en un FPGA que son la FFT/IFFT base-2, base-4, base-8 y base-16 con la ayuda de estos y usando la metodología presentada se pueda obtener la FFT/IFFT de 128 muestras que son el número de datos necesarios para el transmisor diseñado con anterioridad.

7.1 Prueba de los bloques de los algoritmos bases FFT/IFFT.

Cada bloque de las FFT/IFFT de diferente número de muestras tiene la capacidad de trabajar independientemente por lo tanto pueden ser probadas en forma individual.

Empezaremos por probar el algoritmo FFT/IFFT base-2 para lo cual se escoge la secuencia de prueba $xt(n)$ mostrada en la Figura 7.1, ya que con esta podremos observar si el algoritmo trabaja con números tanto reales como complejos.

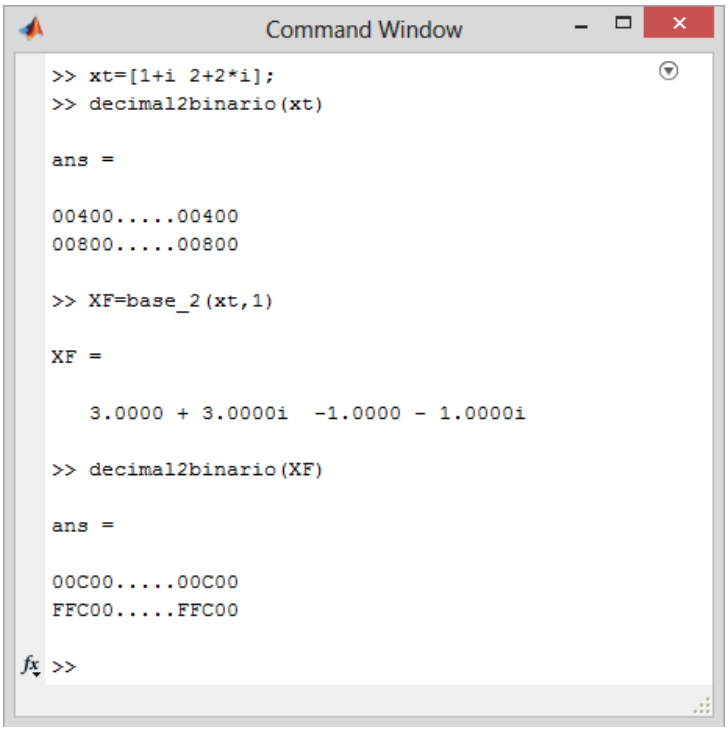

Figura 7.1.- Prueba del algoritmo FFT base-2 en Matlab.

Como se puede ver hacemos uso de una función extra llamada “decimal2binario” que se encarga de mostrarnos la representación de todos los términos como se observaran en el FPGA según la ecuación 6.1.

La simulación se presenta en la Figura 7.2 donde se presenta varios bloques de datos, en primer lugar se observa en verde la entrada que corresponde a la secuencia $xt(n)$ que es la misma que se utilizó en Matlab con su correspondiente representación.

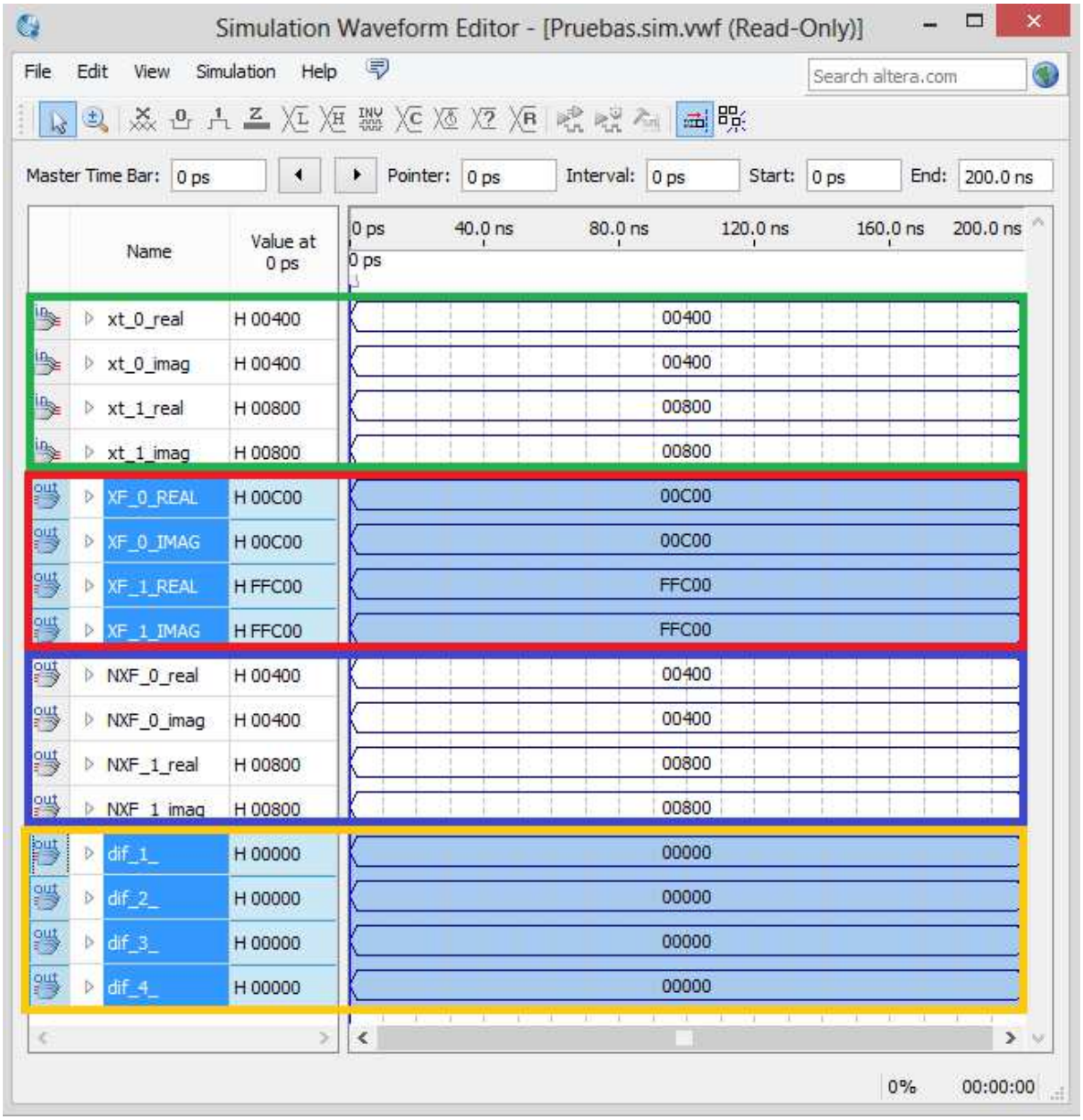

Figura 7.2 Simulación del algoritmo FFT/IFFT base-2 en Quartus II.

En rojo se presenta el resultado del bloque FFT/IFFT que en este caso es la FFT de la secuencia $xt(n)$ el cual denominamos $XF(k)$, siguiendo la Figura 6.8 a $XF(k)$ nuevamente se le aplica el bloque FFT/IFFT junto con el factor de escalamiento $1/N$, con lo cual se espera revertir la FFT y así obtener la secuencia original la cual se marca en azul y cómo podemos observar tanto el bloque en rojo y el bloque en azul tienen los mismos datos por lo cual podemos concluir que el proceso de revertir la FFT es posible, finalmente se presenta en amarillo la diferencia entre la secuencia de entrada y estos mismo una vez que se ha aplicado la FFT y luego la IFFT.

Como se observa la diferencia entre estos datos es de 0 en todos los casos cosa que es esperado ya que en estos bloques no ha intervenido ningún multiplicador por lo cual el proceso de revertir la transformada no genera ningún error.

La simulación se puede corroborar con una herramienta que el mismo Quartus II proporciona, el cual es signaltap embedded logic analyzer que es una poderosa herramienta que permite capturar las señales de nodos internos del FPGA, mientras que el dispositivo se está ejecutando en el sistema, lo que le da un acceso no intrusivo a las señales de los nodos del dispositivos internos. Puede optimizar la signaltap embedded logic analyzer megafunction funciona dentro del software quartus para el desarrollo del diseño, depuración y verificación. La característica más importante de esta herramienta es que nos permite conocer el comportamiento real del dispositivo una vez programado con el firmware diseñado.

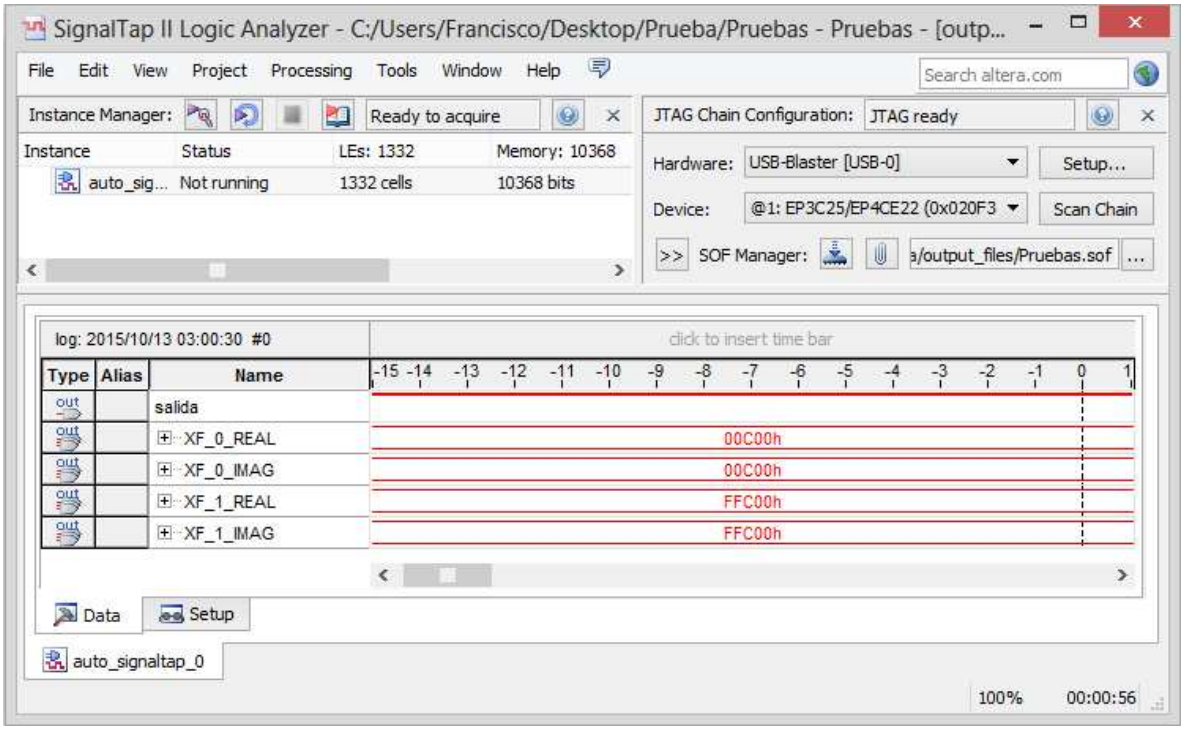

Figura 7.3.- Prueba del firmware del algoritmo FFT base-2 en Matlab.

Con la Figura 7.3 podemos observar que los datos correspondientes a la FFT son los mismos en Matlab, la simulación y el firmware programado en el FPGA.

Para el bloque de la FFT/ IFFT base-4 realizamos el mismo proceso, deseamos conocer su correcto funcionamiento, empezamos observando el comportamiento de una secuencia de prueba en Matlab, que como se ve en la Figura 7.4 ahora se ha agregado una nueva entrada además de $xt(n)$ para poder realizar, ya sea la FFT si esta es 0 o la IFFT si es diferente de 0.

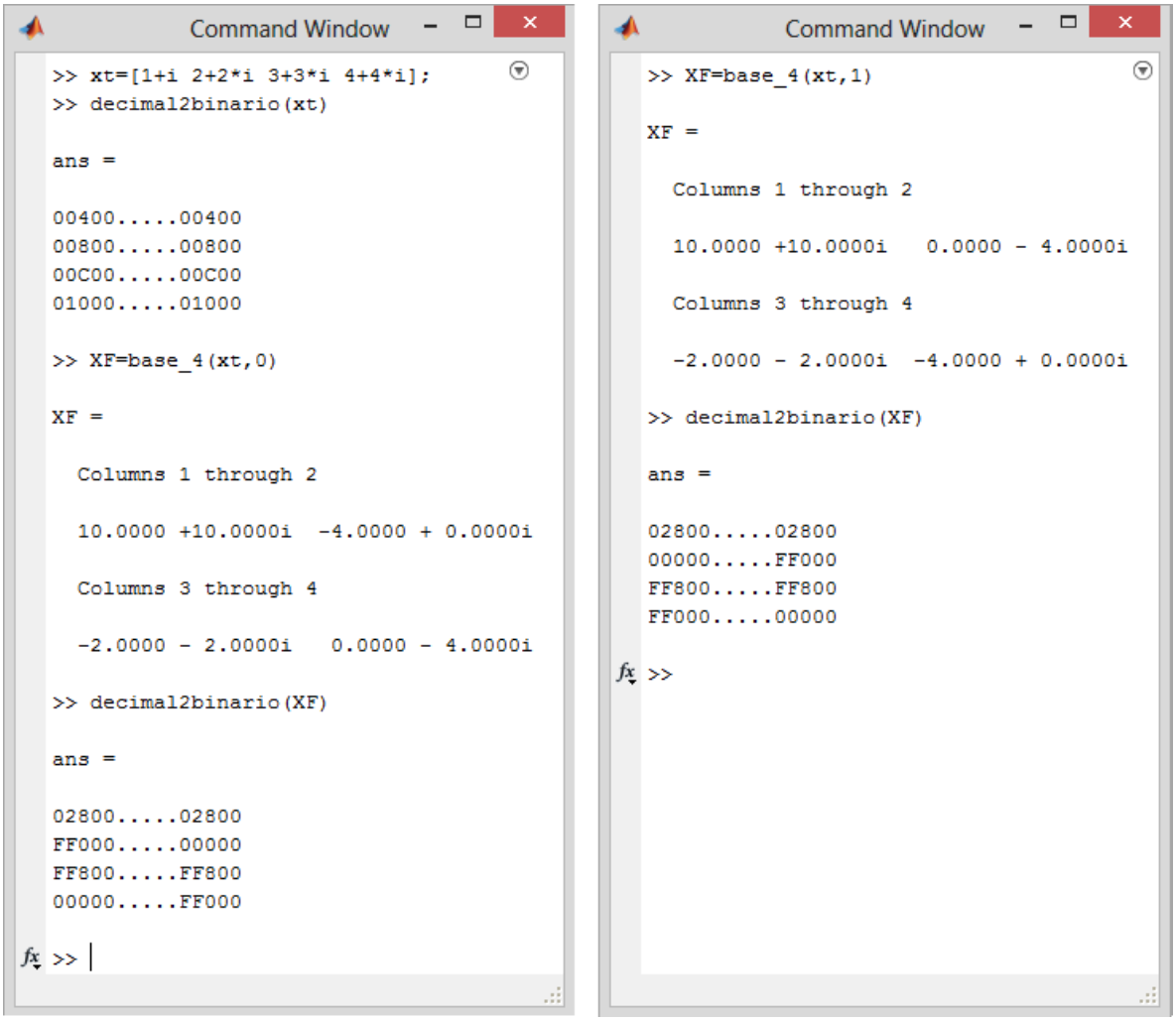

Figura 7.4.- Prueba del algoritmo FFT/IFFT base-4 en Matlab.

Como se puede observar en la Figura 7.5 los resultados concuerdan con los obtenidos con Matlab, además se ha agregado ahora una entrada extra que aparece en la parte superior llamada FFT_IFFT la cual realiza la FFT si está en bajo y la IFFT si está en alto.

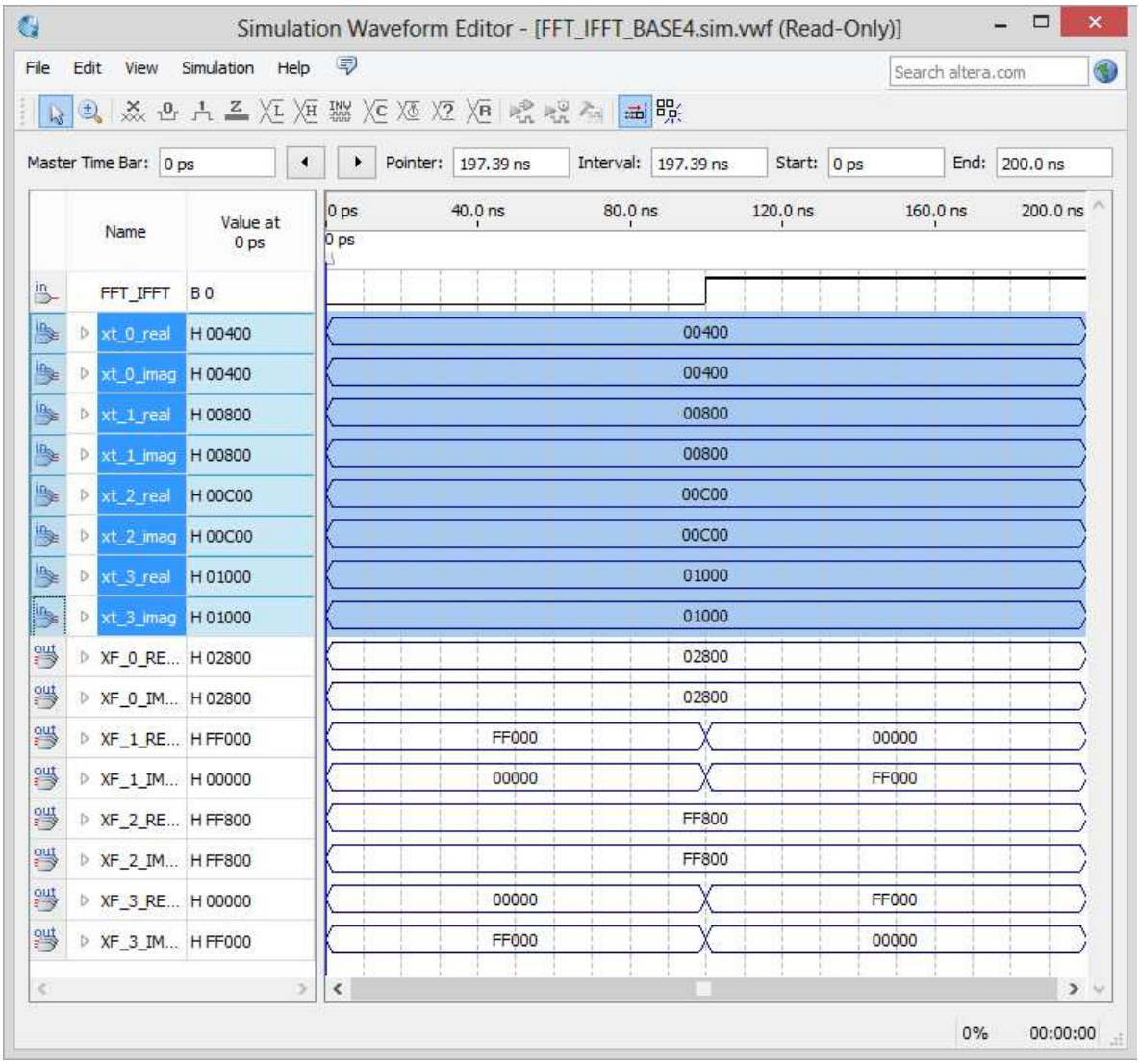

Figura 7.5.- Simulación del algoritmo FFT/IFFT base-4 en Quartus II.

Se puede observar en la Figura 7.6 y Figura 7.7 que los datos obtenidos con el analizador lógico concuerda con lo obtenido tanto en Matlab y la simulación, como en el caso de la FFT/IFFT base-2, la FFT/IFFT base-4 no genera ningún error, ya que en estos módulos no se utiliza ningún multiplicador y estos son los que generan el error debido a los procesos de truncamientos que son necesarios en estos.

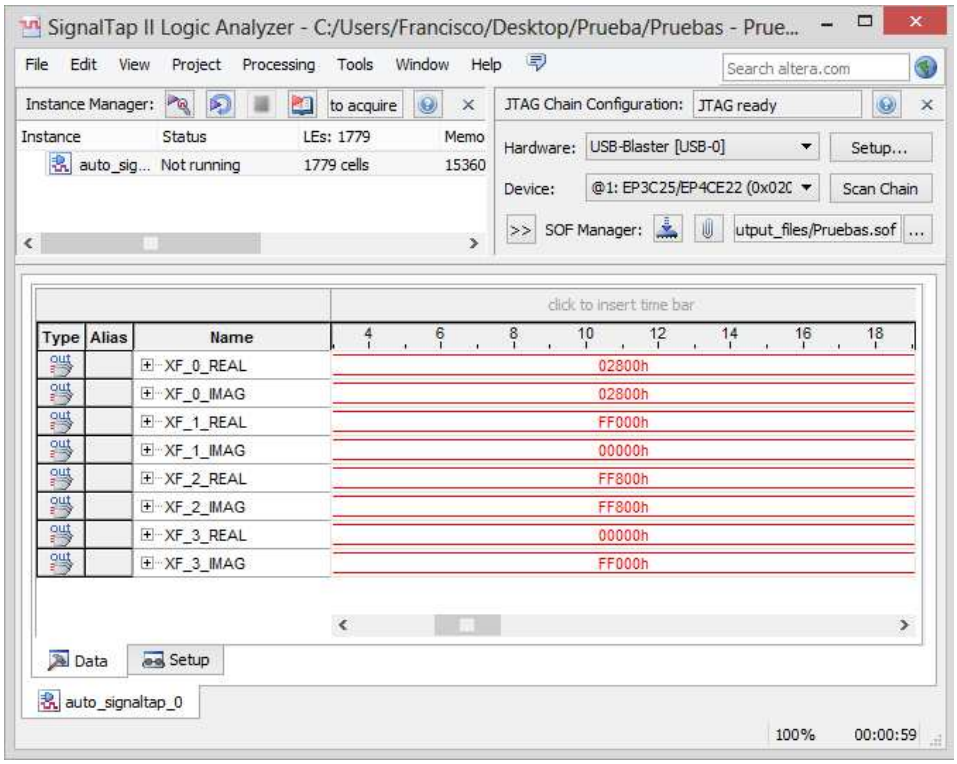

Figura 7.6.- Prueba del firmware del algoritmo FFT base-4.

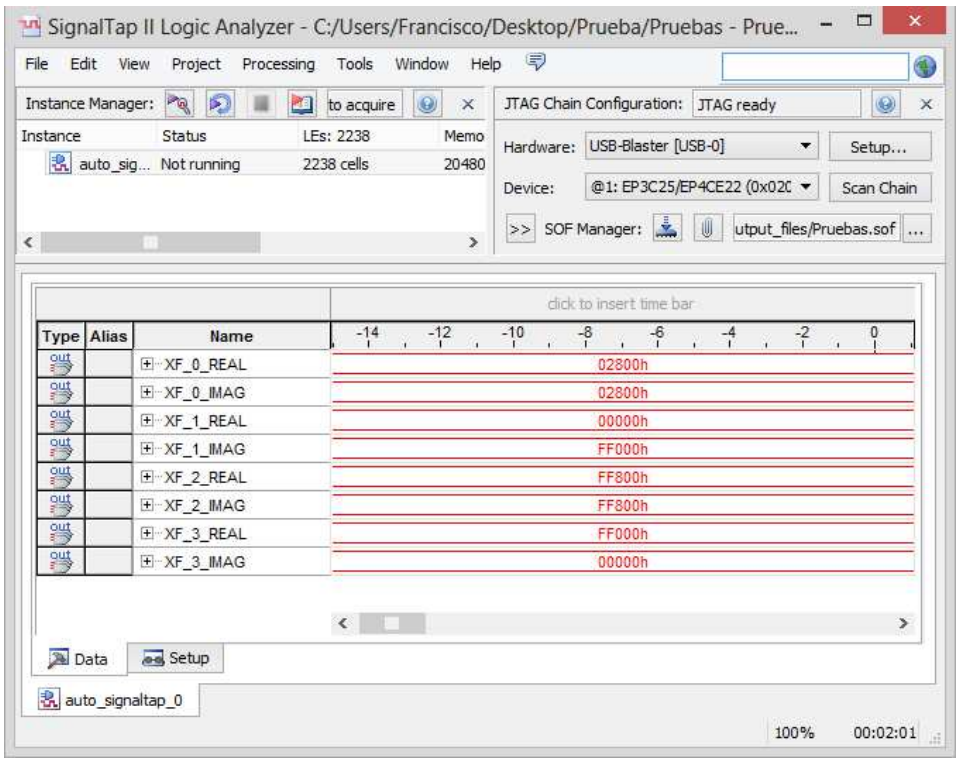

Figura 7.7.- Prueba del firmware del algoritmo IFFT base-4.

Para los algoritmos de 8 y 16 puntos además de conocer el correcto funcionamiento de los bloques es necesario conocer también el error generado, en estos bloques se utiliza el multiplicador de 20 bits y por lo tanto se generan errores de truncamiento, además los factores de rotación al estar sobre el circulo unitario generan valores entre 0 y 1 por lo cual se puede generar error de redondeo. En la Figura 7.7 se puede observar el comportamiento esperado por el bloque.

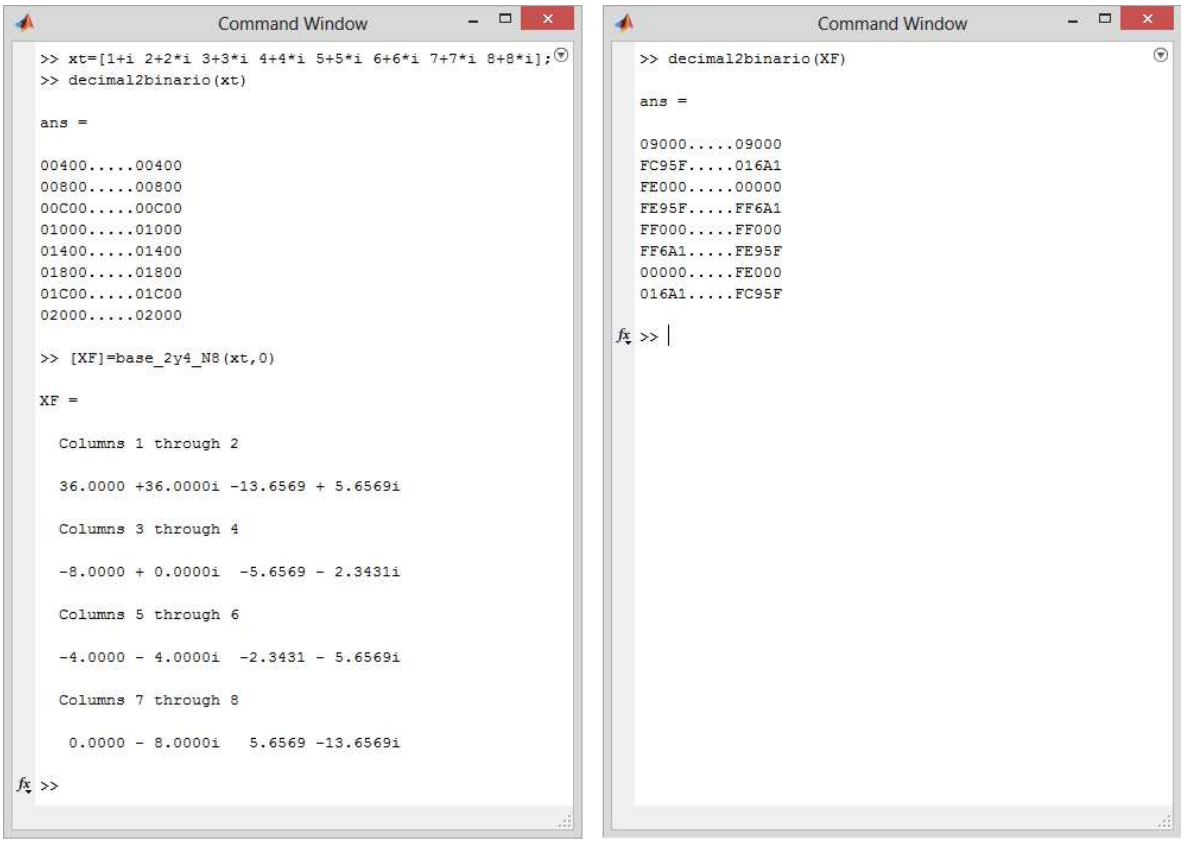

Figura 7.8.- Prueba del algoritmo FFT/IFFT para N=8 en Matlab.

Como se observa entre la Figura 7.8 y la Figura 7.9, los valores obtenidos y los esperados no son los mismo, se ha generado un error, pero este error puede ser despreciable por el hecho de que en el sistema de comunicaciones es necesario realizar la IFFT para luego realizarle la FFT lo que genera la codificación y decodificación, con lo cual es mucha más importante obtener la secuencia original al realizar estos dos procesos y conocer el error generado por estos dos procesos en conjunto.

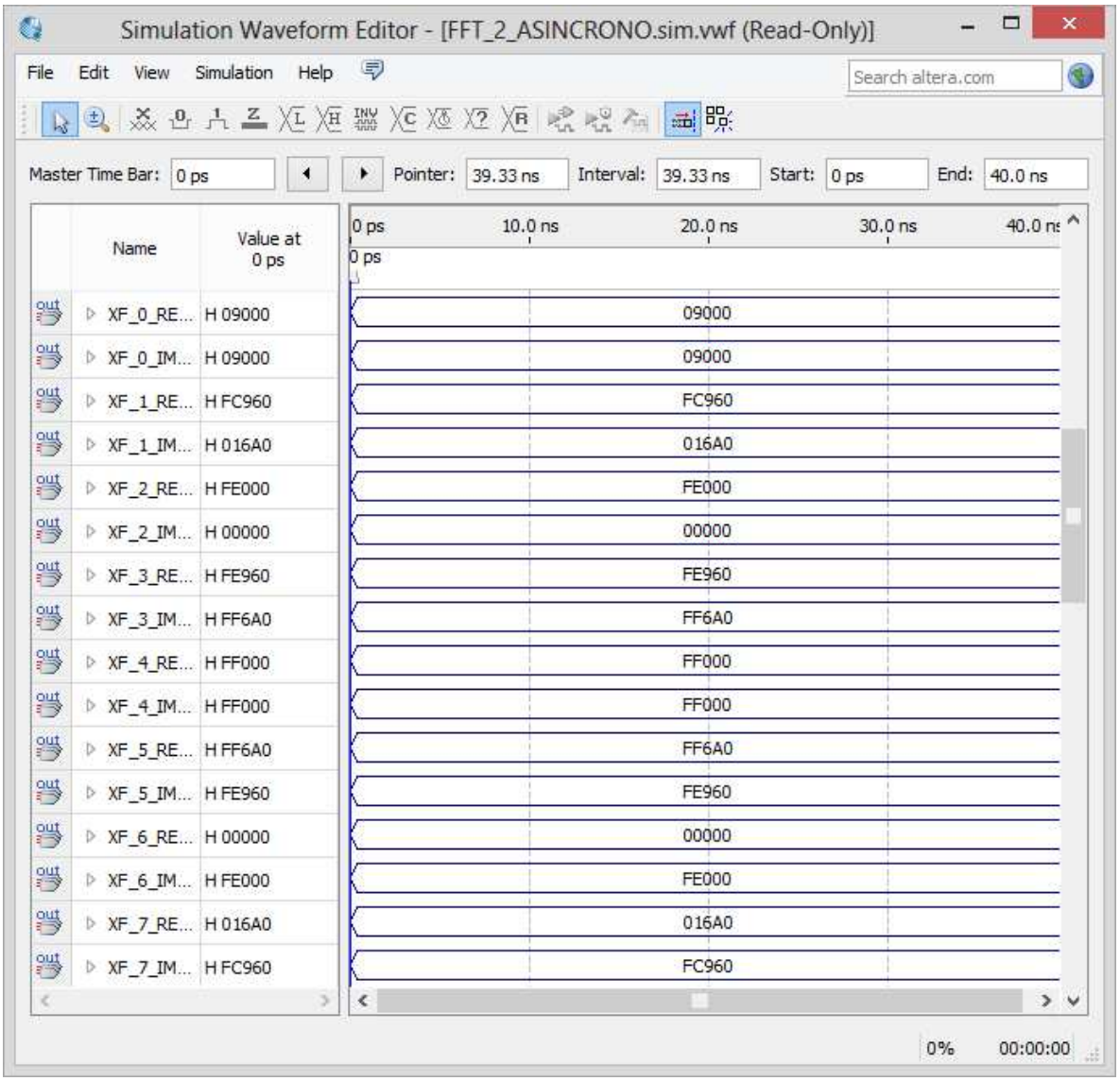

Figura 7.9.- Simulación del Firmware FFT/IFFT para N=8 en Quartus II.

Dicho error puede observarse mejor en la Figura 7.10 donde varias secuencias de números son puesto a prueba para así conocer cuál es el error máximo generado por este bloque. El error se encuentra entre ± 2, si tomamos en cuenta que tenemos un dato de 20 bits con lo cual tenemos un rango de 0 a 2^{20} con lo cual podemos concluir que el error generado por el bloque FFT/IFFT para $N = 8$ es de 0.00019%.

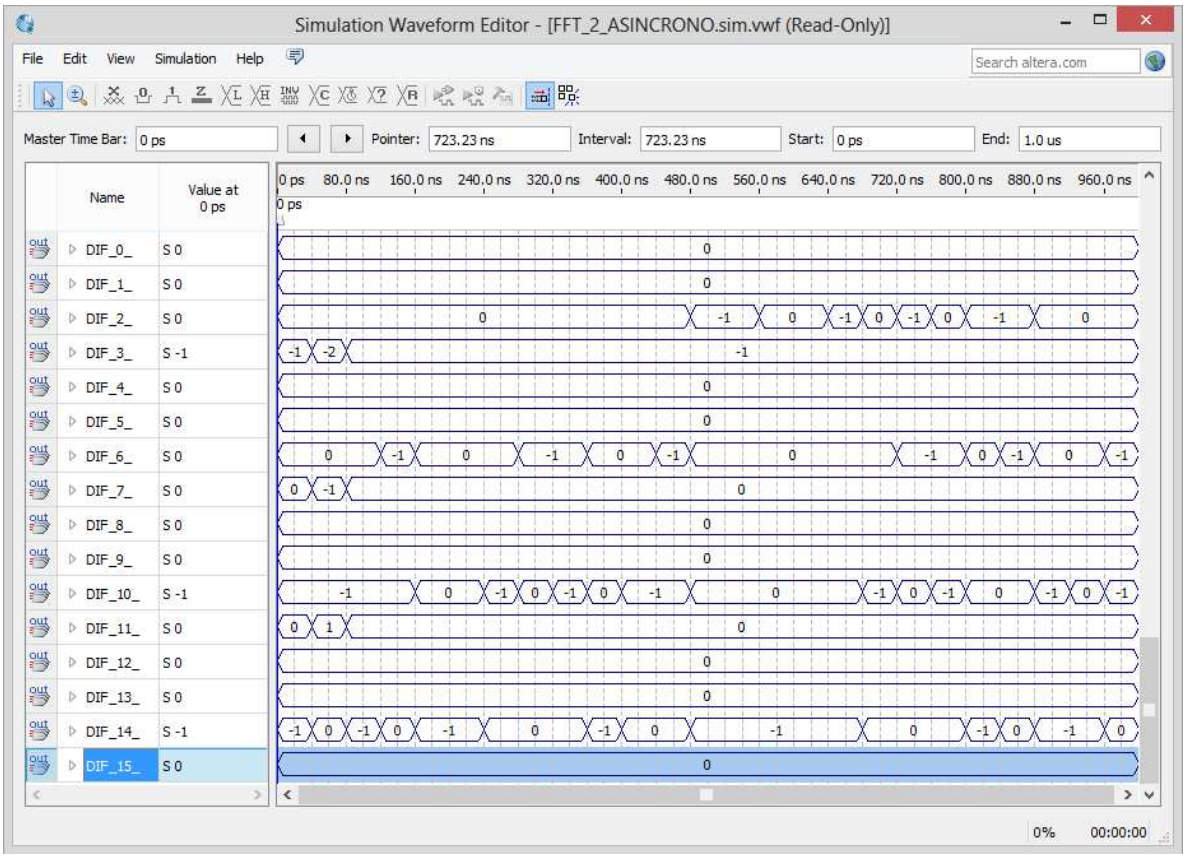

Figura 7.10.- Error generado de varias secuencias para el firmware FFT/IFFT para N=8 en Quartus II.

Para el bloque FFT/IFFT para $N = 16$ solo presentaremos los datos referentes al error generado que como se ve en la Figura 7.11 este puede variar entre ± 11, el aumento del error es debido al aumento en el número de muestras a procesar lo que genera un mayor número de factores de rotación, por lo que se requiere una mayor resolución en estos y por lo tanto se genera mayores errores de redondeo y truncamiento.

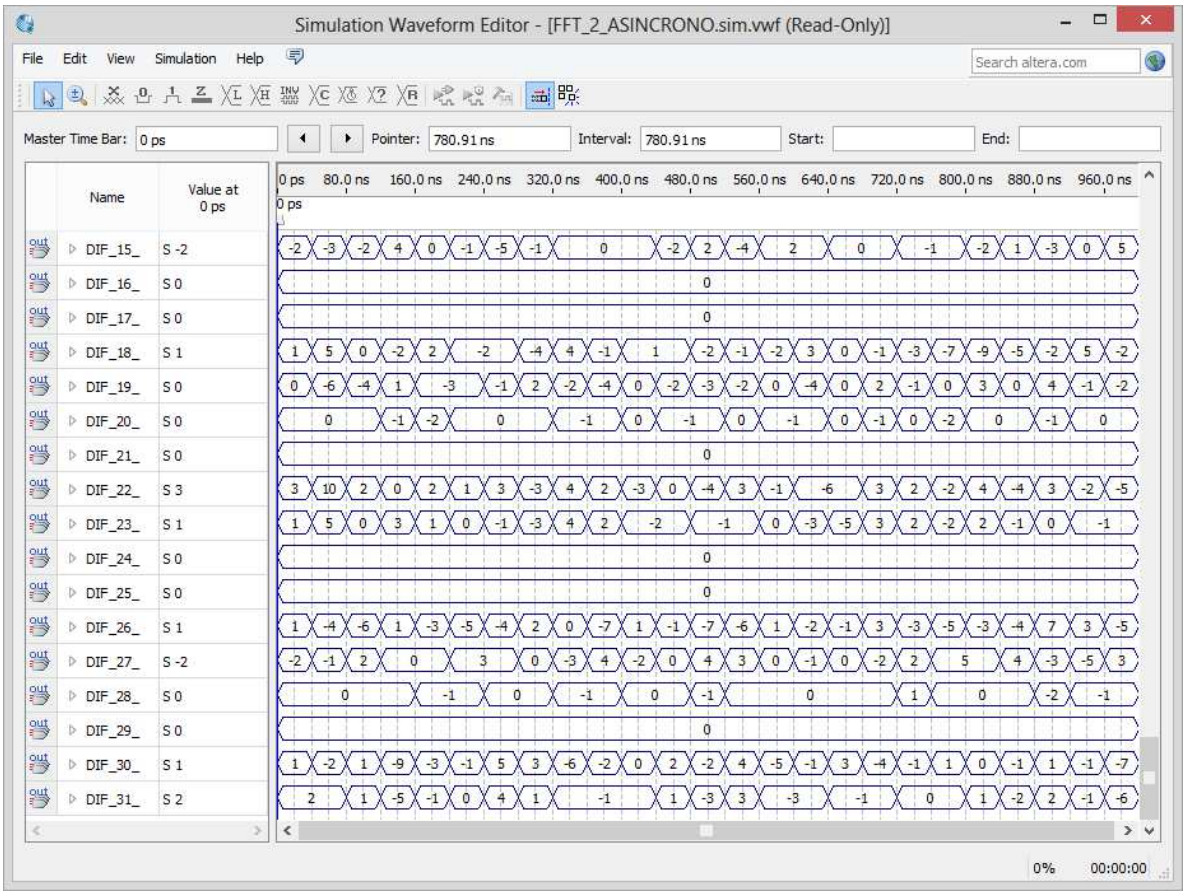

Figura 7.11.- Error generado de varias secuencias para el firmware FFT/IFFT para N=16 en Quartus II.

7.2 Prueba del algoritmo FFT/IFFT de 128 muestras.

Para la comprobación del correcto funcionamiento del firmware FFT/IFFT de 128 muestras se usaron varias secuencias de prueba, entre ellas una específicamente ideada para probar el estándar IEEE 802.16-2009, la cual muestra todas las etapas que tienen que realizarse y las señales que estas etapas generan junto con el error que estas pueden tolerar para poder comprobar el correcto funcionamiento del sistema de comunicaciones, incluyendo la parte la FFT e IFFT que se encargan de realizar la modulación y demodulación de la de las señales.

También se ingresaron otras secuencias de funciones conocidas las cuales buscan mostrar que este bloque no solo funciona para la tarea específica de este sistema de comunicaciones, si no que puede ser utilizado para otras aplicaciones. La primera secuencia de prueba es un escalón unitario de 1 Hz. que por las mismas características del bloque tiene que estar compuesta por 128 muestras como se observa en la Figura 7.12.

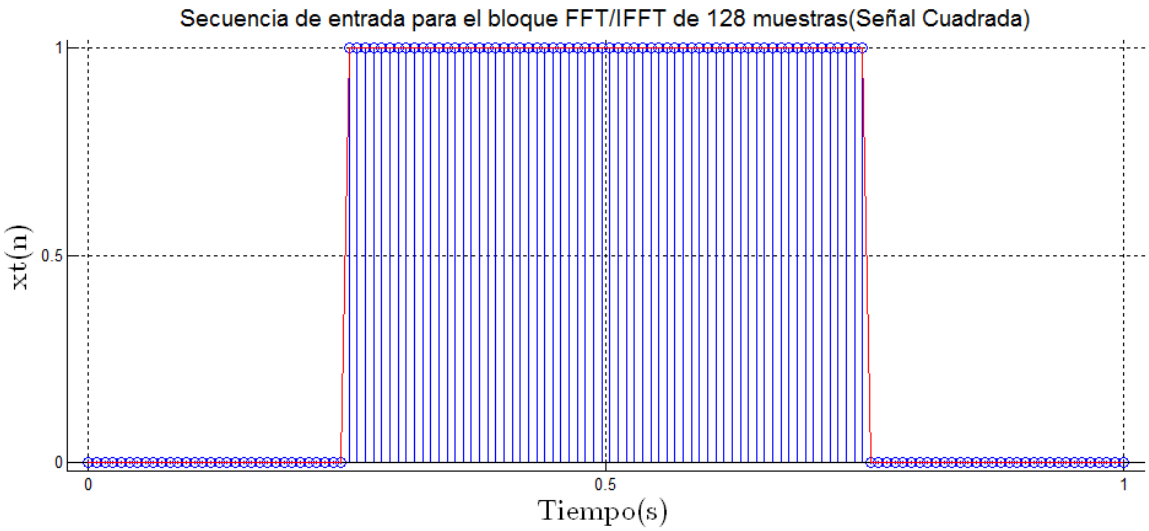

Figura 7.12.- Representación gráfica de la secuencia de entrada de una señal cuadrada para el bloque FFT/IFFT de 128 muestras.

Tenemos una secuencia en el tiempo la cual está compuesta solo de unos y ceros. Ya que se trata de una función conocida sabemos cuál será su representación en frecuencia la cual es una función sinc(x), esta coincide con los datos entregados por el FPGA como se muestra en la Figura 7.13, en esta figura existen 2 señales sobrepuestas, una es la simulación obtenida en Matlab y la otra son los datos obtenidos del FPGA.

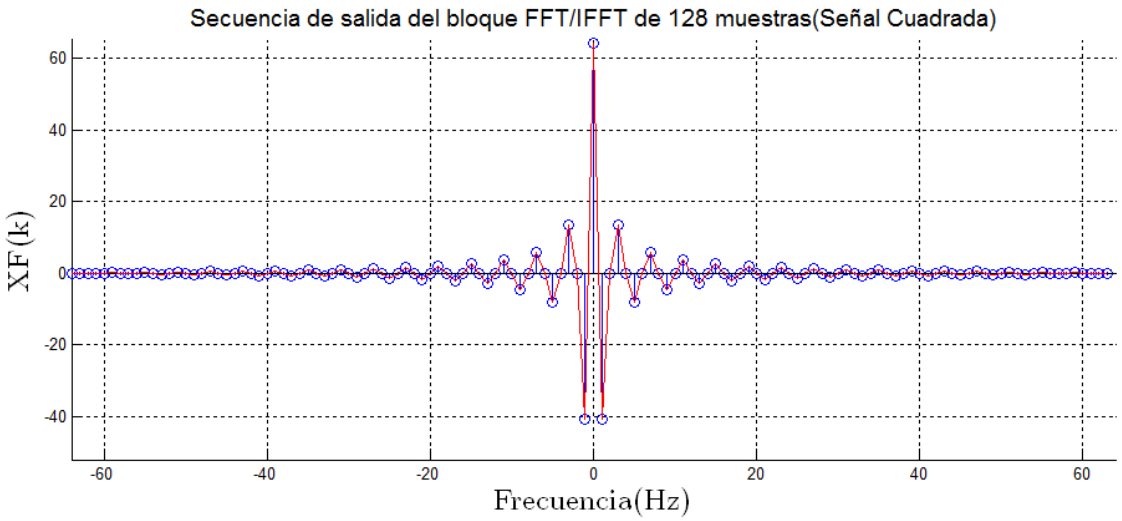

Figura 7.13.- Representación gráfica de la secuencia de salida del bloque FFT/IFFT de 128 muestras al introducir una señal cuadrada.

La señal del FPGA tiene un error generado por los errores de truncamiento y redondeo que se generan por los multiplicadores digitales de 20 bits necesarios para realizar las multiplicaciones complejas que está en el rango de $\pm 0.003\%$, el cual no se puede observar en Figura 7.13. Por lo cual para poder observarlo necesitamos agrandar la imagen como se muestra en la Figura 7.14.

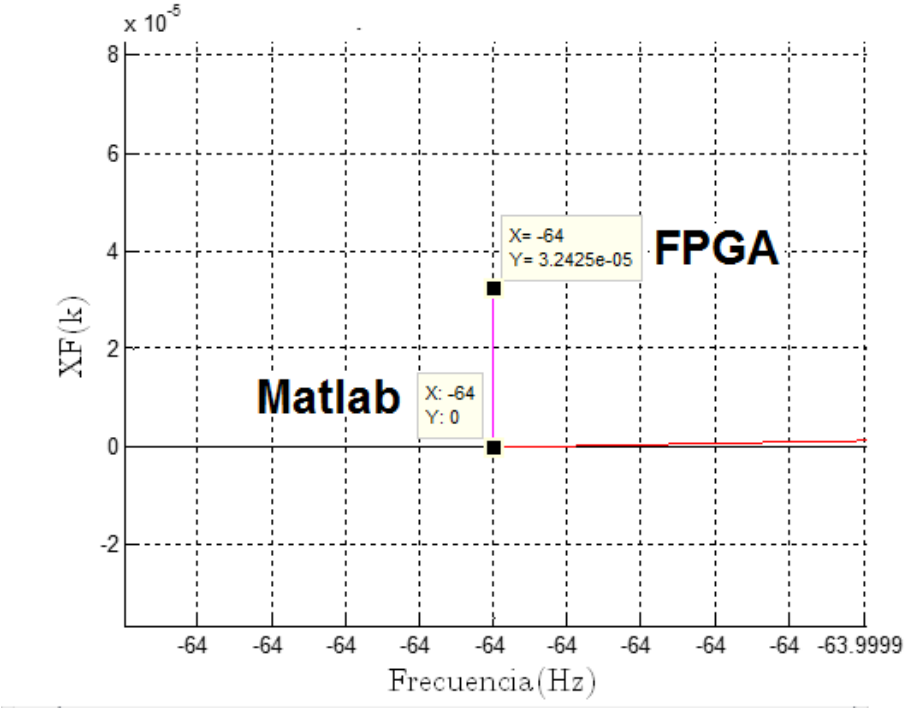

Figura 7.14.- Zoom realizado a la Figura 7.13 para poder observar el error.

Como se puede ver en la última figura la diferencia entre el dato obtenido en Matlab y el FPGA es muy pequeña lo suficiente como para poder ser despreciable para la aplicación deseada, aunque el error está enfocado para que pueda ser utilizado para el sistema de comunicaciones, si se deseara un mayor o menor margen del error para otras aplicaciones se podría variar utilizando una palabra de menor o mayor número de bits. Se presentan dos secuencias de entrada para el bloque FFT/IFFT de 128 muestras que son un seno de 10 Hz. que se puede ver en la Figura 7.15.

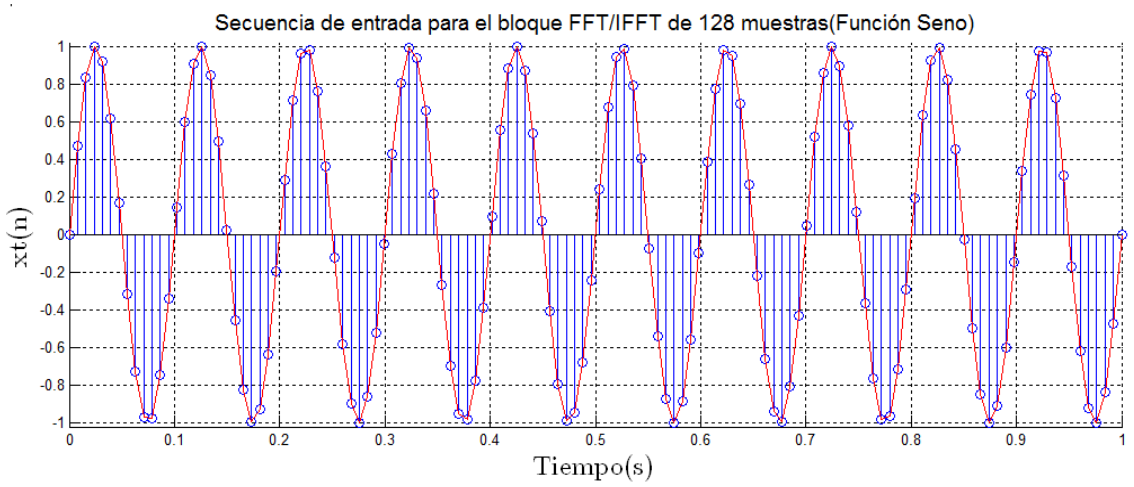

Figura 7.15.- Representación gráfica de la secuencia de entrada de una función seno para el bloque FFT/IFFT de 128 muestras.

Como se observa en la Figura 7.16 al introducir una función seno de 10 Hz. el espectro de dicha señal solo tendrá componentes en ± 10 ya que esta es la única armónica de la función de entrada.

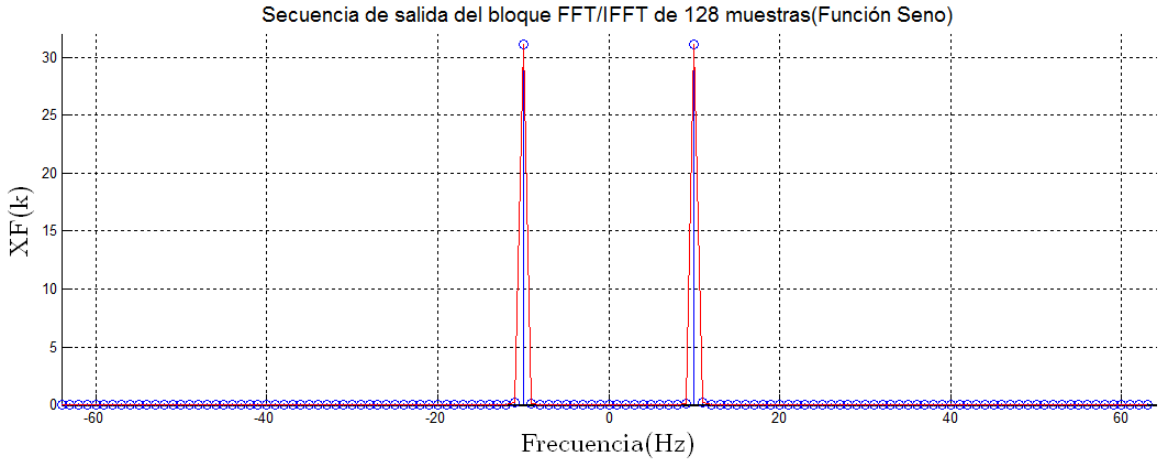

Figura 7.16.- Representación gráfica de la secuencia de salida del bloque FFT/IFFT de 128 muestras al introducir una función seno de 10 Hz.

Por ultimo para la secuencia de entrada se introducirá al bloque FFT/IFFT de 128 muestras una función Delta de Dirac como se muestra en la Figura 7.17.

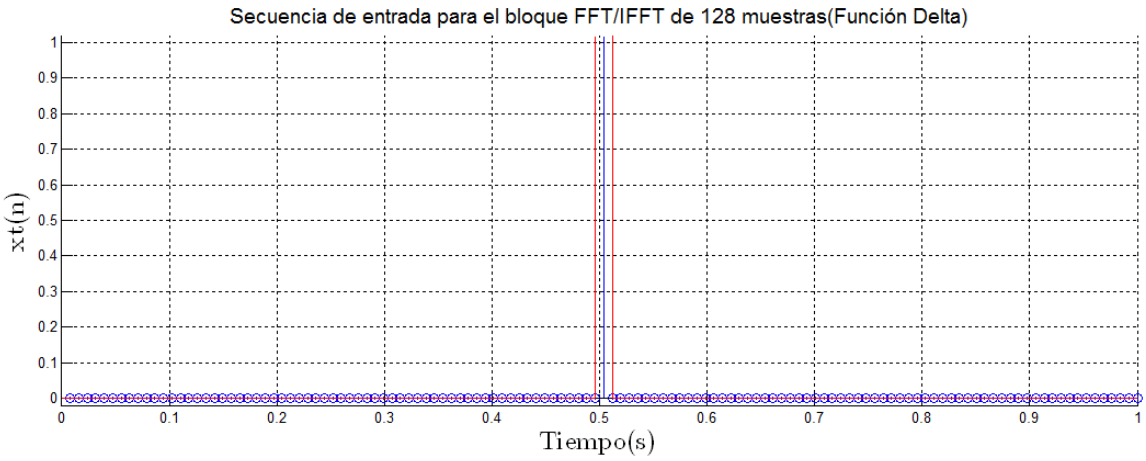

Figura 7.17.- Representación gráfica de la secuencia de entrada de una función seno para el bloque FFT/IFFT de 128 muestras.

La secuencia de salida del bloque FFT/IFFT de 128 muestras correspondiente a una entrada delta se muestra en la Figura 7.18 que como era esperado se forma una señal donde todas las componentes tienen la misma magnitud en todas la frecuencias.

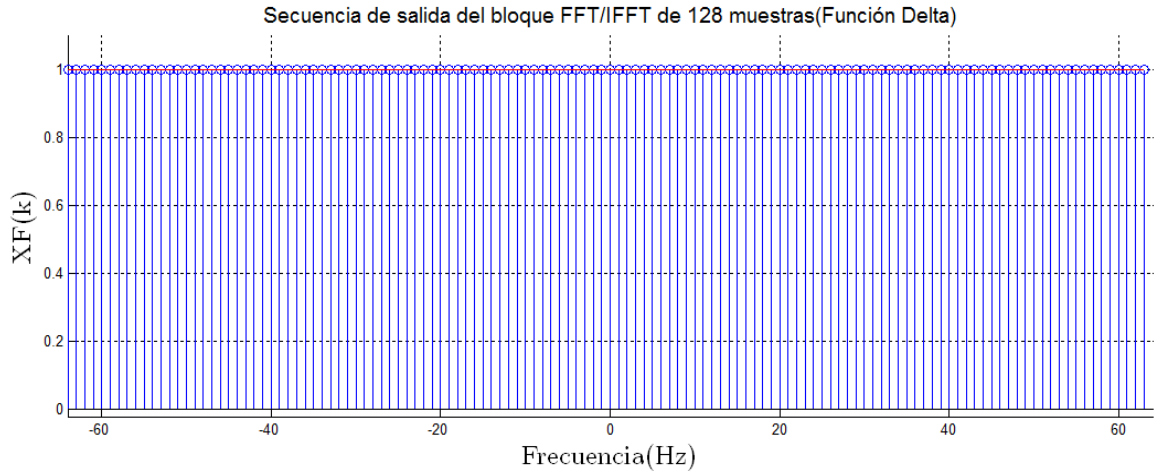

Figura 7.18.- Representación gráfica de la secuencia de salida del bloque FFT/IFFT de 128 muestras al introducir una función Delta.

Tanto las simulaciones realizadas como las pruebas del firmware arrojaron varios resultados positivos, como comprobar que el módulo FFT/IFFT es capaz de manejar números reales y complejos, se puede obtener la FFT o IFFT según se desee de una secuencia de 128 datos, el proceso de obtener la FFT es reversible pero en

algunos casos se genera un error que varía de ± 35 en su representación entera binaria, que al saber el rango de nuestra palabra podemos decir que el error puede ser del $\pm 0.003\%$ lo cual es aceptable para los requerimientos necesarios en el sistema de comunicaciones.

Con esto comprobamos que el bloque no solo cumple con todos los requerimientos necesarios para el sistema de comunicaciones en el cual se desea implementar, también se han hecho pruebas con diversas secuencias de entrada de las cuales se conoce de antemano su FFT para comparar estas señales con los datos entregados por nuestro bloque FFT/IFFT.

7.3 Comparativa de los recursos utilizados entre diversas versiones del firmware del algoritmo FFT/IFFT de 128 muestras.

Existe un módulo de propiedad intelectual de Altera capaz de realizar la FFT e IFFT de una secuencia de entrada que ofrece varias herramientas, como el poder adaptarlo a las diversas versiones de FPGA que maneja Altera, seleccionar el número de muestras para la FFT y el grado de precisión de los factores de rotación. Pero utilizar este módulo es necesario el pago de una licencia la cual solo tiene vigencia de un año y cuyo precio es excesivo, este es un gran problema ya que no se cuentan con los recursos necesarios para poder comprar dicha licencia.

Otro gran problema surge al compilar el módulo IP de Altera FFT/IFFT con características semejantes a el modulo diseñado en esta tesis, se puede observar en la Figura 7.19 el reporte generado por la compilación, solo, del módulo IP core de Altera el cual excede los recursos de nuestro FPGA por lo cual lo hace inviable para poder ser utilizado no solo por el precio de la licencia si no por el uso excesivo de recursos, aunque se podría recortar ciertas características esto provocaría que no se tenga las propiedades necesarias para poder utilizarlo en el sistema de comunicaciones.

Por lo cual se hizo evidente la necesidad de diseñar nuestro propio módulo FFT/IFFT, en un principio se trató de realizar los algoritmos de forma paralela tal cual están representados en la literatura pero esto trajo varios problemas que no pudieron ser solucionados de ninguna forma, sobre todo el hecho de que estos algoritmos no podrían ser implementados en el FPGA ya que utilizaban recursos más allá de los disponibles y a pesar de tener ciertas ventajas como fácil implementación y mayor velocidad de procesamiento, no pudieron ser utilizados debido a que se excedía en gran medida el número de elementos lógicos y multiplicadores necesarios.

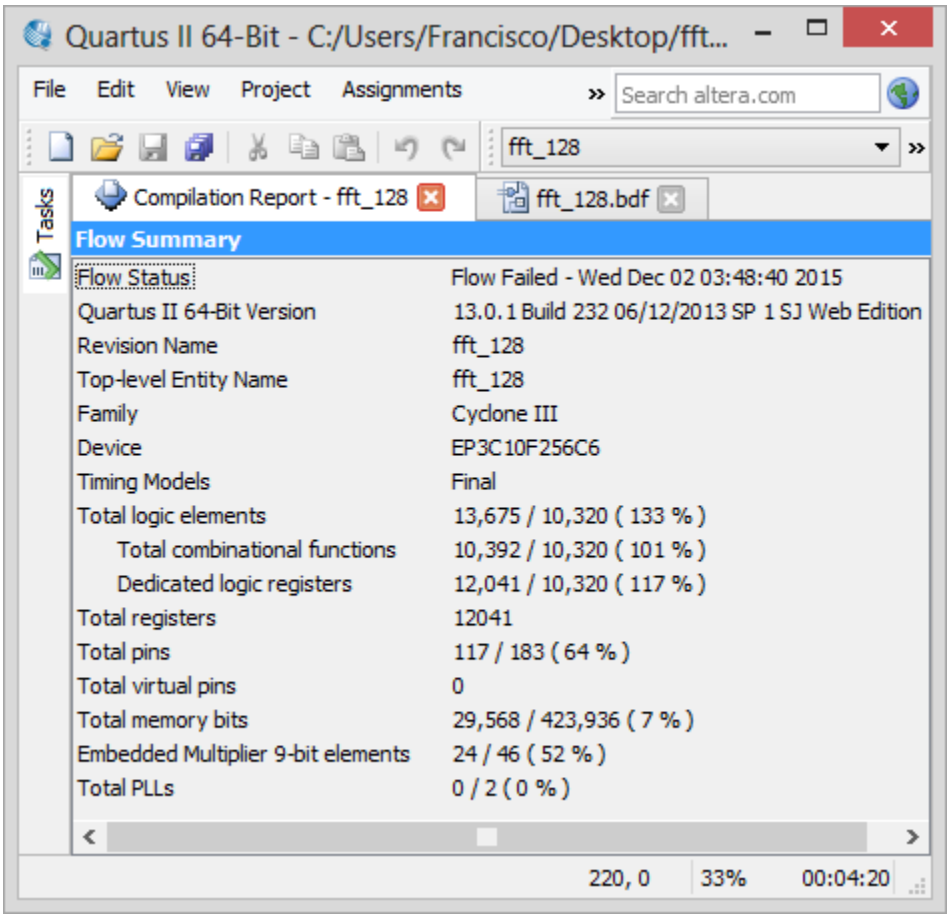

Figura 7.19.- Reporte generado por la compilación del módulo IP de Altera en el FPGA.

Se realizaron varios diseños de los firmwares para los algoritmos que se implementaron en el FPGA para las FFT/IFFT de 4, 8 y 16 puntos, utilizando la técnica de multiplexación se pudo reducir considerablemente el número de módulos necesarios para estos firmware y más importante aún el número de multiplicaciones se redujo significativamente hasta una cantidad que puede ser implementado sin exceder los recursos del FPGA. La técnica de multiplexación redujo los multiplicadores y elementos lógicos necesarios en comparación a otros algoritmos diseñados de forma concurrente.

Se presentan varios cuadros donde se comparan bloques contra sus versiones que utilizaban un mayor número de elementos lógicos. En el Cuadro 1 se presentan los recursos necesarios para implementar el multiplicador complejo en paralelo que como se ve con solo implementar uno, se sobrepasa los multiplicadores embebidos disponibles en el FPGA, pero con el multiplicador complejo multiplexado se reducen los multiplicadores embebidos permitiéndonos poder implementar 3 de estos, los necesarios para la FFT/IFFT de 128 muestras.

Cuadro 1.- Recursos necesarios para implementar un multiplicador complejo.

Bloque	Elementos Lógicos	Multiplicadores Embebidos
Multiplicador Complejo Paralelo	136	24
Multiplicador Complejo Multiplexado	188	6

En el Cuadro 2 y Cuadro 3 están los recursos para la FFT/IFFT para $N = 2$ y $N = 4$ que como se ve no utilizan ningún multiplicador embebido, apresar de esto el número de elementos lógicos necesarios no varía mucho.

Cuadro 2.- Recursos necesarios para implementar la FFT/IFFT para N=2.

Bloque	Elementos Lógicos	Multiplicadores Embebidos
FFT de N=2 sin Factor de Rotación	80	0
FFT de N=2 con Factor de Rotación	216	24

Cuadro 3.- Recursos necesarios para implementar la FFT/IFFT para N=4.

Bloque	Elementos Lógicos	Multiplicadores Embebidos
FFT de N=4 Multiplexado	480	0
FFT de N=4 Paralelo	456	24

En el Cuadro 4 y Cuadro 5 se puede ver que ambos usan 6 multiplicadores embebidos, necesarios para implementar un multiplicador complejo multiplexado

Cuadro 4.- Recursos necesarios para implementar la FFT/IFFT para N=8.

Bloque	Elementos Lógicos	Multiplicadores Embebidos
FFT de $N = 8$ Paralelo	1640	120
FFT de $N = 8$ Multiplexado	1924	6

Cuadro 5.- Recursos necesarios para implementar la FFT/IFFT para N=16.

Bloque	Elementos Lógicos	Multiplicadores Embebidos
FFT de $N = 16$ Paralelo	4872	408
FFT de $N = 16$ Multiplexado	3219	6

En el Cuadro 6 se comparan los recursos utilizados por el IP core de Altera, la primera versión de nuestro firmware la cual requería de poco más 25000 elementos lógicos y 987 multiplicadores embebidos, recursos que están por encima de los disponibles en el FPGA, y finalmente la versión final de nuestro módulo FFT/IFFT de 128 muestras, el cual utiliza las técnicas de multiplexación ya mencionadas que utilizo 9503 elementos lógicos y 18 multiplicadores que son 92% de los elementos lógicos y 78% de los multiplicadores disponibles en el FPGA, la versión multiplexada utilizo solo 11% de los bloques de memoria.

Cuadro 6.- Recursos necesarios para implementar la FFT/IFFT para N=128.

Bloque	Elementos Lógicos	Multiplicadores Embebidos
Módulo FFT/IFFT IP core de Altera	13675	12
Módulo FFT/IFFT de 128 muestras paralelo	Más de 25000	987
Módulo FFT/IFFT de 128 muestras multiplexado	9503	18

Se puede observar en este último cuadro que la única versión posible de implementar en el FPGA con el que contamos, es la versión multiplexada del módulo diseñado en esta tesis.

7.4 Diagrama de entradas y salidas del bloque FFT/IFFT de 128 muestras.

El módulo final está representado en la Figura 7.20 que como se puede ver están representadas todas las entradas y salidas necesarias para su correcto funcionamiento y cada una será descrita a continuación.

Entrada serial: es un bus de 20 bits donde se agregaran cada uno de los datos necesarios para poder realizar la FFT/IFFT como se ha descrito anteriormente un dato está conformado por dos palabras una para la parte real y otra para la parte imaginaria por lo tanto en la entrada serial deberán introducirse 256 palabras de 20 bits con el formato ya antes descrito.

Entrada lista: es una señal de activación que cuando está en 1 nos dirá que se están ingresando datos por la entrada serial cuando sea cero el modulo estará en espera.

Entrada valida: es la señal que nos dirá que el dato que está en el bus es válido y por lo cual cada que ocurra un flanco de subida el dato que este en el bus pasara a guardarse en la memoria.

Procesando: una vez ingresado los 256 datos el módulos empezara a procesar los datos en forma automática por lo cual esta señal estará en alto, una vez terminado el procesamiento volverá estar en bajo.

FFT/IFFT listo: una vez terminado de procesar esta salida se pondrá en alto indicando que ya se puede realizar la extracción de los datos procesados.

Salida serial: Es un bus donde saldrán uno a uno las palabras para formar cada uno de los datos requeridos ya que la FFT/IFFT entrega el mismo número de datos que el que se ingresa este también entregara 256 palabras ya procesadas por el modulo.

Salida valida: Una vez que FFT/IFFT está en alto se puede extraer los datos uno a uno con la ayuda de esta salida, para lo cual cada que ocurra un flanco de subida entregara una palabra de 20 bits.

Salida lista: una vez entregado las 256 palabras en la salida serial esta salida pasa a alto indicando que todas las tareas ha sido completadas.

Restablecer: Esta señal cuando está en alto borra todos los registros y memorias internas del módulo y lo regresa a su condición ideal.

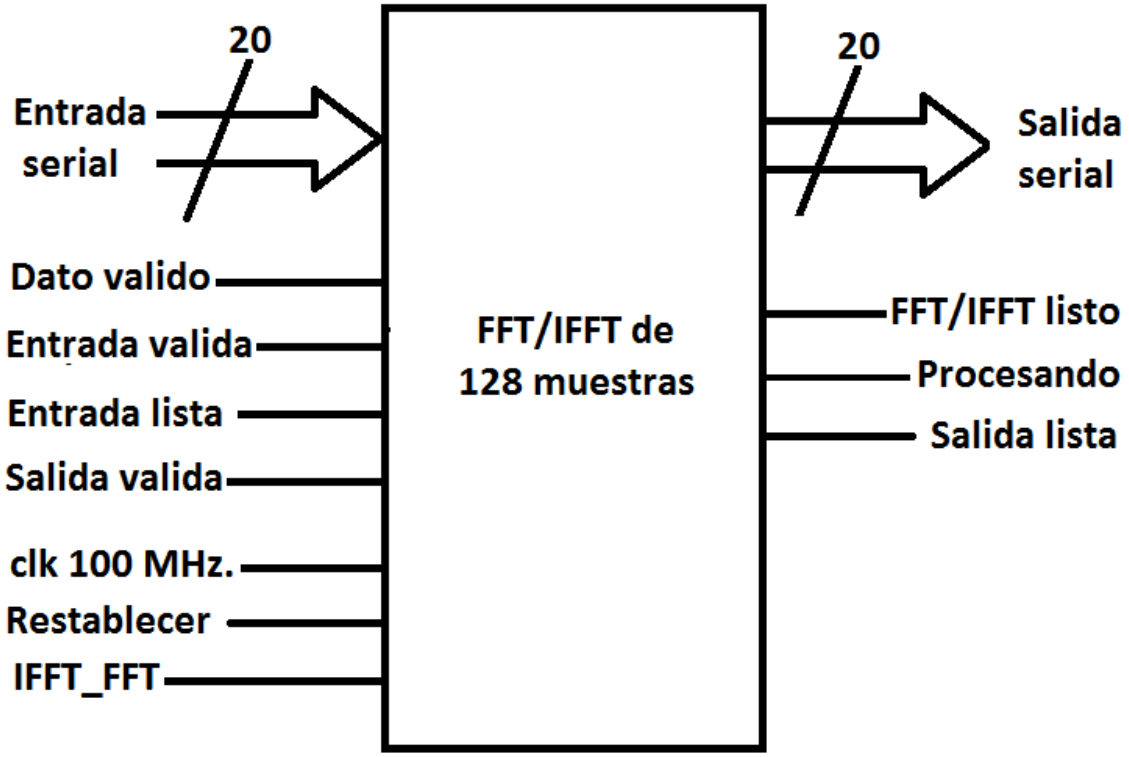

Figura 7.20.- Bloque de entradas y salida del módulo FFT/IFFT de 128 muestras.

Conclusiones.

La importancia de la FFT quedo clara al ver su gran ventana de aplicaciones en muchas y variadas ramas. Además su potencial en los sistemas computacionales ha ido en aumento gracias a las nuevas tecnologías con lo cual se generan nuevos enfoques que se dan a la FFT tanto desde el punto de vista de software como de hardware.

Los script en Matlab fueron de gran utilidad ya que nos permitieron probar y comprobar las modificaciones que se fueron proponiendo conforme se plantearon las ideas de cómo mejorar los firmwares ya hechos, tal es el caso de las FFT's de gran tamaño basadas en los algoritmos bases.

Con el correspondiente análisis de los diferentes algoritmos de la FFT se decidió utilizar los algoritmos Cooley-Tukey ya que estos tienen grandes ventajas para su implementación en el FPGA. Para el caso particular de un sistema de comunicación OFDMA se utilizara los algoritmos base-2 y base-4 ya que a partir de ellos se puede generar la DFT de 128 muestras.

De forma definitiva se descartaron los algoritmos de valores primos debido a que su implementación resulta muy complicada. Estos algoritmos reducen al mínimo el número de multiplicaciones necesarias para su implementación lo cual se logra aumentado significativamente la complejidad del algoritmo y el número de sumas necesarias, esta reducción de las multiplicaciones necesarias en muchos casos no compensa el aumento en la complejidad ya que existen otros métodos como el Zero Padding que nos permite utilizar algoritmos Cooley-Tukey mucho más sencillos de implementar.

Como se observa para obtener el algoritmo de 8 y 16 puntos fue necesario diseñar los algoritmos de menor número de muestras ya que como lo marca la teoría para obtener la FFT/IFFT de N muestras se puede obtener a partir de dos FFT/IFFT de $N/2$ muestras si estamos utilizando un algoritmo base-2.

Integrar en el mismo bloque la FFT y la IFFT resulta uno de los puntos más importantes ya que esto significa una reducción considerable en el diseño del algoritmo ya que nos ahorra la necesidad de desarrollar el firmware para el algoritmo de la FFT y otro para IFFT.

La técnica de multiplexación nos permitió reducir significativamente la cantidad de recursos necesarios para implementar la FFT/IFFT de 8 y 16 puntos esto se logró ya que se utiliza varias veces un mismo módulo en diferentes momentos, esta reducción de recursos conlleva el aumento de registros pero sobretodo el aumento en el tiempo necesario para realizar dicho proceso.

En el firmware para obtener la DFT de 128 muestras se observó que si utilizáramos el método directo necesitaría de 16384 multiplicadores complejos de 20 bits, lo cual sobrepasa de forma de forma sumamente excesiva los recursos del FPGA con el que contamos, por lo cual se hizo necesario utilizar la FFT para poder reducir este número, aun así utilizando la FFT requiere de 622 multiplicadores complejos y a pesar de que la reducción de los multiplicadores fue significativa, está aún por encima de lo posible, por lo cual se requirió de utilizar la técnica de multiplexación lo cual redujo la cantidad de multiplicadores complejos a solo tres lo cual hace posible su implementación.

La gran diferencia entre los algoritmos que trabajan concurrentemente y los que utilizan multiplexores es el tiempo que les lleva realizar la misma tarea, en los primeros algoritmos el procesamiento total no excedía de los 4 ciclos de reloj y esta espera era para asegurar que el resultado sea el correcto, pero, en los algoritmos con multiplexación la FFT de 8 puntos necesita 50 ciclos de reloj y la FFT de 16 puntos 120 ciclos de reloj el procesamiento total de la FFT/IFFT de 128 muestras dura 30µ segundos lo cual está dentro de los requerimientos necesarios.

La gran ventaja de nuestro diseño es el poder modificarlo y utilizarlo sin ningún problema de licencia que fue una de la principales razones para la realización de este tema de tesis ya que utilizar un IP Core de altera dedicado a esta función elevaba en gran medida los costos del sistema de comunicación ya que se requería la compra de una licencia. Nuestro firmware a diferencia de un IP Core, puede ser adaptado para otros sistemas donde sea requerido la FFT, en este caso se enfocó para una FFT de 128 muestras, pero siguiendo las estrategias de diseño de este trabajo se pueden implementar FFT’s de mayor o menor número de muestras utilizando los bloques básicos diseñados.

Trabajo a futuro.

Una de las grandes ventajas de haber desarrollado nuestro propio firmware es la posibilidad de poder realizar modificaciones a este, sin tener problemas de pagos de licencia o derechos de patentes. Por lo cual unos de los primeros trabajos que se tiene pensados hacer sobre este bloque es la optimización de los recursos utilizados por el boque FFT/IFFT con la finalidad de poder implementar en un solo FPGA un transmisor o receptor OFDMA, junto con el bloque diseñado en éste tema de tesis, lo cual puede ser posible multiplexando diversos bloques que trabajan en forma paralela o reduciendo los bloques de memorias utilizados.

El haber desarrollado los bloques básicos de la FFT/IFFT de menor número de muestras como bloques independientes nos permite poder utilizarlos para poder manejar diversos números de muestras para obtener la FFT. Como ejemplo se podría desarrollar una FFT de 32 muestras usando el los bloques FFT/IFFT para 4 y 8 muestras usando la misma metodología aquí propuestas, o se podría realizar la FFT/IFFT de 256 muestras usando dos veces el módulo FFT/IFFT de 16 muestras.

Las aplicaciones de la FFT resultan ser muy bastas y varias, como se indicó en el capítulo II, por lo cual el número de posibles implementaciones de estos bloques es muy amplia, no solo en lo relacionado al firmware si no al también al algoritmo ya que se plantea una idea muy fácil de implementar la FFT para cualquier número de muestras que podrían ser usado para desarrollar analizadores espectrales o procesamiento de señales biométricas por decir algunas.

Bibliografía.

[1] Gholamreza, P., & Abdulrahman, Y. (s.f.). OFDMA for the 4 Generation Cellular Networks. 2325-2330.

[2] Samuel c., C. (2010). *OFDMA System Analysis and Desing.* Artech House.

[3] Ortiz-Buenrostro, L. M. (2009). *Analisis y Diseño de Tecnicas de Reservación para Redes WIMAX IEEE 802.16.* Mexico,Mexico.

[4] Agusti Comes, R., A. F., Casadevall Palacio, F., Ferrus ferre, R., Perez Romero, J., & Sallent Roit, O. (2010). *LTE: Nuevas tendencias en comunicaciones Moviles.* Fundacion Vodafon de España.

[5] LaSorte, N., W. Barnes, J., & H. Refai, H. (2009). The History of Orthogonal Frequency Division Multiplexing. *Communications Magazine, IEEE*.

[6] Trimberger, S. M. (2015). Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology. *Proceedings of the IEEE*, 318-331.

[7] International Telecommunication Union. (mayo de 2015). *http://www.itu.int*. Obtenido de http://www.itu.int/net/pressoffice/press_releases/2014/46-es.aspx#.VVDdaPl_Okp.

[8] 4g Americas. (mayo de 2015). *http://www.4gamericas.org*. Obtenido de http://www.4gamericas.org/es/newsroom/press-releases/4g-americas-la-cooperacion-entre-operadores-y-gobierno-es-esencial-para-el-futuro-inalambrico-de-la-region.

[9] Vergara González, J. M. (2008). *Simulacion de un Esquema deModulacion/Demodulacion OFDM Utilizando un Modelo de Canal Multitrayectoria.* Guayaquil, Ecuador.

[10] Proakis, J. G., & Manolakis, D. G. (1986). *Digital Signal Processing.* Prentice Hall International Inc.

[11] López Martínez, F. J. (2005). *Diseño de Transmisor y Receptor para Redes Inalámbricas W-MAN.* Malaga.

[12] Arioua, M., Belkojuch, S., & Hassani, M. M. (2012). Efficient 64-Point FFT/IFFT Processor Based on 8-Point FFT Module for OFDM BAse WLAN. *International Review on Computers and Software*, 92-99.

[13] Kaur, S., & Mehra, R. (2012). FPGA Implementacion of OFDM Transciver Using FFT Algorithm. *International Journal of Engineering Science and Technology*, 1532-1537.

[14] Cortez, J. M., Gil, M. d., & lopez, D. B. (1998). La Transformada Rapida de Fourier y su controvertido origen. *Ciencia y desarrollo*, 70-76.

[15] Duhamel, P., & Vetterli, M. (1989). Fast Fourier Transforms: A tutorial review and a state of the art. *Signal Processing 19*, 259-299.

[16] Cooley, J. W., & W. Tukey, J. (1965). An Algorithm for the Machine Computation of Complex Fourier Series. *Mathematics of computation*, 297-301.

[17] Heideman, M. T., Johnson, D. H., & Burrus, S. C. (1984). Gauss and the History of the Fast Fourier Transform. *IEEE ASSP Magazine*, 14-21.

[18] Frigo, M., & Johnson, S. G. (1997). The Fastest Fourier Transform in the West. *MIT Labroratory for Computer Science*, 1-20.

[19] Cooley, J. W., Lewis, P. A., & Welch, P. D. (1967). Historical Notes on the Fast Fourier Transfor. *IEEE TRans. Audio Electroacoust*, 260-262.

[20] Schneider, D. (24 de Febrero de 2012). *http://spectrum.ieee.org/*. Obtenido de http://spectrum.ieee.org/computing/software/a-faster-fast-fourier-transform

[21] Rao, K., Kim, D., & Hwang, J. (2010). *Fast Fourier Transform: Algorithms and Applications.* Springer.

[22] Mateer, T. (2008). *Fast Fouries Trasform With Applications.*

[23] Briceño Marquez, J. E. (2011). *Principios de las Comunicaciones.* Taller de publicaciones de la facultad de Ingenieria, Universidad de los Andes.

[24] Sidney Burrus, C. (2009). *Fast Fourier Transform.* Connexions.

[25] Lofgren, J., & Nilsson, P. (2011). On Hardware Implementation of Radix 3 and Radix 5 FFT Kernels for LTE systems., (págs. 1-4). Lund.

[26] J. Tocci, R., S. Widmer, N., & L. Moss, G. (2007). *Sistemas Digitales. Principios y Aplicaciones.* Prentice Hall.

[27] ALTERA. (agosto de 2012). www.altera.com. Obtenido de http://www.altera.com/literature/hb/cyc3/cyclone3_handbook.pdf

[28] Oran Brigham, E. (1988). The Fast Fourier Transform and its Applications. Prentice hall.

[29] Boemo Scalvinoni, E. (2005). *Estado del Arte de la Tecnologia FPGA.* Instituto nacional de Tecnologia Industrial.

[30] Brown, S., & Rose, J. (1996). FPGA and CPLD Architectures: A Tutorial. *IEEE Design & Test of Computers*, 42-57.

[31] Trimberger, S. (1993). A Reprogrammable Gate Array and Applications. *proceedings of the IEEE*, 1030-1041.

[32] Quadri, F., & Tete, A. (2013). FPGA Implementation of Digital Modulation Techniques. *International Conference on Communication and Signal Processing*, (págs. 913-917). India.

[33] ALTERA. (agosto de 2012). *www.altera.com.* Obtenido de http://www.altera.com/literature/hb/cyc3/cyclone3_handbook.pdf

[34] Altera. (1995). *Max + Plus II: Programmable Logic Development System.*

[35] Oppenheim, A. V. (1989). *Computation of the Discrete Fourier Transform.* Prentice-Hall.

[36] Navarro giovanetti, J. A. (2008). *Evolución de 3G y su Convergencia a 4G en Comunicaciones Móviles.* Tesis de Ingenieria, Valdivia.

Glosario

2D	*Second Dimension / Segunda Dimensión.*
2G	*Second Generation / Segunda Generación.*
3D	*Third Dimension / Tercera Dimensión.*
3G	*Third Generation / Tercera Generación.*
3GPP	*Third Generation Partnership Project / Proyecto Asociación de Tercera Generación.*
4G	*Fourth Generation / Cuarta Generacion.*
ACM	*Association for Computing Machinery / Asociación para la Maquinaria Informatica.*
ADSL	*Asymmetric Digital Subscriber Line / Línea de Abonado Digital Asimétrica.*
AHDL	*Altera HDL / HDL de Altera.*
ASIC	*Application-Specific Integrated Circuit / Circuitos Integrados de Aplicación Específica.*
CP	*Cyclic Prefix / Prefijo Ciclico.*
CLB	*Configurable Logic Block / Bloque Lógico Configurable.*
COFDM	*Coded OFDM / OFDM Codificado.*
CPLD	*Complex Programmable Logic Device / Dispositivos Logicos Programables Complejos.*
DBA	*Digital Audio Broadcasting / Difusión de Audio Digital.*
DFT	*Discrete Fourier Transform / Transformada Discreta de Fourier.*
DIF	*Decimation in Frequency / Dicemación en Frecuencia.*
DIT	*Decimation in Time / Dicemación en Tiempo.*
DSP	*Digital Signal Processor / Procesadores Digitales de Señales.*

DVB-T	*Digital Video Broadcasting – Terrestrial / Difusión de Video Digital por Redes Terrestres.*
E-UTRA	*Evolved UMTS Terrestrial Radio Access / Acesso de Radio Terrestre Envolvente UMTS.*
FDM	*Frequency Division Multiplexing / Multiplexación por División de Frecuencia.*
FEC	*Forward Error Correction / Seguidor de Corrección de Errores.*
FFT	*Fast Fourier Transform / Transformada Rápida de Fourier.*
FFTW	*The Fastest Fourier Transform in the West / La más Rápida Transformada de Fourier en el Oeste.*
FIFO	First Input, First Output / Primera Entrada, Primera Salida.
Flash-OFDM	*Fast Low Latency Access with Seamless Handoff – OFDM / Acceso Rapido de Baja Latencia con Traspaso sin Fisuras.*
FPGA	*Field Programmable Gate Array / Arreglo de Compuertas Programable en Campo.*
FT	*Fourier Transform / Transformada de Fourier.*
GPS	*Global Positioning System / Sistema de Posicionamiento Satelital.*
GSM	*Groupe spécial mobile / Sistema global para las comunicaciones móviles.*
HD	*High Definition / Alta Definición.*
HDL	*Hardware Description Language / Lenguaje de Descripción de Hardware.*
HSOPA	*High Speed OFDM Packet Access / Alta velocidad de acceso a paquetes OFDM.*
ICI	*Inter-Carrier Interference / Interferencia Entre Portadoras.*
IDFT	*Inverse Discrete Fourier Transform / Transformada Discreta de Fourier Inversa.*
IEEE	*Institute of Electrical and Electronics Engineers/ Instituto de Ingenieros en Electricidad y Electrónica.*

IFFT	Inverse Fast Fourier transform / *Transformada Rápida de Fourier Inversa.*
IP	*Internet ProtocoL / Protocolo de Internet.*
IR	*Infrared radiation / Radiación Infrarroja*
ISI	*Inter-Symbol Interference / Interferencia Entre Símbolos.*
ITU	*International Telecommunication Union / Unión Internacional de Telecomunicaciones.*
LE	Logical Element / Elemento Lógico.
LPF	*Low-Pass Filter / Filtro Pasa Bajas.*
LTE	*Long Term Evolution / Evolución a Largo Plazo.*
MBWA	*Mobile Broadband Wireless Access / Acceso de Banda Ancha Movil Inalámbrica.*
MIMO	*Multiple-Input Multiple-Output / Múltiples Entradas Múltiples Salidas.*
MIT	*Massachusetts Institute of Technology / Instituto tecnológico de Massachusetts.*
OFDM	*Orthogonal frequency-division multiplexing / Multiplexación por División de Frecuencias Ortogonales.*
OFDMA	*Orthogonal Frequency-Division Multiple Access / Acceso Múltiple por División de Frecuencias Ortogonales.*
PAL	*Programmable Array Logic / Arreglo Logico Programable.*
PFA	*Prime Factor Algorithm / Algoritmo de Factores Primos.*
PLD	*Programmable Logic Device* / Dispositivos lógicos Programables.
.	
PLL	*Phase Locked Loop / Lazo de Seguimiento de Fase.*
PSK	*Phase Shift Keying / Modulación por Desplazamiento de Fase.*
QAM	Quadrature Amplitude Modulation / Modulación de Amplitud en Cuadratura.
QoS	*Quality of Service / Calidad de Servicio.*

QPSK	Quadrature Phase Shift Keying / Modulación por Desplazamiento Cuadrafásica.
RAM	*Random Access Memory / Memoria de Acceso Aleatorio.*
RF	*Radiofrecuencia.*
SFT	*Spears Fourier Transform / Transformada Dispersa de Fourier.*
SIAM	Society for Industrial and Applied mathematics / Sociedad para la Industria y Matematicas Aplicadas.
SoC	Sistem on Chip / Sistema en un Chip.
SOFMDA	*Scalable Orthogonal Frequency Division Multiple Access / Acceso Múltiple por División de Frecuencias Ortogonales Escalable.*
UHF	*Ultra High Frequency / Ultra Alta Frecuencia.*
UMTS	*Universal Mobile Telecommunications System / Sistema universal de telecomunicaciones móviles.*
VHDL	*Very High Speed Integrated Circuit HDL / HDL para Circuito Integrado de Muy Alta Velocidad.*
VHF	*Very High Frequency / Muy Alta Frecuencia.*
VLSI	*Muy Alta Escala de Integracion / Very Large Scale Integration.*
WiMAX	*Worldwide Interoperability for Microwave Access / Interoperabilidad Mundial para Acceso por Microondas.*
WFTA	*Winograd Fourier Transform Algorithm / Algoritmo de la Transformada de Fourier de Winograd.*
WMAN	*Wireless Metropolitan Area Network / Red de Área Metropolitana Inalambrica.*
WRAN	*Wireless Regional Area Networks / Red Inalámbricas de Área Regional.*

Apéndice A.

Script del algoritmo base 2 de 2 puntos.

```
function XF=base_2(x,Wn)

    XF(1,1)=x(1,1)+x(1,2);
    XF(1,2)=(x(1,1)-x(1,2))*Wn;

End
```

Script del algoritmo base 4 de 4 puntos.

```
function XF=base_4(x,IFFT_FFT)

    a=real(x(1,1));
    b=imag(x(1,1));
    c=real(x(1,2));
    d=imag(x(1,2));
    e=real(x(1,3));
    f=imag(x(1,3));
    g=real(x(1,4));
    h=imag(x(1,4));

    XF(1,1)=  a+c+e+g  + i*( b+d + f+h );
    XF(1,3)=(a+e)-(c+g)+ i*((b+f)-(d+h));

    if IFFT_FFT == 0

        XF(1,2)= (a+d)-(e+h)+ i*((b+g)-(f+c));
        XF(1,4)= (a+h)-(e+d)+ i*((b+c)-(f+g));

    else

        XF(1,2)= (a+d)-(e+h)+ i*((b+c)-(f+g));
        XF(1,4)= (a+h)-(e+d)+ i*((b+g)-(f+c));

    end

end
```

Script del algoritmo base-2 y base-4 de 8 puntos.

```
function [XF]=base_2y4_N8(x,FFT_IFFT)

 if FFT_IFFT == 0
        h=i ;
    else
        h=-i;
    end

    a=[ x(1,1) , x(1,5) ] ;
    b=[ x(1,2) , x(1,6) ] ;
    c=[ x(1,3) , x(1,7) ] ;
    d=[ x(1,4) , x(1,8) ] ;

    [A]=base_2( a , 1 ) ;
    [B]=base_2( b , 1 ) ;
    [C]=base_2( c , 1 ) ;
    [D]=base_2( d , 1 ) ;

    A=[ A(1,1) , B(1,1) , C(1,1) , D(1,1) , ...
        A(1,2) , B(1,2) , C(1,2) , D(1,2) ];


    fr_1 = [ 1 , 1 , 1 , 1 , 1 , exp(-2*pi*1*h/8) , exp(-2*pi*2*h/8) ,
exp(-2*pi*3*h/8) ] ;

    A_fr=A.*fr_1 ;

    [A_F]=base_4(A_fr(1,1:4),FFT_IFFT);
    [B_F]=base_4(A_fr(1,5:8),FFT_IFFT);

    XF=[ A_F(1,1) , B_F(1,1) , A_F(1,2) , B_F(1,2) , ...
         A_F(1,3) , B_F(1,3) , A_F(1,4) , B_F(1,4) ] ;
end
```

Script del algoritmo base-4 de 16 puntos.

```
function [XF]=base_4_N16(x,FFT_IFFT)

    if FFT_IFFT == 0
        h=i ;
    else
        h=-i;
    end

    a=[ x(1,1) , x(1,5) , x(1,9)  , x(1,13) ] ;
    b=[ x(1,2) , x(1,6) , x(1,10) , x(1,14) ] ;
    c=[ x(1,3) , x(1,7) , x(1,11) , x(1,15) ] ;
    d=[ x(1,4) , x(1,8) , x(1,12) , x(1,16) ] ;


    [A]=base_4(a,FFT_IFFT) ;
    [B]=base_4(b,FFT_IFFT) ;
    [C]=base_4(c,FFT_IFFT) ;
    [D]=base_4(d,FFT_IFFT) ;

    A_2=[ A(1,1) , B(1,1) , C(1,1) , D(1,1) ] ;
    B_2=[ A(1,2) , B(1,2) , C(1,2) , D(1,2) ] ;
    C_2=[ A(1,3) , B(1,3) , C(1,3) , D(1,3) ] ;
    D_2=[ A(1,4) , B(1,4) , C(1,4) , D(1,4) ] ;

    fr_1 = [ 1 , exp(-2*pi*1*h/16) , exp(-2*pi*2*h/16) , exp(-
2*pi*3*h/16) ] ;
    fr_2 = [ 1 , exp(-2*pi*2*h/16) , exp(-2*pi*4*h/16) , exp(-
2*pi*6*h/16) ] ;
    fr_3 = [ 1 , exp(-2*pi*3*h/16) , exp(-2*pi*6*h/16) , exp(-
2*pi*9*h/16) ] ;

    A_fr=A_2  ;
    B_fr=B_2.*fr_1 ;
    C_fr=C_2.*fr_2 ;
    D_fr=D_2.*fr_3 ;

    [A_F]=base_4(A_fr,FFT_IFFT) ;
    [B_F]=base_4(B_fr,FFT_IFFT) ;
    [C_F]=base_4(C_fr,FFT_IFFT) ;
    [D_F]=base_4(D_fr,FFT_IFFT) ;

    XF =[ A_F(1,1) , B_F(1,1) , C_F(1,1) , D_F(1,1) ,...
          A_F(1,2) , B_F(1,2) , C_F(1,2) , D_F(1,2) ,...
          A_F(1,3) , B_F(1,3) , C_F(1,3) , D_F(1,3) ,...
          A_F(1,4) , B_F(1,4) , C_F(1,4) , D_F(1,4) ] ;
end
```

Script del algoritmo base-8 y bse-16 de 128 puntos.


```
function [ X_F ] = FFT_128_8x16 ( x_t , FFT_IFFT )

    if FFT_IFFT == 0

        h=i ;

    else

        h=-i;

    end

    ind=0;

    for fil=1:16

        ind=fil;

        for col=1:8

           ind=fil;
           ind2=ind+16*(col-1);
           sec_M(fil,col)=x_t(ind2)

        end

    end

    for ciclos = 1 : 16

        ciclos
        a=sec_M(ciclos,:)
        X_F8(ciclos,:)=base_2y4_N8(a,FFT_IFFT)

        if ciclos > 1

            ciclos

W_n_k=[exp(h*(2*pi/128)*(ciclos1)*(1)),exp(h*(2*pi/128)*(ciclos1)*(2)),..
exp(-h*(2*pi/128)*(ciclos-1)*(3)) , exp(-h*(2*pi/128)*(ciclos-1)*(4)),...
exp(-h*(2*pi/128)*(ciclos-1)*(5)) , exp(-h*(2*pi/128)*(ciclos-1)*(6)),...
                    exp(-h*(2*pi/128)*(ciclos-1)*(7)) ]

            X_F8(ciclos,2:8)
            X_F8(ciclos,2:8)=X_F8(ciclos,2:8).*W_n_k ;
            X_F8

        end
```

```
    end

    for ciclos2 = 1 : 8

        ciclos2
        a=X_F8(:,ciclos2)
        a=[a(1,1)  , a(2,1)  , a(3,1)  , a(4,1)...
           a(5,1)  , a(6,1)  , a(7,1)  , a(8,1)...
           a(9,1)  , a(10,1) , a(11,1) , a(12,1)...
           a(13,1) , a(14,1) , a(15,1) , a(16,1) ]
        X_F16(:,ciclos2)=base_4_N16(a,FFT_IFFT)

    end

    ind=0;

    for fil=1:16

        for col=1:8

           ind=ind+1;
           X_F(1,ind)=X_F16(fil,col);

        end

    end

end
```

Script del algoritmo base 3.

```
function [fft_m,XF]=base_3(x)

    W13=exp(((-i*2*pi)/3)*1)
    W23=exp(((-i*2*pi)/3)*2)

    XF(1,1)=x(1)+x(2)+x(3);
    XF(1,2)=x(1)+x(2)*W13+x(3)*W23;
    XF(1,3)=x(1)+x(2)*W23+x(3)*W13;

    fft_m=fft(x);

end
```

Script del algoritmo base–5.


```
function  [fft_m,XF]=base_5(x,FFT_IFFT)

    u=(2*pi)/5;

    s1=x(2)+x(5)
    s2=x(2)-x(5)
    s3=x(4)+x(3)
    s4=x(4)-x(3)
    s5=s1+s3
    s6=s1-s3
    s7=s2+s4
    s8=s5+x(1)

    a1=((cos(u)+cos(2*u))/2)-1
    a2=((cos(u)-cos(2*u))/2)

    if FFT_IFFT==1

        a3=i*(sin(u)+sin(2*u))
        a4=i*sin(2*u)
        a5=i*(sin(u)-sin(2*u))

    else
        a3=-i*(sin(u)+sin(2*u))
        a4=-i*sin(2*u)
        a5=-i*(sin(u)-sin(2*u))

    end

    m1=a1*s5
    m2=a2*s6
    m3=a3*s2
    m4=a4*s7
    m5=a5*s4

    s9=s8+m1
    s10=s9+m2
    s11=s9-m2
    s12=m3-m4
    s13=m4+m5
    s14=s10+s12
    s15=s10-s12
    s16=s11+s13
    s17=s11-s13

    XF(1,1)=s8;
    XF(1,2)=s14;
    XF(1,3)=s16;
    XF(1,4)=s17;
    XF(1,5)=s15;

    if FFT_IFFT==1
```

```
       % XF=XF/5;
        fft_m=ifft(x)*5;
    else
        fft_m=fft(x);
    end
 end
```

Script convertidor de decimal a binario.

```
function [X_bin]=decimal2binario(X_dec)

    [f,n]=size(X_dec);

    n;

    for i=1 : n

       W=real(X_dec(1,i));

       if real(X_dec(1,i)) < 0

           A=2^9-real(X_dec(1,i))*-1+2^9;

       else

           A=real(X_dec(1,i));

       end

       if imag(X_dec(1,i)) < 0

           B=2^9-imag(X_dec(1,i))*-1+2^9;

       else

           B=imag(X_dec(1,i));

       end

       X= dec2hex(round(A*1024),5);
       Y = dec2hex(round(B*1024),5);

       Z =imag(X_dec(1,i));

       X_bin(i,:)=strcat(X,'..... ',Y) ;
    end

end
```

Apéndice B.

CIINDET 2015

XI Congreso Internacional sobre Innovación y Desarrollo Tecnológico, 25 al 27 de marzo, Cuernavaca Morelos, México.

Implementación de la Transformada Discreta de Fourier IFFT/FFT en un sistema embebido para un sistema de comunicación OFDMA.

F. Mora Hernández, J. Castañeda Camacho, S. Vergara Limón, E. Ríos Silva, J. E. M. Gutiérrez Arias.

Resumen: En el presente trabajo se implementó la Transformada Rápida de Fourier (FFT) y su contraparte la Transformada Inversa Rápida de Fourier (IFFT) para un sistema de comunicaciones de Acceso Múltiple por División de Frecuencias Ortogonales (OFDMA) que se realizó en un FPGA de Altera de la familia ciclon III, para lo cual se tomaron en cuenta los diversos métodos existentes para la correcta selección del más adecuado con el fin de utilizar correctamente los recursos disponibles en el FPGA, ya que estos son limitados y la obtención de FFT e IFFT requiere muchos recursos computacionales que de no ser correctamente administrados ocasionan la necesidad de recursos excesivos que hablando de un FPGA se traduce en la cantidad de elementos lógicos y registros necesarios.

Abstract: In this work is implement the Fast Fourier Transform (FFT) and its counterpart the Fast Fourier Transform Inverse (IFFT) for a communication system Orthogonal frequency-division multiplexing (OFDMA) was implemented in FPGA Altera Cyclone III family for which we took into account the various methods for the correct selection of the most appropriate in order to properly use the resources available in the FPGA since these are limited and obtaining FFT and IFFT is resource computer which if not properly managed resulting in the need for excessive talking resources of an FPGA results in the number of logic elements and registers necessary .

Keywords: FFT, IFFT, FPGA, OFDMA.

Francisco Mora Hernández, Benemérita Universidad Autónoma de Puebla, Av. san Claudio y 18 Sur, Ciudad Universitaria, México, francisco.mora.h@gmail.com.
Josefina Castañeda Camacho, Benemérita Universidad Autónoma de Puebla, Av. san Claudio y 18 Sur, Ciudad Universitaria, México, josefinacastaneda@yahoo.com.mx.
Sergio Vergara Limón, Benemérita Universidad Autónoma de Puebla, Av. san Claudio y 18 Sur, Ciudad Universitaria, México, svergara@ece.buap.mx.
Eduardo Ríos Silva, Benemérita Universidad Autónoma de Puebla, Av. san Claudio y 18 Sur, Ciudad Universitaria, México, erios@ece.buap.mx.
Jose Emilio Gutierrez Arias, Benemérita Universidad Autónoma de Puebla, Av. san Claudio y 18 Sur, Ciudad Universitaria, México, jmgutierrez@kim.ece.buap.mx
.

Introducción.

La gente está cada vez más acostumbrada a comunicarse en cualquier lugar, en cualquier momento y de cualquier forma, este patrón de comunicación va acompañada de la creciente demanda de acceso inalámbrico de banda ancha móvil. A medida que aumenta la demanda, los proveedores de acceso a la red inalámbrica y los proveedores de servicios de red están implementando sistemas de última generación que pueden soportar el flujo de datos de alta velocidad [1].
Hoy en día la cantidad de usuarios de comunicaciones inalámbricas ha ido en aumento y no solo eso, sino la cantidad de transferencia de datos ha crecido drásticamente haciendo de gran importancia la renovación de los sistemas de comunicaciones para poder satisfacer dichas demandas, una de las tecnologías que ha tenido gran aceptación es la Multiplexación por División de Frecuencias Ortogonales (OFDM) que permite la transmisión de grandes cantidades de datos digitales a través de una onda de radio la cual ha tenido gran aceptación en Europa para la transmisión de televisión y radio digital, pero a nivel mundial la atención hacia OFDM y la versión multiusuario de esta OFDMA ha ido en aumento haciendo de estos claramente la tendencia a seguir para los próximos años [1].

La Trasformada Rápida de Fourier (FFT).

En Abril de 1965 se publicó en la revista "*Mathematics of Computation*" un pequeño artículo de apenas 5 páginas sin grandes pretensiones titulado "*An algorithm for the machin calculation of complex Fourier series*" [2]. Poco después de la publicación de este artículo la comunidad científica que trabajaba en el procesamiento digital de señales se concentró alrededor de este algoritmo, lo cual produjo en muy poco tiempo una avalancha de aplicaciones para este, consecuencia directa de acortar significativamente la brecha entre el

IEEE Sección Morelos

CIINDET
HACIA UN DESARROLLO SUSTENTABLE CON TECNOLOGÍA PROPIA

CONSTANCIA

Artículo: ***"Implementación de la Transformada Discreta de Fourier IFFT/FFT en un sistema embebido para un sistema de comunicación OFDMA."***

Autores: **Francisco Mora Hernandez, Josefina Catañeda Camacho, Sergio Vergara Limón**

Id. artículo: **49**
Área: **Electrónica e Instrumentación**

El Comité Técnico del XII Congreso Internacional Sobre Innovación y Desarrollo Tecnológico CIINDET 2015, que se llevó a cabo en la ciudad de Cuernavaca, Morelos, México, del 25 al 27 de marzo de 2015, hace constar que el artículo citado fue presentado de acuerdo con el programa técnico del congreso e incluido en las memorias del mismo.

La presente constancia se expide para los fines legales que a los autores convengan.

Cuernavaca, Morelos, México a 27 de Marzo de 2015.

Atentamente

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC. FOUNDED NEW YORK 1884

CONGRESO INTERNACIONAL SOBRE INNOVACIÓN Y DESARROLLO TECNOLÓGICO CIINDET

Dr. Jorge Guillermo Calderón Guizar
Presidente del Comité Técnico CIINDET 2015

CIINDET 2015
HACIA UN DESARROLLO SUSTENTABLE CON TECNOLOGÍA PROPIA

XII CONGRESO INTERNACIONAL SOBRE INNOVACIÓN Y DESARROLLO TECNOLÓGICO

26 AÑOS 1989-2015 IEEE SECCION MORELOS ANIVERSARIO

La Sección Morelos del Instituto de Ingenieros en Electricidad y en Electrónica

Otorga la presente

Constancia

A

Francisco Mora Hernández

Por su participación como

CONGRESISTA

SRI 2015 Energizando con Inteligencia

Durante el XII Congreso Internacional sobre Innovación y Desarrollo Tecnológico, realizado del 25 al 27 de marzo del 2015, en la ciudad de Cuernavaca, Morelos, México.

Dr. Francisco Ponce Maldonado
Presidente IEEE Sección Morelos

Dr. Rafael Castellanos Bustamante
Presidente del Ciindet 2015

Apéndice C.

TOEFL ITP Score Report

Name of Institution: FACULTAD DE LENGUAS DE LA BUAP

Name: MORA FRANCISCO

Student Number:

DOB: 04/08/1985 Sex: M Degree:

Times Taken TOEFL: 1

Native Country: Mexico
Native Language: Spanish

Scaled Scores:

Listening Comprehension:	44
Structure & Written Expression:	46
Reading Comprehension:	54
Total Score:	480

Test Date: 04/24/2015
Form: TOEFL ITP

ETS TOEFL ITP

Student's File Copy
Do Not Copy

The face of this document has a security background. The back contains a watermark. Hold at an angle to view.

The *TOEFL® ITP* Assessment Series is designed to be used for placement, progress monitoring, and exit purposes. *TOEFL® ITP* scores can also be used for admissions to programs and institutions where English is not the dominant language of instruction for content courses. Learn more at **www.ets.org/toefl_itp/use.**

106132-16573 • FB614R100 • Printed in U.S.A. I.N. 770462