UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

FACULTAD DE INGENIERÍA

"Rediseño de IPCORES FFT y STFT basado en FPGA"

TESIS

Que como parte de los requisitos para obtener el título de

INGENIERO ELECTROMECÁNICO

Presenta:

Jean Karlo Gómez Reyes

Director

Dr. Roque Alfredo Osornio Ríos

Co-Asesor

Dr. Luis Morales Velázquez

San Juan del Río, Qro. Noviembre 2015

En especial dedico este trabajo a mi abuelo Esteban Reyes Alcázar, quien fue mi inspiración para cursar esta carrera y a quien le debo las gracias por apoyarme siempre y animarme a seguir, a no ir por lo fácil y realizar siempre un trabajo honesto.

A mi madre Patricia que nunca dejó de creer en mí y darme los medios necesarios para culminar esta etapa. A mis hermanos Michel y Mariel por ser ejemplos a seguir y apoyarme siempre. A mi padre que me presto su ayuda.

AGRADECIMIENTOS

Le agradezco principalmente a Dios por todas las bendiciones que me otorgó durante esta etapa de formación, por las personas que conocí y que hoy forman parte de mí, mis amigos, compañeros y profesores.

A mi familia que siempre me ha acompañado, les agradezco a todos mis tíos y primos que has estado conmigo y me han ayudado en lo necesario.

A la Universidad Autónoma de Querétaro por brindarme la oportunidad de realizar esta travesía y a la facultad de Ingeniería por darme las herramientas necesarias para mi desarrollo profesional.

Al Dr. Roque por la oportunidad de realizar este proyecto, por sus consejos y sugerencias para la elaboración de esta tesis, al Dr. Luis Morales por su gran ayuda y guía en la realización de este trabajo, Dr. Arturo Yosimar por aconsejarme durante esta etapa y al Dr. J Jesús de Santiago por apoyarme. A todos les agradezco por prestarme su valioso tiempo por sus comentarios y consejos.

Y a mis amigos Daniel, Adrián, Ismael y a todos mis amigos de la Universidad muchas gracias por su amistad.

Contenido

Índice de Figuras

CAPÍTULO 1

1. Introducción.

En la actualidad el análisis por medio de la frecuencia es una gran herramienta para el procesamiento de señales puesto que tiene una amplia gama de aplicaciones como: medicina (Banerjee et al., 2001), arqueología (Salazar & Vergara, 2010), sistemas (Cabal-Yepez et al., 2012), maquinado (Molino et al., 2009), etc. Por lo que se ha tenido la necesidad de crear y mejorar herramientas que permitan realizar este proceso de discretización, un ejemplo es la transformada discreta de Fourier (DFT), que proporciona información de la señal al pasar del dominio del tiempo al de frecuencia.

El análisis en frecuencia permite extraer información que no es evidente mediante la simple observación de una señal en el tiempo. La transformada discreta de Fourier (DFT) es la respuesta natural e inmediata a la transformada continua de Fourier en el mundo digital (CFT), pero con limitaciones de tiempo de ejecución debido a la casi total ausencia de optimización de dicho algoritmo (Cortes et al., 2010).

Debido a las limitaciones de velocidad y procesamiento de la DFT se han propuesto diferentes métodos para simplificar el cálculo y disminuir el número de operaciones obteniendo así un resultado de forma más rápida, por esto se formularon algoritmos para calcular la FFT (Fast Fourier Transform). Pero en ocasiones es necesario conocer el instante en el tiempo en el cual hay variaciones de la señal en frecuencia, por lo que se utiliza la STFT (Short Time Fourier Transform) que permite visualizar la información de la FFT, pero en pequeños lapsos o ventanas de tiempo, así se monitorea la señal de forma más eficiente y se pueden localizar los cambios. Para utilizar estas herramientas matemáticas en procesos reales existen diferentes sistemas comerciales un ejemplo es el software LabVIEW, que ofrece diferentes herramientas para la adquisición de datos, medidas y sistemas de control para diferentes procesos, pero lamentablemente a un alto costo, además de que se obtiene por módulos y si se necesita cambiar de aplicación es necesario un equipo extra y una licencia diferente, lo que aumenta aún más los costos.

La Universidad Autónoma de Querétaro (UAQ) buscó una alternativa más económica pero eficaz para realizar este proceso además de independizarse de tecnología

extranjera, por lo que se procedió a la creación e implementación de Núcleos de Propiedad Intelectual o IP cores que se puedan unir con algunos otros para procesos más complejos , mediante la creación de bloques lógicos que han sido programados en tarjetas FPGA (Field Programmable Gate Array), sin embargo, en el caso de los algoritmos de la FFT y la STFT se han generado diferentes softwares para diversas aplicaciones, pero aún se tiene la necesidad de generar un IP core que contenga el algoritmo de forma genérica de manera que sea posible ensamblarlo con otros para aplicaciones más específicas sin tener que mover parámetros fundamentales, la ventaja de un IP core es que puede ser usado en distintos diseños sin necesidad de modificar su código un ejemplo es el trabajo de Mora, (2010) quien presento un IP core de un controlador PID con conexión USB, para controlar un torno CNC, pero al ser un Core es genérico y se puede reutilizar en una máquina herramienta distinta.

Esta tesis está dividida en cinco capítulos principales, en el primer capítulo se presentan los antecedentes, objetivos, justificación y planteamiento general. En el segundo capítulo se desarrollan los conceptos necesarios para abordar el tema y los módulos a utilizar. La metodología y desarrollo se presentan el tercer capítulo además de la implementación de estos. En el cuarto capítulo se presenta las pruebas de los módulos así como los resultados y comparaciones. Las conclusiones de este trabajo se presentan en el quinto capítulo. Además se anexarán los diseños de los Cores así como la guía para el uso de estos.

1.1. Antecedentes.

Los primeros avances sobre el procesamiento de señales se dieron en el siglo XVII con la invención del cálculo, lo que permitió a los científicos e ingenieros desarrollar modelos para representar los fenómenos físicos en términos de funciones de variables continuas y ecuaciones diferenciales. (Oppenheim et al., 1999).

En relación a lo anterior una de las aplicaciones se da en el procesamiento de señales la cual abarca una gran cantidad de funciones como son, el monitoreo de sistemas

mecánicos, los sistemas de comunicación, la medicina, la acústica, el procesamiento de imágenes, la robótica, los sistemas eléctricos, etc. (Nasser & Sidharth, 2010).

En la actualidad la tecnología y los sistemas digitales han mejorado de forma significativa permitiendo el monitoreo de señales en tiempo real, sin embargo, no todos los dispositivos de medición entregan un resultado completo, por eso la necesidad de filtrar y mejorar una señal obtenida mediante un procesamiento de ésta, para lo cual se utiliza la FFT, este algoritmo permite analizar el espectro en frecuencia de la señales que nos interesan medir y obtener una visualización más completa sobre el estado de la señal. Otra de las técnicas es la STFT, esta herramienta es similar a la FFT pero su diferencia radica en que permite comparar los datos de la frecuencia contra el dominio del tiempo de la señal analizada para así identificar en qué momento se presentan los cambios en el espectro de frecuencia. La FFT permite filtrar la señal de interés mediante la obtención de la DFT de forma más rápida reduciendo el número de operaciones, obteniendo el espectro de la frecuencia y filtrando las señales de menor frecuencia permitiendo el monitoreo del sistema en tiempo real de forma que se pueda evidenciar el error que permita ajustar los parámetros para realizar una acción de control adecuada o procesar el valor de frecuencia para diferentes operaciones.

Con la tecnología de hoy es posible generar códigos de programación que realicen la gran carga de operaciones para obtener una transformada en tiempo real, mejorando de esta manera el procesamiento de señales, de ahí que diferentes compañías generen y vendan su propio software/hardware para ciertas aplicaciones de monitoreo: el problema es cuando se requiere utilizar ese paquete en ciertas aplicaciones, pues la mayoría de las veces queda limitado y no se obtienen los resultados deseados, es por esto que se busca obtener programas que realicen el proceso sin importar el sistema en el que sea implementado.

Anteriormente se buscaba minimizar el número de operaciones con punto flotante realizadas por el algoritmo pero con los microprocesadores actuales esta medida es menos importante, la interacción del procesador de arquitectura pipeline y la jerarquía de memoria tienen un gran impacto en el rendimiento, por lo que es necesario conocer la arquitectura de la computadora con la que se diseñará el software para obtener un algoritmo rápido y tener así una respuesta más rápida del sistema o hardware acorde al

programa a utilizar aunque también se ha buscado tener software que se puedan adaptar a diferentes sistemas .En su trabajo Frigo & Johnson, (1998) propusieron un programa adaptable de la FFT para diferentes sistemas y obtuvieron un resultado favorable comparando su software con diferentes sistemas comerciales en 7 máquinas, demostrando así las ventajas de la implementación de software genérico y adaptable. Pero a pesar de la versatilidad de este código, sigue siendo lento en comparación a la integración correcta de software y hardware.

En el campo de la matemática existen diferentes tipos de transformaciones y variables, pero en cuanto al análisis de señales y la transformación de estas las variables de interés están limitadas al tiempo, frecuencia, fase y amplitud. Al utilizar la herramienta de la transformada de Fourier de tiempo reducido (STFT) se puede realizar un análisis de frecuencia contra tiempo en un arreglo de ventana tiempo-frecuencia.

El algoritmo de la STFT tiene la ventaja de dar una perspectiva más completa sobre la señal pero a la hora de programar también es necesario el uso de memoria para mayor rapidez en el cálculo, Qin & Zhong, (2004), realizó un modelo unificado para simplificar el cálculo de la FFT y la STFT de tal forma que pudiera ser utilizado para diferentes señales y transformaciones incluyendo en su trabajo la transformada Wavelet, mostrando un método más simple de calcular estas funciones.

Un software adaptable otorga la ventaja de su implementación en diferentes computadoras pero si se utiliza un sistema especializado la comunicación entre estos dos genera un sistema más eficaz. Existen diferentes sistemas comerciales especiales para la adquisición y procesamiento de señales, ejemplos de esto son National Instruments (NI) que tiene diferentes sistemas de análisis de señales y el trabajo de Kıymık et al. (2005), que con el fin de obtener más información de las señales de un encefalograma realizaron una comparación entre la STFT y la transformada Wavelet para obtener un análisis más completo sobre el espectro de las señales del estudio de encefalograma descomponiéndola en sub espectros que conforman la señal y así obtener las ondas cerebrales α, θ, δ y β;en el utilizaron el lenguaje gráfico de National Instruments (NI) LabVIEW para programar las transformadas, junto con una tarjeta de adquisición de “NI”, la conclusión de su trabajo comprobó que la STFT tenía mejores resultados en cuanto a tiempo de procesamiento además de la ventaja de ser aplicable

en línea pero teniendo una desventaja de exactitud en comparación a la transformada Wavelet que tardo más tiempo pero mejoró la exactitud de detección de ondas.

La ventaja de utilizar un sistema software-hardware compatible es que permite realizar diferentes aplicaciones en sistemas de bajo costo tales como las FPGA que posibilita realizar cambios en el software sin modificar el hardware y su aplicaciones son bastantes, ejemplo de ellas es el trabajo de Banerjee et al.(2001), quienes diseñaron un algoritmo CORDIC (Coordinate Rotation DIgital Computer, Computadora Digital para Rotación de Coordenadas) de la FFT implementado en una FPGA (Field Programmable Gate Array, Arreglo de compuertas programables) ahorrando memoria del procesador sin reducir la velocidad de procesamiento del sistema. Esto con la finalidad de monitorear señales Biomédicas como ecografías, electromiografías, electrogastrograma y electrocardiograma de forma rápida y efectiva además de ser una solución más rentable en cuanto al procesamiento.

Dentro del país se han realizado diferentes trabajos con impacto internacional, para la optimización de la FFT en sistemas de maquinado buscando formas de apoyar a la industria, algunas aplicaciones son la detección rápida de fallas en motores como las fracturas en la barra del rotor ya que esta falla provoca un mayor consumo de energía y es responsable de un daño mayor a la máquina. En su trabajo Rangel et al. (2009), propusieron un método para el análisis del estado de la barra de rotor con la ayuda de la implementación de dos sistemas de la FFT de 1024 puntos de relación para la obtención de un espectro más exacto, realizaron sus pruebas con motores sin carga y con carga para obtener una comparación de resultados obteniendo buenos resultados de exactitud.

Otro antecedente relacionado a este trabajo es el de Cabal-Yepez et al. (2013), quienes propusieron un análisis de frecuencia y tiempo para el monitoreo de la señal de tres sistemas uno el suministro de corriente eléctrica de un motor de inducción , el voltaje suministrado a un motor de inducción controlado por un variador de velocidad, y las vibraciones en la uniones de un robot industrial, mediante la implementación de la STFT y la transformada Wavelet en tiempo discreto (DWT) en un FPGA y así comparar las señales obtenidas y mostrar el resultado en una interfaz gráfica.

Los trabajos antes mencionados evidencian la necesidad de contar con IP cores genéricos, ya que estos hasta ahora utilizan los algoritmos de forma particular para cada aplicación. El contar con un IP core facilita el diseño de software para procesos específicos reduciendo el tiempo y creación de herramientas similares.

1.2. Objetivos.

1.2.1. *Objetivo General.*

Rediseñar y optimizar una aplicación de la FFT y STFT para generar un IP core sencillo de conectar con diferentes archivos para diseños más complejos sin la necesidad de editar el archivo fuente, al mismo tiempo crear la documentación adecuada para evitar errores por falta de información

1.2.2. *Objetivos Particulares.*

- Mejorar los IP CORES FFT y STFT basados en un estándar de diseño para ser utilizados en diferentes aplicaciones Mecatrónicas futuras y que pueda ser utilizado simplemente como librería en otros proyectos que requieran integrarlo.
- Desarrollar un manual de funcionamiento del IP Core para facilitar al diseñador las propiedades del Core de forma que si necesita conocer ciertas características el programa pueda encontrar las especificaciones necesarias en la documentación.
- Presentar las pruebas de funcionamiento de los IP Cores para evidenciar los cambios y la fidelidad con la que trabaja el Core

1.3. Justificación.

La UAQ se ha caracterizado por el desarrollo de proyectos y automatización de procesos relacionados con máquinas herramienta, robótica, sistemas eléctricos, etc. Para lo anterior, se han desarrollado diferentes IP cores que controlan y monitorean señales que se requieren en estos procesos y se han realizado con el conocimiento y tecnología propia buscando economizar así los gastos e independizarse de sistemas comerciales. En el caso de la FFT no se cuenta con un IP core propio si no con arquitecturas del algoritmo para procesos específicos, de ahí que sea una necesidad la creación de este núcleo, pues facilitaría el diseño de sistemas sin necesidad de crear el algoritmo para cada aplicación.

Este IP core se realizará utilizando el estándar propuesto por OpenCores, de forma que sea compatible para diversos procesos ya que dicho estándar es muy utilizado al momento de realizar estos núcleos, además de tener la ventaja de ser un estándar internacional. La documentación será generada por el software Doxygen el cual proporciona un modelo específico de escritura para los documentos creados.

La creación de un IP core generaría un gran avance tecnológico en la universidad, pues permitiría entrar en el rango de competencia sistemas comerciales con herramientas similares, propiciando así la independencia a estos sistemas.

El propósito de esta tesis es diseñar mediante una guía formal los IP cores de la FFT y STFT para su utilización en aplicaciones Mecatrónicas que lo requieran. De esta forma se facilita la información de estos sistemas para diferentes aplicaciones que requieran la implementación de éstos, también proveerá la estructura completa y detallada del software, además de las variables y procesos que realizará el programa.

Otra gran ventaja es la independencia a sistemas comerciales que además de su elevado costo, si se desea unir a otra aplicación u otro proceso requiere de adquirir los demás sistemas, aumentando así el precio total, mientras que al utilizar los sistemas de bajo costo como lo son las tarjetas FPGA y los IP cores es posible ensamblarlos con otros bloques para generar un diseño más completo para las aplicaciones que se requieran.

También es un gran avance tecnológico dentro de la UAQ pues se generarán herramientas que facilitan el análisis para diferentes proyectos que requieran más adelante un procesamiento de señales o trabajos relacionados en lo que estos bloques faciliten la investigación simplificando el monitoreo de señales.

1.4. Planteamiento general

Para realizar la Documentación de los IP cores como base de datos de la Facultad de Ingeniería de la Universidad Autónoma de Querétaro se utilizarán diferentes herramientas, para mantener un estándar, tales como Doxygen y el sistema de textos LaTeX que permiten la creación de textos de alta calidad tipográfica para más tarde realizar un documento PDF que cuente con las características principales de diseño de cada IP core mostrando los elementos más significativos o mínimos necesarios para la correcta implementación del CORE, esta documentación se realizará mediante el estándar propuesto por la organización OpenCores, de manera que sea un estándar más universal, en la Figura 1 se muestra el esquema de las herramientas ya mencionada que se utilizarán para la documentación completa de los IP cores, además se tiene la hipótesis de mejorar el diseño de los Cores de la FFT y la STFT para reducir tiempo de procesamiento, obtener un código compacto y genérico

Figura 1 Generación de documentos mediante codigo

CAPÍTULO 2

2. Revisión de literatura.

2.1. Estado del arte.

En la actualidad los sistemas digitales permiten realizar un procesamiento de señales para distintos motivos, el principal es el monitoreo de señales para así conocer el estado del sistema que emite la señal esto tiene muchas aplicaciones en la física y en la ingeniería. Es posible utilizar la DFT para actividades de procesamiento de audio, señales biomédicas y en la industria para conocer la calidad de la energía de una máquina herramienta además de poder identificar fallas previniendo un problema mayor a éstas.

2.2. FFT

La Transformada Rápida de Fourier o FFT es una herramienta que permite calcular de forma rápida la transformada Discreta de Fourier y su inversa la transformada inversa de Fourier discreta (IDFT). La FFT es de gran importancia por su gran variedad de aplicaciones digitales, desde el tratamiento digital de señales y filtrado digital en general, a la resolución de ecuaciones diferenciales parciales o los algoritmos de multiplicación rápida de grandes enteros.

La transformada de Fourier es una transformación matemática que permite pasar señales entre el dominio del tiempo y el dominio de la frecuencia y se debe al trabajo del físico y matemático Jean Baptiste Joseph Fourier (1768-1830), en su trabajo expresaba que “cualquier señal continua y periódica podía representarse como la suma una serie de ondas senoidales adecuadamente elegidas”. En la Figura 2 se muestra el efecto de la aplicación de la DFT y la IDFT

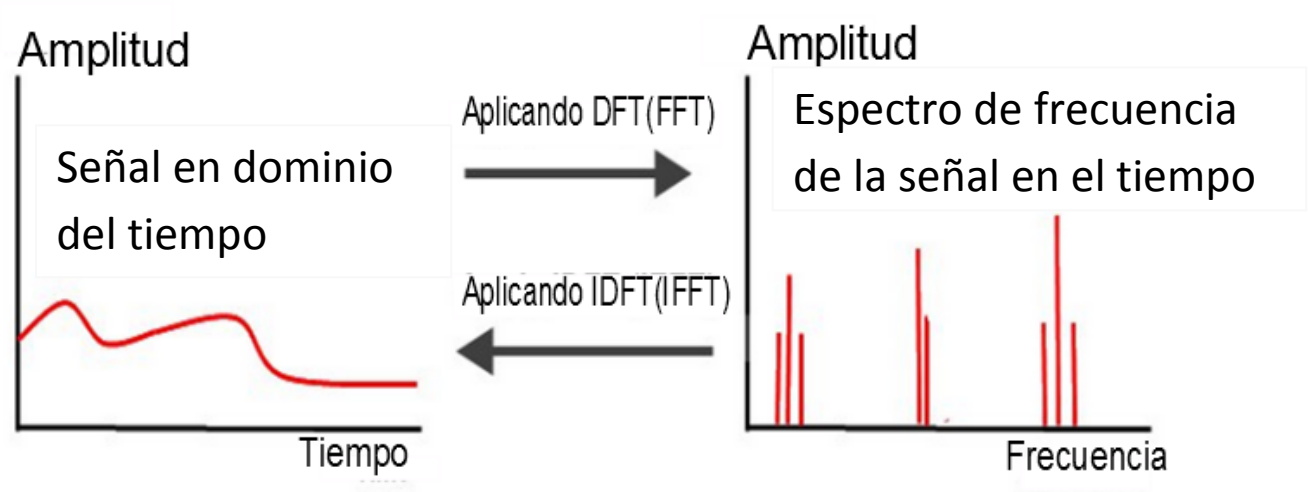


Figura 2 Gráficas en dominio de tiempo y de frecuencia

2.2.1 Series de Fourier

Una función en el dominio del tiempo muestra la variación de la amplitud de una señal en el tiempo la representación en el dominio de la frecuencia permite conocer que tan a menudo se presentan estos cambios. El paso del domino de tiempo al de frecuencia se puede visualizar considerando que la señal en estudio está compuesta por la suma de ondas sinusoidales simples de amplitud y fase adecuadas o de exponenciales complejas relacionadas armónicamente.

La herramienta matemática que permite pasar del dominio del tiempo al dominio de la frecuencia es la Serie de Fourier para las señales periódicas, y de la Transformada de Fourier para las señales de energía finita.

La transformada de Fourier ofrece la posibilidad de representar una función f definida en un conjunto de números reales $\mathbb{R}$ como una serie trigonométrica de la forma mostrada en la ecuación 1.

$$f(x) = \sum_{k=0}^{\infty} (a_k \cos kx + b_k sen\, kx) \qquad (1)$$

Esta es una función periódica de periodo 2π de modo que f debe tener también esta propiedad. De esta forma basta con conocer la función en un intervalo de 2π de las relaciones de Euler.

$$e^{jkF_0t} = \cos k\omega_0 t + jsen\, k\omega_0 tx \quad (2)$$

Los componentes frecuenciales individuales son conocidos como armónicos.

Lo que permite reescribir la ecuación además de tomar funciones de periodo π en lugar de 2π lo que modifica el sistema representando la ecuación 1 como:

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j2\pi k\omega_0 t} \quad (3)$$

En la tabla 1 se muestra la comparación entre ecuaciones para el cambio de dominio de tiempo a frecuencia y viceversa de las series de Fourier :

Tabla 1 Series de Fourier

TIEMPO CONTINUO		
Serie de Fourier	Dominio del tiempo	Dominio de la frecuencia
$T_0 = \frac{2\pi}{\omega_0}$	$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j2\pi k\omega_0 t}$	$a_k = \frac{1}{T_0}\int_{T_0} x(t)e^{-j2\pi k\omega_0 t}$
TIEMPO DISCRETO		
Serie de Fourier	Dominio del tiempo	Dominio de la frecuencia
$N = \frac{2\pi}{\Omega_0}$	$x[n] = \sum_{k=(N)} c_k e^{jk\left(\frac{2\pi}{N}\right)n}$	$a_k = \frac{1}{N}\sum_{n=<N>} x[n]e^{jk\left(\frac{2\pi}{N}\right)n}$

2.2.2. Series Discretas de Fourier

Considerando una secuencia $x[n]$ periódica con un periodo N de forma $x[n] = \mathrm{x[n + rN]}$ para cualquier valor entero de *n* y *r*. Esta señal puede representarse mediante la serie correspondiente de Fourier de sumas armónicas de las secuencias compleja exponencial, las exponenciales complejas con frecuencias que son múltiplos enteros de la frecuencia fundamental $(2\pi/N)$ asociada con la secuencia periódica $x[n]$. Esta exponencial compleja periódica es de la forma expresada en la ecuación 4.

$$e_k[n] = e^{j(2\pi/N)kn} = e_k[n + rN] \quad (4)$$

Donde *k* es un entero, y la serie de Fourier tiene la forma mostrada en la ecuación 5.

$$x[n] = \sum_{k} c_k e^{jk(2\pi/N)n} \quad (5)$$

2.2.3. Transformada de Fourier

En las aplicaciones reales las señales muy pocas veces son periódicas por lo que se debe realizar una transición de las series para analizar estas señales. La transformada de Fourier emerge a cuando se pasa de señales periódicas a funciones no periódicas. Para el ejemplo se utiliza una función no periódica (que puede ser casi cualquier señal). Como un límite de la función se toma el periodo más las largo. En realidad, este proceso no produce inmediatamente el resultado deseado. Si se considera la serie de Fourier general en la cual todas las frecuencias son múltiplos de la fundamental de acuerdo a la ecuación 6.

$$\omega_k = k\omega_0 \quad (6)$$

El hecho de que la señal no es periódica se puede representar por la ecuación 7.

$$\omega_0 \to 0 \quad (7)$$

Esta ecuación señala la falta del mínimo común denominador entre las frecuencias de todos los fasores. Cuando el número de fasores tiende a infinito la sumatoria se convierte en la integral de la ecuación 8:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}\, d\omega \quad (8)$$

Para la transformada de Fourier la señal se puede definir como función de la frecuencia ω, es decir $X(\omega)$. La ecuación que define $X(\omega)$, viene dada por la ecuación 9.

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt \quad (9)$$

El análisis de frecuencias de las señales no periódicas de tiempo continuo se puede resumir en la tabla 2:

Tabla 2 Ecuaciones del análisis en frecuencia de señales no periódicas en el tiempo

Análisis en Frecuencia de señales no periódicas continuas en el tiempo	
Ecuación inversa de la transformada	$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}\, d\omega$ (8)
Ecuación Directa de la transformada	$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt$ (9)

Es evidente que la diferencia esencial entre la serie de Fourier y la transformada de Fourier es que el espectro en el último caso es continua y por lo tanto la síntesis de una señal aperiódica a partir de su espectro se logra por medio de la integración en lugar de la suma.

2.2.4. Transformada Discreta de Fourier (DFT)

La transformada Discreta de Fourier toma una secuencia de datos muestreados (señal) y calcula el contenido de frecuencia de esta señal representando estos datos en el dominio de la frecuencia. La transformada de Fourier para señales de tiempo discreto $x(n)$ está definida por la ecuación 10.

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} \qquad (10)$$

Físicamente $X(\omega)$ representa la frecuencia dentro de la señal $x(n)$. Por lo tanto $X(\omega)$ es la descomposición de $x(n)$ en las señales de diferentes frecuencias que la componen. Para encontrar el equivalente discreto de la transformada inversa de Fourier se debe transformar la variable continua *t* por la variable discreta nT_s. Fuera de los límites $\pm\pi/$

Ts, el espectro se repite, por lo tanto se puede cambiar los límites de este valor, como se muestra en la ecuación 11.

$$x(n) = \frac{1}{2\pi}\int_{2\pi} X(\omega)e^{j\omega n}d\omega \qquad (11)$$

El espectro obtenido al usar esta transformada tiene la propiedad de ser periódica. También para una señal real, como los fasores aparecen en forma de complejos conjugados, el espectro siempre tiene simetría par a lo largo del eje real, y simetría impar en el eje imaginario. Esto significa que si se sabe que se trabaja con una señal real, la cantidad de información que se necesita recordar sobre el espectro en frecuencias es menor, ya que es repetitivo. Las ecuaciones de la DFT se presentan a continuación en la tabla 3:

Tabla 3 Transformada Discreta de Fourier

Análisis frecuencial de señales en tiempo discreto	
Transformada inversa	$x(n) = \frac{1}{2\pi}\int_{2\pi} X(\omega)e^{j\omega n}d\omega$
Transformada Directa	$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}$

2.2.5. Algoritmo de la FFT

La DFT es una herramienta muy útil, sin embargo, al implementar esta ecuación se involucra un número de sumas y multiplicaciones complejas que es proporcional a N^2. Lo anterior se puede apreciar fácilmente ya que para cada uno de los *N* valores de μ, la expansión de la sumatoria requiere *N* multiplicaciones complejas de $f(x)$ por $e^{-j2\pi n/N}$ y $(N-1)$ sumas de resultados. El término $e^{j2\pi n/N}$ puede ser calculado de una vez y almacenado en una tabla para las aplicaciones subsecuentes. El ahorro o reducción en el número de operaciones al utilizar el algoritmo de la FFT es significativo para valores de *N*. Si se tiene un número de elementos *N* de 1024.

Sí $N = 1024$, se necesitaran $1.048.576$ operaciones complejas ya que son proporcionales a N^2 . Pero en cambio al utilizar el algoritmo de la FFT se reduce el número de operaciones complejas a 10.240 pues estas están dadas por la operación logarítmica de $Nlog_2N$.

Con una reducción de 102.4:1, el tiempo de cómputo, empleando máquinas equivalentes, se reduce a menos del 1%. La Transformada de Fourier de dos variables puede ser calculada por aplicación sucesiva de la Transformada de Fourier de una variable.

El término de transformada rápida de Fourier (FFT) abarca distintos algoritmos con distintas características, ventajas y desventajas. Por ejemplo, una FFT diseñada y optimizada usando un lenguaje de alto nivel probablemente no funcionará correctamente en un procesamiento digital de señales (DSP) de coma fija. Sin embargo, todas las FFT's usan la misma aproximación para reducir el algoritmo en un número reducido de DFT's sucesivas, cortas y simples. El truco para un cálculo más rápido de la DFT de orden N es utilizar esa estructura para reorganizar los productos para llevar a la DFT de orden $N / 2$ .Se asume que *N* es par.

El algoritmo que se plantea está basado en el método denominado "doblamiento sucesivo". Para simplificar las expresiones, la ecuación 10 se reescribe en la ecuación 12 como:

$$F(k) = \sum_{n=0}^{N-1} f(x) W_n^{nk} \qquad 0 \leq k \leq N-1 \tag{12}$$

Donde Wn es

$$W_n = e^{-j2\pi/N} \tag{13}$$

En general, la secuencia de datos *x(n)* también se supone que es de valor complejo. De manera parecida, la IDFT se convierte

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk} \qquad 0 \leq n \leq N-1 \tag{14}$$

Considere el cálculo de la DFT de $N = 2^v$ a partir de dividir la secuencia de datos de N puntos, en dos secuencias de $N/2$, correspondientes a las muestras pares e impares de $x[n]$, respectivamente, esto se expresa en las ecuaciones 15 y16:

$$f_1 = x[2n] \tag{15}$$

$$f_2 = x[2n+1], \quad n = 0,1,\dots,\frac{N}{2}-1 \tag{16}$$

Obsérvese, que se realizó el diezmado de la secuencia $x[n]$, una vez. La DFT de N puntos puede expresarse ahora en términos de las DFTs de las secuencias diezmadas partiendo de la ecuación (12)

$$X(k) = \sum_{n=0}^{N-1} X[n]W_n^{nk}$$

$$= \sum_{m=0}^{(N/2)-1} X[2m]W_n^{2mk} + \sum_{m=0}^{(N/2)-1} X[2m+1]W_n^{k(2m+1)} \tag{17}$$

Pero tomando $W_N^2 = W_{N/2}$. Sustituyendo esta igualdad en la expresión (17) se obtiene la ecuación 18:

$$X(k) = \sum_{m=0}^{(N/2)-1} f_1[m]W_{N/2}^{km} + W_N^k \sum_{m=0}^{(N/2)-1} f_2[m]W_{N/2}^{km}$$

$$= F_1(k) + W_N^k F_2(k), \qquad k = 0,1,\dots,N-1 \tag{18}$$

Donde $F_1(k)$ y $F_2(k)$son las DFTs de $N/2$ puntos de las secuencias de la expresión

Puesto que $F_1(k)$ y $F_2(k)$ son periódicas, de periodo $N/2$, se tiene $F_1(k+N/2) = F_1(k)\ y\ F_2(k+N/2) = F_2(k)$. Por otro lado, se cumple$W_N^{k+N/2} = -W_N^k$. Por lo que se puede rescribir la expresión (18) de la siguiente manera:

$$X(k) = F_1(k) + W_N^K F_2(k) \quad , k = 0,1,\dots.,\frac{N}{2}-1$$

$$X\left(k+\frac{N}{2}\right) = F_1(k) - W_N^K F_2(k) \quad , k = 0,1,\dots.,\frac{N}{2}-1 \tag{19}$$

Se observa que el cálculo directo de $F_1(k)$ requiere $(N/2)^2$ multiplicaciones complejas al igual que F_2 (k). Además, se requieren $N/2$ multiplicaciones más para calcular

$W_N^K F_2(k)$.De aquí que el cálculo de $X(k)$ requiere $N^2/2 + N/2$ multiplicaciones complejas. El primer pasó realizado de una reducción en el número de multiplicaciones de N^2 a $N^2/2 + N/2$, lo que equivale aproximadamente a dividir por dos el número de multiplicaciones cuando *N* es grande.

Habiendo realizado el diezmado en tiempo una vez, se puede repetir el proceso para cada una de las secuencias de la expresión (15). En la Figura 3 se muestra de forma gráfica el primer paso a realizar el diezmado en el tiempo del algoritmo de la FFT.

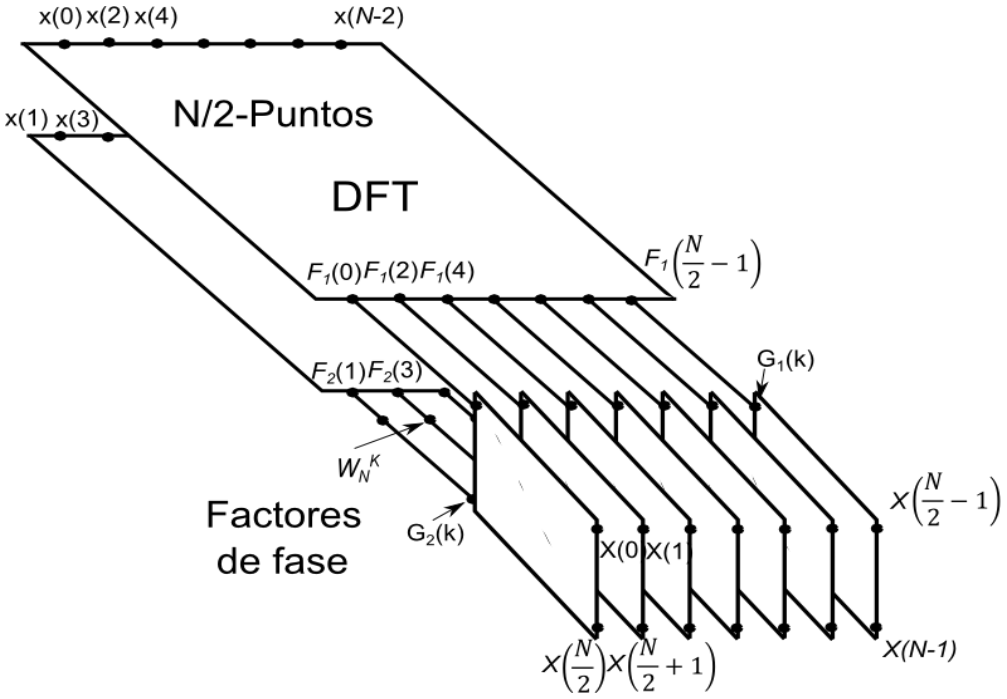


Figura 3 Primer paso del algoritmo de diezmado en tiempo (Proakis & Manolakis, 1996)

Por lo tanto, se obtendrá dos secuencias de N/4 puntos:

$$
\begin{aligned}
v_{11} &= f_1[2n], \qquad n = 0,1,\ldots,\frac{N}{4}-1 \\
v_{12} &= f_1[2n+1], \quad n = 0,1,\ldots,\frac{N}{4}-1 \\
v_{21} &= f_2[2n], \qquad n = 0,1,\ldots,\frac{N}{4}-1 \\
v_{22} &= f_2[2n+1], \quad n = 0,1,\ldots,\frac{N}{4}-1
\end{aligned}
\qquad (20)
$$

Calculando las DFT's de $N/4$ puntos se obtienen las DFTs de $N/2$ puntos $F_1(k)$ y $F_2(k)$a partir de las siguientes relaciones:

$$
\begin{aligned}
F_1(k) &= V_{11}(k) + W_{\frac{N}{2}}^{K} V_{12}(k), && k = 0,1,\ldots,\frac{N}{4}-1 \\
F_1\left(k+\frac{N}{4}\right) &= V_{11}(k) - W_{\frac{N}{2}}^{K} V_{12}(k), && n = 0,1,\ldots,\frac{N}{4}-1 \\
F_2(k) &= V_{21}(k) + W_{\frac{N}{2}}^{K} V_{22}(k), && n = 0,1,\ldots,\frac{N}{4}-1 \\
F_2\left(k+\frac{N}{4}\right) &= V_{21}(k) - W_{\frac{N}{2}}^{K} V_{22}(k), && n = 0,1,\ldots,\frac{N}{4}-1
\end{aligned}
\qquad (21)
$$

Donde $V_{ij}(k)$ son las DFT's de $N/4$ puntos de las secuencias $v_{ij}(n)$. Se observa que el cálculo de $V_{ij}(k)$ requiere $4(N/4)^2$ multiplicaciones y por lo tanto el calculo de $F_1(k)$ y $F_2(k)$ puede realizarse con $N^2/4 + N/2$ multiplicaciones complejas. Se requieren $N/2$ multiplicaciones complejas más para calcular $X(k)$ a partir de $F_1(k)$ y $F_2(k)$. Consecuentemente, el número total de multiplicaciones necesarias $N^2/4 + N/2$ se reduce otra vez aproximadamente por un factor de dos. El diezmado de la secuencia de datos se repite $v = \log_2 N$ veces, ya que se tienen $N = 2^v$ datos. Por lo tanto el número total de multiplicaciones complejas se reduce a $(\mathrm{N}/2)\log_2 N$, mientras que el número de sumas complejas es $\mathrm{N}\log_2 N$. En la Tabla 4 se muestra la comparación entre el número de multiplicaciones complejas usando la FFT y el cálculo directo de la DFT. En la Figura 4 se representa esquemáticamente el desarrollo del algoritmo de la FFT en base 2.

Tabla 4 Comparación de la DFT y la FFT

Comparación entre la cantidad de multiplicaciones complejas a realizar por parte de la DFT y el algoritmo FFT de base 2 (Proakis & Manolakis, 1996)			
Números de puntos, N	Multiplicaciones Complejas en calculo directo, N^2	Multiplicaciones Complejas en el algoritmo FFT, $(\mathrm{N}/2)\log_2 N$	Factor de mejora de la velocidad
4	16	4	4.0
8	64	12	5.3
16	256	32	8.0
32	1024	80	12.8
64	4096	192	21.3
128	16384	448	36.6
256	65536	1024	64.0
512	262144	2304	113.8
1024	1048576	5120	204.8

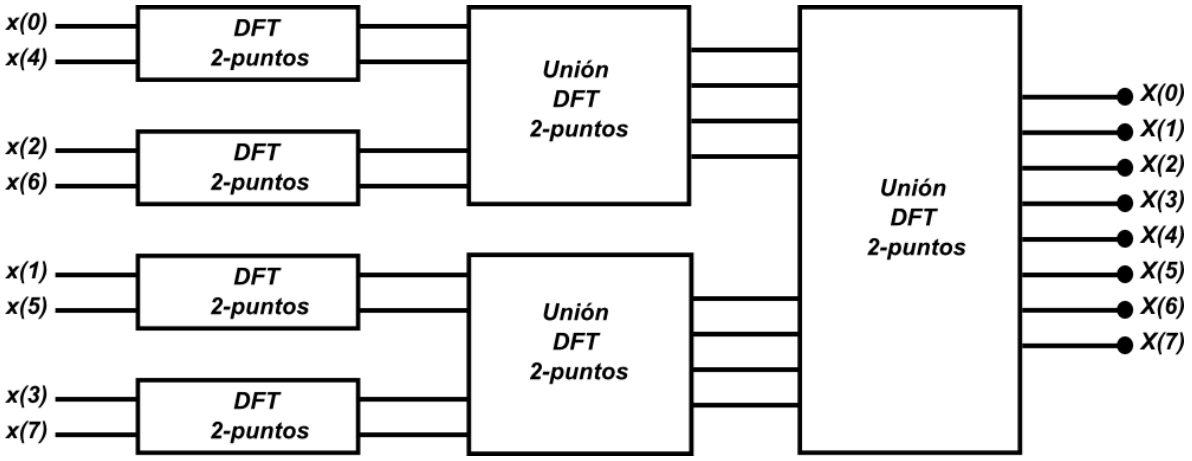


Figura 4 Tres etapas en el cálculo de la FFT de 8 puntos (Proakis & Manolakis, 1996)

Como puede observarse, el cálculo que se realiza en cada etapa, el cual consiste en aplicar las operaciones de una transformada DFT de dos puntos o "mariposa" como se muestra en la Figura 5.

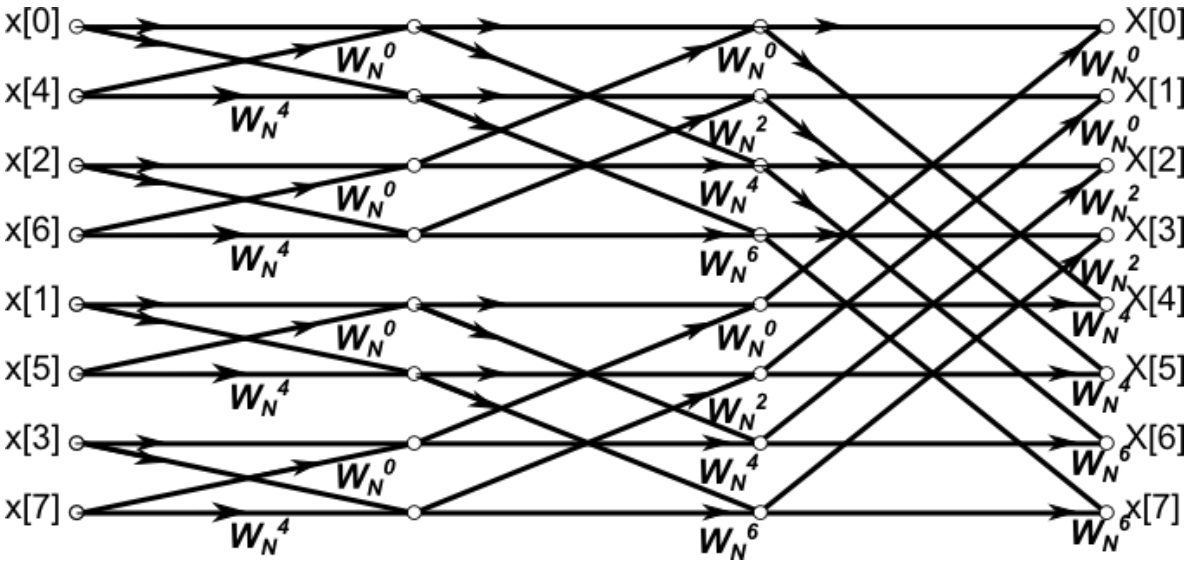


Figura 5 Algoritmo para la FFT de diezmado en tiempo de 8 puntos (Oppenheim, Schafer, & Buck, 1999)

En general cada mariposa implica una multiplicación y dos sumas complejas. Para N puntos, se tiene $N/2$ mariposas por cada etapa del proceso y 2 log N etapas de mariposas en la Figura 6 se muestra el trazo de las operaciones de la "mariposa". Por

lo tanto se guarda el resultado de cada operación de la mariposa (A, B), en las mismas posiciones de sus operandos (A, B). En consecuencia, es necesaria una cantidad fija de memoria, en concreto $2N$ registros de almacenamiento para guardar los resultados de N números complejos.

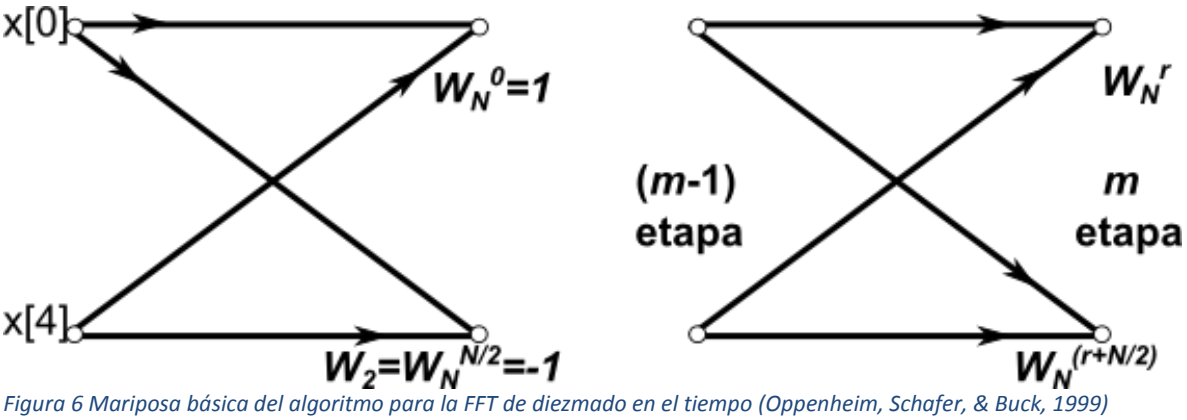


Figura 6 Mariposa básica del algoritmo para la FFT de diezmado en el tiempo (Oppenheim, Schafer, & Buck, 1999)

Con respecto a la tarea de diezmado, esta se puede entender como un reacomodamiento antes de proceder con el algoritmo de la FFT. Por ejemplo si $N = 16$ la tarea de diezmar queda graficada a través de la Figura 7 esto $es\ (v\text{-}1) = \log_2 N$ veces.

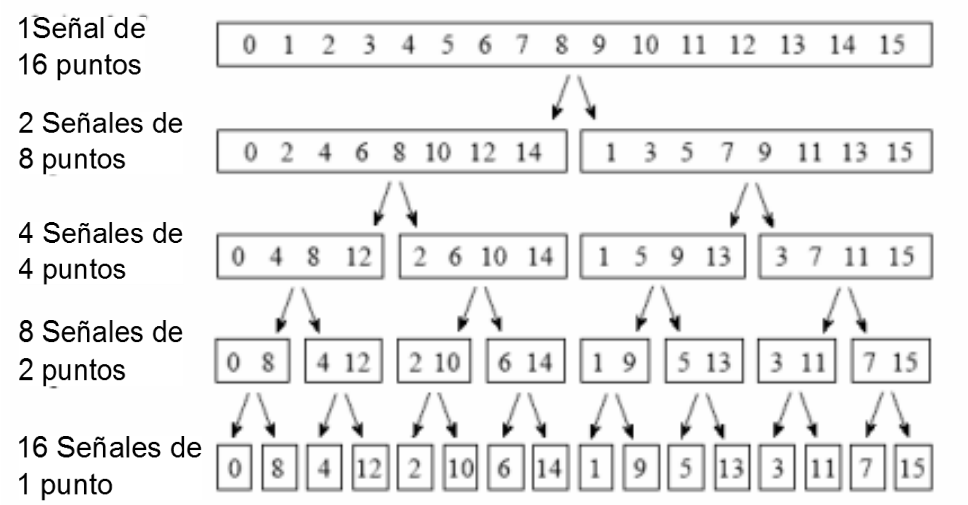


Figura 7 Secuencia de diezmado para N = 16 (Smith, 1999)

En el diezmado la secuencia de datos $x[n]$, se almacena en orden binario invertido esto se muestra en la Tabla *5*.

Tabla 5 Inversión Binaria

Números Ordenados		Cambio	Inversión de números	
Decimal	Binario	➔	Decimal	Binario
0	0000	➔	0	0000
1	0001	➔	1	1000
2	0010	➔	2	0100
3	0011	➔	3	1100
4	0100	➔	4	0010
5	0101	➔	5	1010
6	0110	➔	6	0110
7	0111	➔	7	1110
8	1000	➔	8	0001
9	1001	➔	9	1001
10	1010	➔	10	0101
11	1011	➔	11	1101
12	1100	➔	12	0011
13	1101	➔	13	1011
14	1110	➔	14	0111
15	1111	➔	15	1111

2.2.6. Algoritmo en base 2 diezmado en frecuencia

Otro algoritmo es el algoritmo en base 2 o raddix-2 en inglés. Para deducir el algoritmo se divide la fórmula de la DFT en dos sumatorias, una de las cuales contiene los primeros N/2 puntos de datos y el otro los últimos N/2 puntos de datos. De esta manera se obtiene:

$$X(k) = \sum_{n=0}^{(N/2)-1} x[n]\, W_N^{Kn} + \sum_{n=N/2}^{N-1} x[n] W_N^{Kn}$$

$$= \sum_{n=0}^{(N/2)-1} x[n]\, W_N^{Kn} + W_N^{N\,K/2} \sum_{n=0}^{(N/2)-1} x[n+\frac{N}{2}] W_N^{Kn} \qquad (22)$$

Dado que $W_N^{KN/2} = (-1)^k$, esta expresión puede reescribirse como

$$X(k) = \sum_{n=0}^{(N/2)-1} \left[x[n] + (-1)^k x\left(n + \frac{N}{2}\right)\right] W_N^{Kn} \qquad (23)$$

Se realiza el primer diezmado X (k) (diezmado en frecuencia), obteniendo dos secuencias, par e impar respectivamente de la transformada, esto es:

$$X(2k) = \sum_{n=0}^{(N/2)-1} \left[x(n) + x\left(n + \frac{N}{2}\right)\right] W_{N/2}^{kn} \qquad k = 0,1,\dots,\frac{N}{2} - 1$$

$$X(2k+1) = \sum_{n=0}^{(N/2)-1} \left\{\left[x(n) - x\left(n + \frac{N}{2}\right)\right] W_N^n\right\} W_{N/2}^{kn} \qquad k = 0,1,\dots,\frac{N}{2} - 1 \qquad (24)$$

Se utiliza la propiedad de simetría $W_N^2 = W_{N/2}$. Se definen las secuencias de $N/2$ puntos $g_1\,[n]$ y $g_2\,[n]$ como:

$$g_1\,[n] = x[n] + x[n + \frac{N}{2}]$$

$$g_2\,[n] = \left[x[n] - x\left[n + \frac{N}{2}\right]\right] W_N^n, \quad n = 0,1,\dots,\frac{N}{2} - 1 \qquad (25)$$

Con esto se puede reescribir la ecuación (24) de la forma:

$$X(2k) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} g_1\,[n] W_{\frac{N}{2}}^{kn}$$

$$X(2k+1) = \sum_{n=0}^{(N/2)-1} g_1\,[n] W_{N/2}^{kn} \qquad (26)$$

El cálculo de las secuencias $g_1\,[n]$ y $g_2[n]$ según la expresión (25) y el uso de estas secuencias para el cálculo de las DFTs de $N/2$ puntos se muestra en la Figura 8.

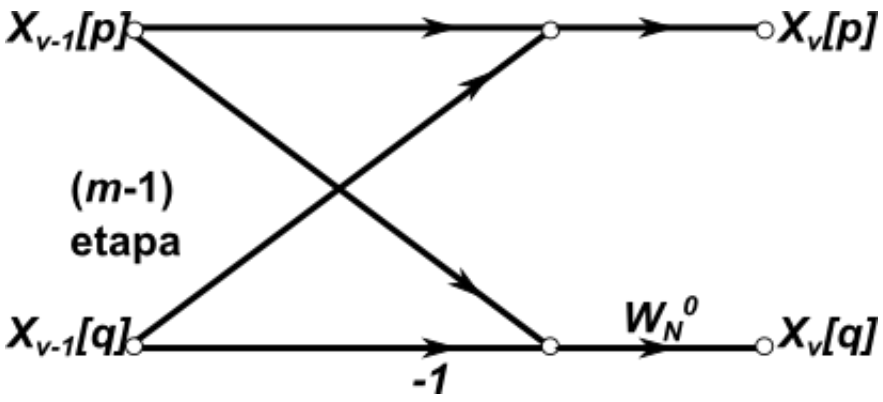


Figura 8 Mariposa básica del algoritmo FFT de diezmado en frecuencia (Oppenheim, Schafer, & Buck, 1999)

Este procedimiento computacional puede repetirse diezmando las DFTs de $N/2$ puntos, $X(2k)$ y $X(2k+1)$. El proceso completo conlleva $v = \log_2 N$ etapas de diezmado, donde cada etapa implica $N/2$ mariposas. Consecuentemente, el cálculo de la DFT de N puntos por medio de la DFT a través del algoritmo FFT de diezmado en frecuencia requiere $(\mathrm{N}/2)\log_2 N$ multiplicaciones complejas y $N\,log_2\,N$ sumas complejas. En la Figura 9 se grafica como es la entrada de la secuencia de datos *x[n]* y en la Figura 10 se muestra el algoritmo de diezmado en frecuencia completo de ocho puntos.

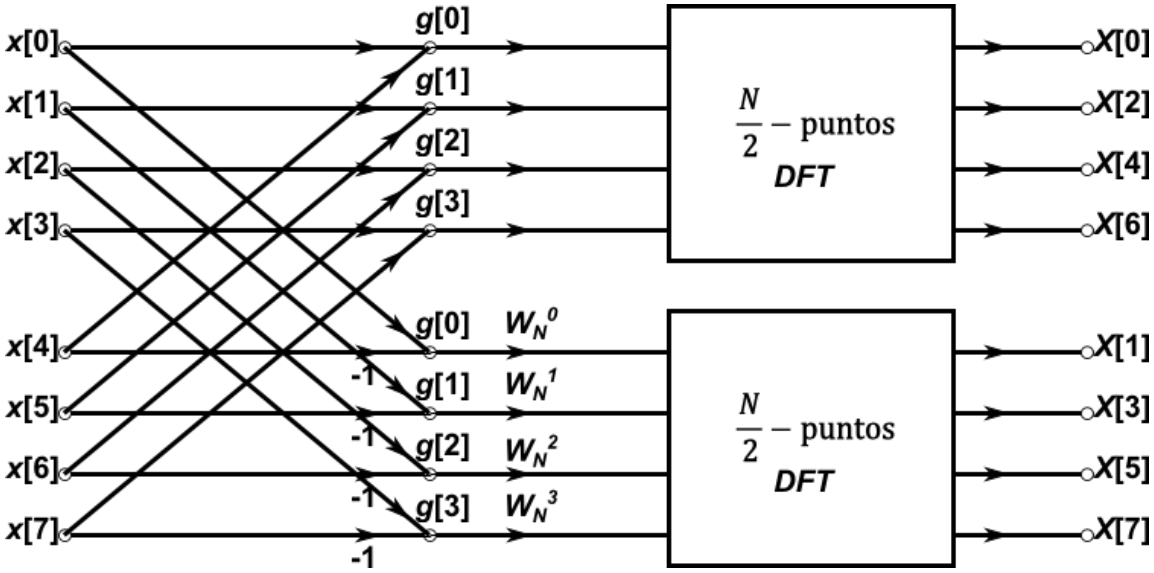


Figura 9 Primera etapa del algoritmo para la FFT de diezmado en frecuencia (Oppenheim, Schafer, & Buck, 1999)

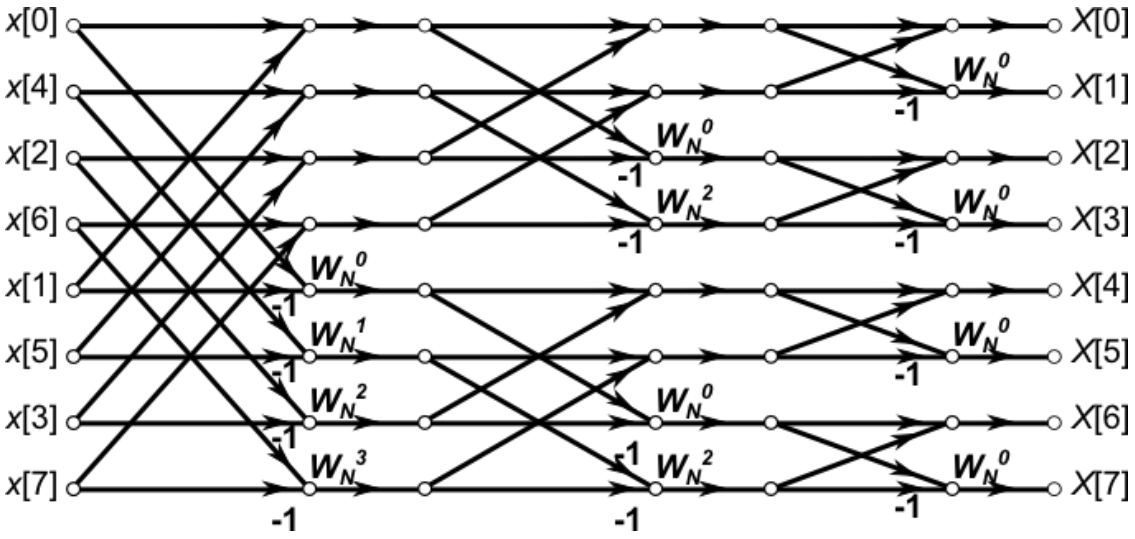


Figura 10 Algoritmo para la FFT de diezmado en frecuencia para N= 8 (Oppenheim, Schafer, & Buck, 1999)

2.2.7. Transformada inversa de Fourier mediante la FFT

Para obtener la transformada inversa de Fourier de forma directa desde la FFT partimos de la ecuación de la IDFT de la forma:

$$DFT \rightarrow X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi k}{N}n} \qquad \text{n=0,1,2,3,..., N-1;} \qquad (27)$$

$$IDFT \rightarrow x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi n}{N}k} \qquad \text{k=0, 1, 2, 3, .., N-1;} \qquad (28)$$

Para simplificar la expresión se utiliza W_n donde

$$e^{-j\frac{2\pi}{N}} = W_N$$

Las ecuaciones 27 y 28 se pueden escribir como:

$$DFT \rightarrow X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk} \qquad (29)$$

$$IDFT \rightarrow \frac{1}{N} x[n] \sum_{k=0}^{N-1} X[k] W_N^{-nk} \qquad (30)$$

Para obtener la IDFT mediante la FFT primero se busca la relación entre la ecuación de la DFT y la IDFT, de las ecuaciones anteriores se observa que la diferencia es el signo

del factor W_N^{nk} además del factor de escala $\frac{1}{N}$ se puede reescribir el signo del W_N^{nk}como:

$$DFT \rightarrow X[k] = \sum_{n=0}^{N-1} x[n] conj(W_N^{-nk}) \quad (31)$$

El siguiente paso es aplicar la propiedad de los complejos conjugados

$$\boldsymbol{conj(a)conj(b) = conj(ab)}$$

Para esto primero se debe tener el conjugado de x[n]

$$DFTconj(x) = \sum_{n=0}^{N-1} conj(x[n])\ conj(W_N^{-nk})$$
$$DFTconj(x) = \sum_{n=0}^{N-1} conj(x[n] W_N^{-nk}) \quad (32)$$

Después se aplica otra propiedad de los complejos conjugados

$$\boldsymbol{conj(a) + conj(b) = conj(a + b)}$$

De forma que podemos simplificar la ecuación (32) de la siguiente forma

$$DFTconj(x) = conj\left(\sum_{n=0}^{N-1} x[n] W_N^{-nk}\right) \quad (33)$$

Como se observa el lado derecho de la ecuación es similar a la IDFT de la ecuación (30) multiplicado por N

$$DFTconj(x) = conj\big(N * IDFT(X)\big) \quad (34)$$

Dado que $conj(N) = N$ y simplificando mediante algebra podemos obtener la siguiente expresión

$$DFTconj(x) = N * conj\big(IDFT(X)\big)$$
$$\frac{1}{N} DFTconj(x) = conj(IDFT(X) \quad (35)$$
$$\frac{1}{N} conj\left(DFT\big(conj(x)\big)\right) = IDFT(X)$$

De esta forma obtenemos el proceso de la transformada inversa mediante el uso de la FFT

$$IFFT(X) = \frac{1}{N} conj\left(FFT\left(conj(X)\right)\right) \quad (36)$$

2.3. STFT

La Transformada de Fourier de Tiempo Corto (Short-time Fourier Transform, STFT) Debido al muestreo de la señal con un periodo finito se presenta una serie de limitaciones y distorsiones en el análisis mediante la FT. Con el fin de eliminar estas anomalías de la transformada de Fourier y eliminar de este modo algunas de sus limitaciones, Gabor en 1946 definió lo que hoy en día se conoce como transformada de Fourier de tiempo reducido. La STFT al igual que las transformaciones estándar de Fourier y otras herramientas son frecuentemente usadas para analizar música. El espectrograma puede por ejemplo, mostrar la frecuencia en el eje horizontal, con las frecuencias más bajas a la izquierda y las más altas a la derecha.

Dadas las series en tiempo *x[n]* la STFT en un tiempo *n* está dada por:

$$X(n, \omega) = \sum_{m=-\infty}^{\infty} x[m] w[n-m] e^{-j\omega m} \quad (37)$$

Donde *w[n]* es la ventana de análisis, que se supone que es distinto de cero solamente en el intervalo $[0, N_w - 1]$.

2.3.1. Transformada corta de Fourier en tiempo discreto

La transformada discreta de Fourier en tiempo corto (STDFT) no es más que la DFT, pero con una ventana de selección. De esta forma, se obtiene un mejor comportamiento a la hora de llevar a cabo un análisis tiempo-frecuencia de las señales. Existen numerosas ventanas, de las cuales se han seleccionado aquellas con un buen índice de desempeño y con implementación en Matlab.

La STFT se obtiene aplicando la transformada de Fourier por un tamaño fijo, moviéndose la ventana a la serie de entrada. La ventana se mueve en un punto de tiempo a la vez, así que se tienen ventanas superpuestas quedando la ecuación 28

$$X(n,k) = X(n,w)_{w=\frac{2\pi}{N}k} = \sum_{m=-\infty}^{\infty} x[m]w[n-m]e^{-j\frac{2\pi}{N}km} \quad (38)$$

Donde N es el factor de frecuencia de muestreo y $2\pi/N$ es la frecuencia intervalo de muestreo.

2.4. VHDL

EL lenguaje VHDL es una herramienta muy útil para describir los sistemas electrónicos digitales. Surgió del programa Circuitos integrados de alta velocidad (Very High Speed Integrated Circuits, VHSIC). Dicho lenguaje permite jerarquizar las diferentes etapas de diseño para lograr una repartición óptima de recursos. Esto mantiene la descripción y diseño de sistemas complejos manejables

Las principales características se pueden mencionar:

• Lenguaje estándar.

• Soporte de las principales compañías proveedoras de herramientas CAD (Diseño Asistido por Computadora).

• Existen formas metódicas para el diseño de máquinas de estados, filtros digitales, bancos de pruebas etc.

• Flexibilidad de implementación en circuitos integrados, el código VHDL es portable entre herramientas, aunque normalmente es necesario hacer ajustes según el dispositivo o la tecnología.

• Es un lenguaje popular cuyo número de usuarios sigue aumentando.

2.4.1. ESTRUCTURA

El lenguaje VHDL es un lenguaje estructurado que cuenta con cuatro bloques principales para el diseño y ejecución que son:

- Bloque de declaración de librerías
- Declaración de terminales externas

- Descripción de la arquitectura del circuito
- Banco de pruebas

En la Figura 12 se puede observar la estructura a seguir según estos bloques. Las librerías que utiliza son las definidas por el estándar IEEE (Institute of Electrical and Electronics Engineers) algunos ejemplos son:

- use ieee.std_logic_1164.all
- use ieee.numeric_std.all
- use ieee.std_logic_arith.all

En el segundo bloque las terminales se declararan lo que en otros entornos de programación llamarían las variables y que designarán las entradas y salidas del circuito.

La arquitectura es la parte fundamental donde se describe el funcionamiento y características del sistema. En la Figura 11 se muestra un ejemplo de un chip de una FPGA.

Y la parte de banco de pruebas no está en la descripción del circuito pero resulta muy útil realizar esta sección pues permite controlar la cantidad de entradas del proyecto.

VHDL

Figura 11 FPGA programable mediante VHDL

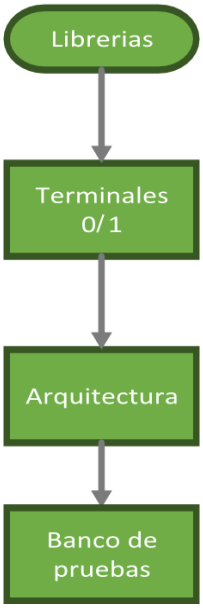


Figura 12 Esquemas de programación en VHDL

2.5. FPGA

Los FPGA surgen como la evolución de los dispositivos electrónicos CPLD (Complex Programmable Logic Device, Dispositivo lógico programable complejo) y PLD (Programmable Logic Device, Dispositivo lógico programable). Combinan lo mejor de los circuitos integrados de aplicación específica (ASICs) y de los sistemas basados en procesadores. La FPGA cuenta con un gran número de celdas elementales o básicas pero con una mayor interconectividad (Romero Troncoso, 2007).

Los FPGA's son una alternativa para la aplicación de la lógica digital en sistemas. Son chips de silicio prefabricados que se pueden programar eléctricamente para implementar cualquier diseño digital. Desde la de primer FPGA que contenía 64 CLB y 58 entradas y salidas, los FPGA's han crecido enormemente en complejidad. Los FPGA modernos ya puede contener aproximadamente 330 000 bloques de lógica y alrededor de 1.100 entradas y salidas.

La arquitectura básica de FPGA consiste en tres componentes principales: bloques lógicos programables que implementan las funciones lógicas, enrutamiento programable (interconexiones) para implementar estas funciones, y las entradas\salidas de bloques para hacer conexiones fuera del chip. Un ejemplo de la arquitectura típica FPGA se muestra en la Figura 13

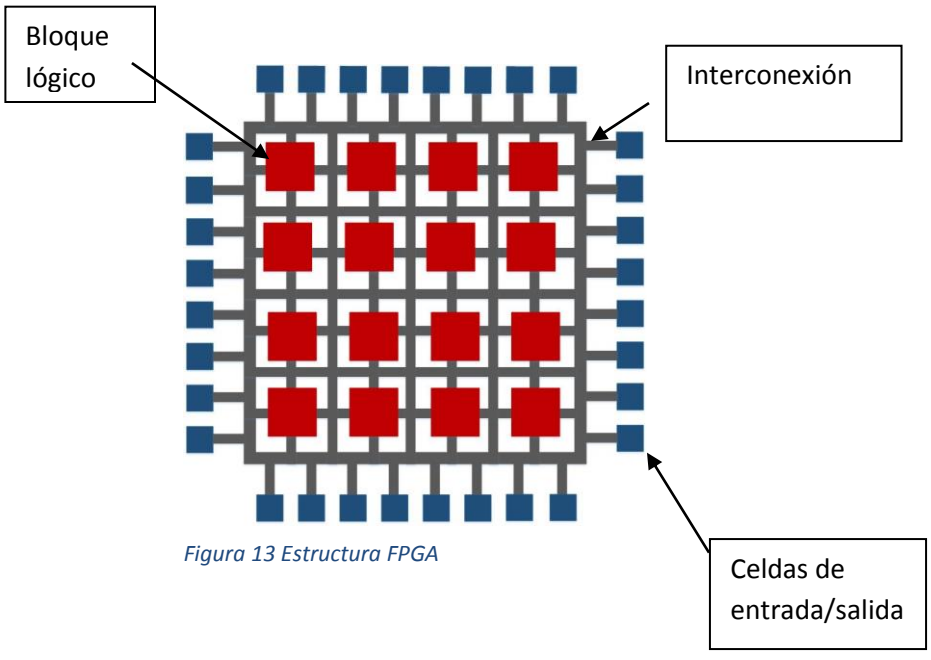


Figura 13 Estructura FPGA

Los FPGA son procesadores que permiten implementar funciones lógicas mediante circuitos lógicos reprogramables. Además de la posible reconfiguración de las líneas de interconexiones. También poseen bloques de entradas y salidas que permiten la comunicación con dispositivos externos (Nasser & Sidharth, 2010).

Beneficios de la FPGA:

- Tiempos más rápidos de respuesta de E/S y funcionalidad especializada
- Exceder la potencia de cómputo de procesadores de señales digitales
- Rápida generación de prototipos y verificación sin el proceso de fabricación del diseño personalizado de ASIC.
- Implementar funcionalidad personalizada con la fiabilidad de hardware determinístico dedicado.
- Se puede actualizar en campo, eliminando los gastos por rediseño personalizado de ASIC y mantenimiento.

2.6. IP CORES

Un núcleo de propiedad intelectual (IP core) es un bloque lógico o de datos que se utilizado en la fabricación de una FPGA o en circuitos integrado de aplicación específica (ASIC) para un producto. Una de las características esenciales de estos bloques es la reutilización de diseños, los IP core son parte de la creciente automatización de diseño electrónico (EDA) tendencia de la industria hacia el uso repetido de los componentes diseñados previamente. Idealmente, un núcleo IP debería ser completamente portátil - es decir, capaz de ser insertado fácilmente en cualquier tecnología de proveedor o metodología de diseño. Algunos ejemplos de IP cores son: Receptor / Transmisor Universal Asíncrono (UART's), unidades centrales de procesamiento (CPU's), controladores Ethernet y las interfaces PCI. El uso de IP cores en un diseño facilita la

generación de este ahorrando tiempo al programador en la Figura 14 se muestra una representación simple del diseño mediante IP cores

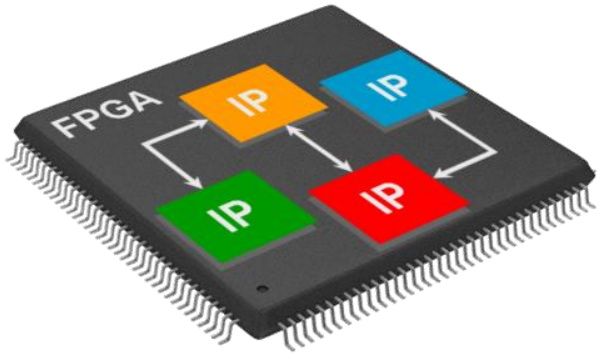


Figura 14 Ejemplo de diseño mediante IP cores

Los IP core son bloques que se pueden reutilizar eficientemente mediante la optimización, verificación y documentación de estos. Los IP cores existen en diferentes formatos. El hard IP core es la representación física del diseño creado para aplicaciones específicas por lo que no son tan flexibles como los soft IP cores o los firm IP cores, aunque tienen un mejor desempeño que estos. Los firm IP cores proporciona una descripción de nivel de transferencia de registros de un circuito, que especifican las operaciones que realizaran los operandos almacenados en los registros. Son muy cercanos a los hard IP cores pero estos pueden ser configurados para diferentes aplicaciones. Los soft IP cores proporcionan la descripción del comportamiento del circuito que puede ser sintetizado para diferentes aplicaciones y tecnologías por su configuración flexible en la Tabla *6* se muestra la clasificación de IP cores. El diseño creado mediante la utilización de los IP cores es llamado diseño SoC (system on a chip), con la ventaja de realizar el diseño más rápido mediante el reúso de diseños anteriores (Vai, 2001).

Tabla 6 Tipos de IP Core

Categoría	Modificaciones	Costo	Descripción
Soft IP core	**Muchas**	**Alto**	**HDL**
Firm IP core	**Algunas**	**Medio**	**Conexiones**
Hard IP core	**No**	**Bajo**	**Diseño**

Para el diseño de SoC hay disponible una gran variedad de tipos de IP cores con diferentes funciones. Los diseñadores tienen la ventaja de poder escoger de diferentes opciones los cores asegurándose de que sean verificados y bien hechos para poder integrarlos a su diseño mediante el método de conectar y utilizar (Gizopoulos et al. 2004).

2.7. Doxygen

El Doxygen es una herramienta estándar que ayuda en la generación de documentos con referencias C++ empleados en lenguajes de programación, sin embargo también es compatible con otros lenguajes populares como C, objetive-C, C#, PHP, Java entre otras.

Doxygen puede ayudar de tres formas distintas:

1. Puede generar un navegador de documentación en línea (HTML) así como un manual de referencias *offline* (LaTeX escrito en texto plano) a partir de un conjunto de archivos fuente. También puede dar soporte para la generación de salidas en formatos RTF, PostScrib, PDF etc. La documentación es extraída directamente de las fuentes, lo que facilita la documentación de conformidad con el código fuente.
2. Puede ser configurado para extraer la estructura del código de los archivos de origen indocumentado. Esto es muy útil para rápidamente el camino en códigos fuente de gran distribución. Doxygen puede visualizar las relaciones entre los distintos elementos por medio de incluir gráficos de dependencia y otros que son generados automáticamente.
3. De la misma forma Doxygen puede ser empleado para la creación de documentación normal. (Heesch, 2014)

CAPÍTULO 3

3. Metodología.

En este capítulo se presenta la metodología seguida para el diseño de los IP core así como la documentación de los módulos utilizados en estos núcleos.

El diseño de un IP core requiere del desarrollo de módulos propios que operen sin requerir unidades externas para su funcionamiento, de forma que trabajen sin la necesidad de modificar el código interno.

En este aspecto la documentación es muy importante, pues permite al usuario implementar el núcleo de forma adecuada y en caso de que se necesite alguna modificación para la mejora de éste, logre de forma sencilla conocer con exactitud la función de cada módulo así como la interconexión de estos. La Figura 15 muestra la metodología a seguir para la generación y documentación de los IP cores.

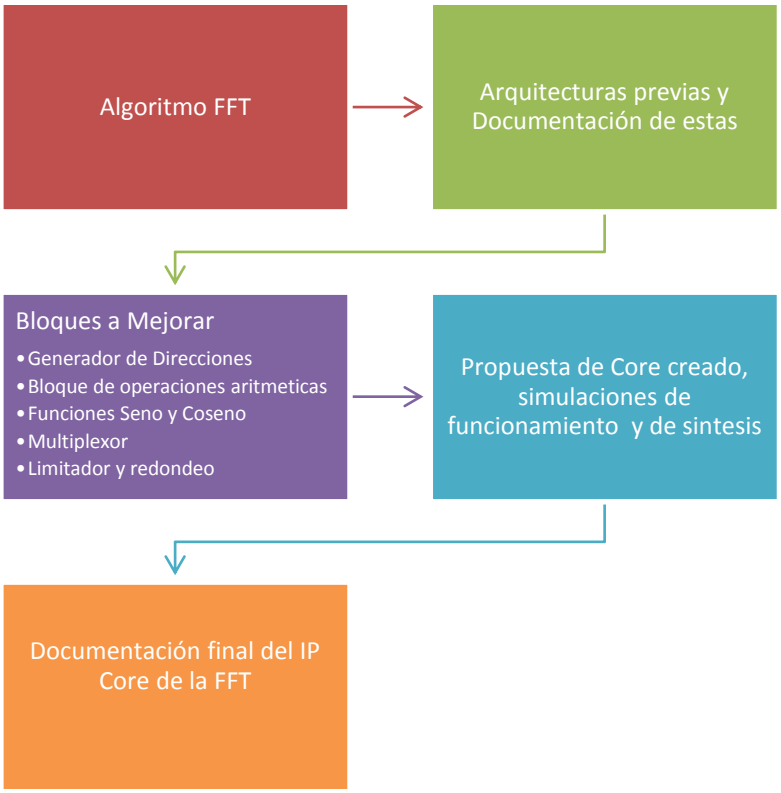


Figura 15 Diagrama de Metodología

3.1. Algoritmo de la FFT

La primer actividad fue la de familiarizarse con el proceso de la FFT para implementarlo en un código VHDL , como se mencionó en la sección 2.2.5 este algoritmo permite obtener la DFT de forma rápida en un tiempo menor que el uso directo de la DFT puesto que reduce el número de operaciones complejas a realizar además de simplificar las operaciones a sumas y restas complejas , pero para obtener un espectro de frecuencia, la frecuencia máxima de la señal muestreada deber ser menor a la mitad de la frecuencia de muestreo según el teorema de muestreo de Nyquist (Lavry Engineering, Inc., 2004), esto se muestra en la ecuación 39

$$Fs < 2F_{max} \qquad (39)$$

Siendo *Fs* la frecuencia de muestreo y F_{max} la frecuencia de la señal.

La operación de la FFT se reduce a una serie de sumas y resta multiplicadas por el factor de fase *W* cambiando la dirección de las entradas para tomar los diferentes puntos de la señal esto se representa en la Figura 16 Operaciones de una transformada DFT de dos puntos o "mariposa"

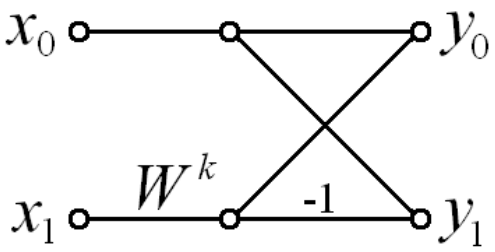


Figura 16 Operaciones de una transformada DFT de dos puntos o "mariposa"

Los índices de x_0 y x_1 van cambiando con respecto al número de operación a realizar , lo mismo para el factor *W* que cambia respecto a funciones sinusoidales de frecuencia $2\pi n/N$ para *N* puntos entregando *N* componentes espectrales equiespaciadas cada $2\pi/N$, donde 2π representa la frecuencia de muestreo *Fs.* Para simular esto se puede generar ondas sinusoidales de 0 a π.

3.2. Herramienta de textos Doxygen

Para realizar la documentación correcta de los archivos en VHDL se optó por el uso del software libre Doxywizard, para esto es necesario hacer los comentarios correctos sobre las funciones, librerías y datos adicionales que se utilizará en la documentación final. Este software analiza y compila los archivos que constituyen al proyecto y genera un archivo documentando cada sección del archivo que se le indique , en la Figura 17 se muestra la interfaz del programa la cual consta de tres opciones principales : "Wizard" que genera los documentos de forma automática utilizando una configuración predeterminada , la opción de "Expert" que permite modificar los parámetros y opciones que se requieren generar en el proyecto y la última opción que es "Run" esta opción es el botón de inicio para la compilación y generación de documentos del proyecto para el caso de las dos opciones anteriores.

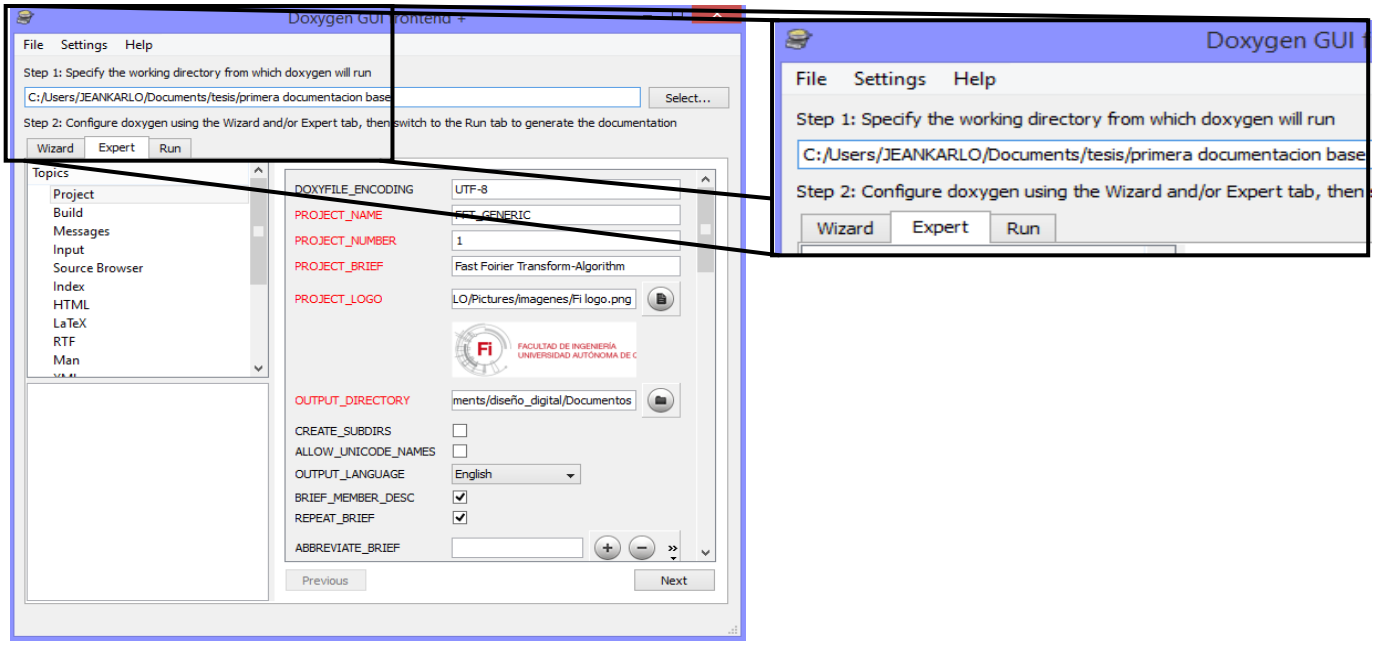


Figura 17 Ventana de trabajo Doxywizard

Para tener un control completo sobre los archivos generados se utiliza la opción experto, esto permite controlar la edición del documento a crear así como los archivos que genera en este caso se crearan dos tipos de documentos, uno en formato HTML y el otro en LaTeX el cual sirve como base para generar un archivo PDF más completo, la ventaja de este archivo es que es posible editarlo más adelante con algún compilador

como TexWorks por si requiere modificarlos para estilizarlos o si se le requiere editar alguna opción que doxywizard no proporcione. En la Figura 18 se muestra un ejemplo de la interfaz de TexWorks.

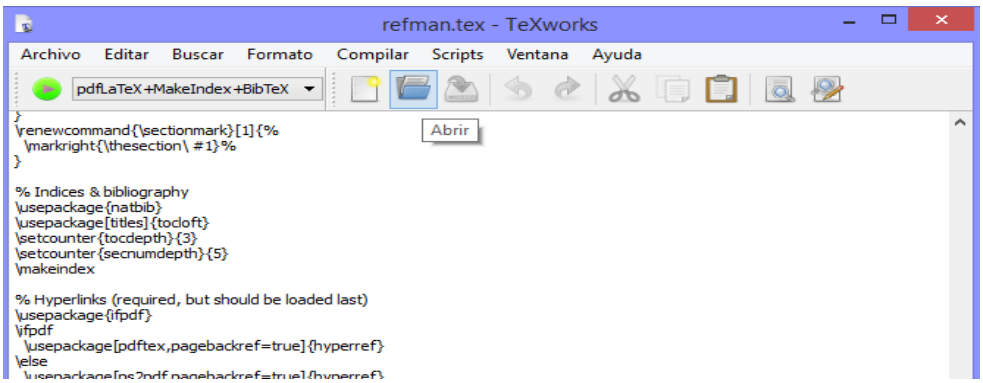


Figura 18 Ventana de Tex Works para compilación de texto LaTeX

El resultado proporcionado por doxywizard es la generación de documentos PDF mediante LaTeX y documentos en HTML que brindan las opciones de navegación sencilla para el usuario con la opción de mostrar los bloques de forma jerárquica así como las señales utilizadas y el archivo fuente .

3.3. Revisión de implementación en arquitectura Digital

Para comenzar se parte de la revisión de la arquitectura de hardware desarrolladas.

El problema con estas arquitecturas es que están diseñadas para un proceso en específico además de que está compuesta por diferentes archivos lo que dificulta el seguimiento del proceso además de tener nombres que no corresponden con el estándar para la documentación correcta, además de contener archivos que no tienen una función dentro del proceso.

Una vez obtenida la información necesaria se prosiguió con la parte de simulación y bosquejo del diagrama de los bloques utilizados para iniciar el rediseño para la generación de los IP cores (núcleos) así como la función de cada uno de los bloques. Se llegó a la conclusión de utilizar una de estas arquitectura pero desarrollar mejoras en los bloques de operación de la “mariposa” y en los bloques de las señales de Seno y

Coseno pues para generar estas memorias ROM (Read only memory) se utiliza el software MATLAB mediante el uso de dos archivos que permites generar la memoria ROM esto representa un problema ya que la ventaja de los IP cores que no necesitan señales externas por eso se formulará una operación para crear la ROM dentro del mismo CORE y así evitar el uso de programas externos.

3.4. Módulo IP core propuesto

3.4.1. Operación “mariposa”

Para la operación de la mariposa se usan tres bloques de operaciones aritméticas que son la suma, la resta y la multiplicación. En la Figura 19 se presenta el diagrama jerárquico del proceso de la mariposa y en la Figura 20 se muestra la conexión de los bloques ya mencionados de las operaciones aritméticas.

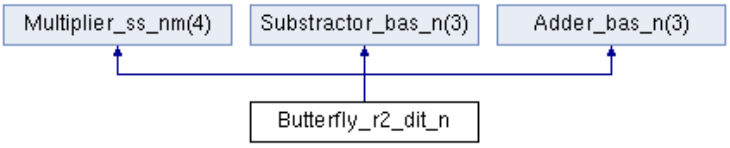


Figura 19 Diagrama jerárquico de la mariposa

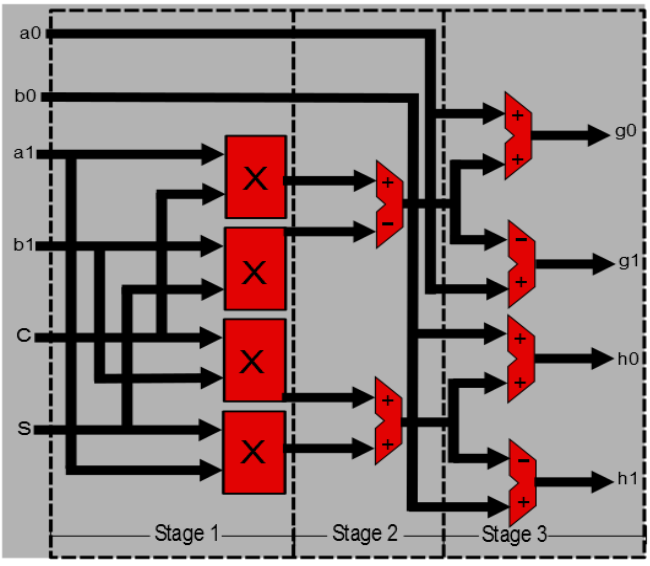


Figura 20 Diagrama a bloque de la mariposa

Para disminuir el número de archivos y el tamaño utilizado por estas operaciones se propone realizar las operaciones de forma directa contemplando el cambio de formato que puedan necesitar para así reducir el número de archivos a utilizar.

En esta parte también se mejora el ajuste de datos correspondiente al resultado de la suma y resta de las señales SA y SB, para esto se planea usar un algoritmo que redondee los datos de salida y que al mismo tiempo límite los valores de los datos para evitar el sobre flujo, este método es el algoritmo del banquero o Banker´s method. Esto mejora el ajuste anterior el cual solo tomaba los bits más significativos de las operaciones quitando los bits menos significativos el problema es que se existe la posibilidad de que la información omitida pueda modificar el resultado de las operaciones posteriores.

3.4.2. ROM de Seno y Coseno

Se busca implementar un método que permita generar la memoria ROM de las señales seno y coseno pues en ejemplos anteriores se utilizaba el software Matlab para la generación de estas memoria creando conflictos con la idea de un core el cual debe de ser capaz de trabajar sin necesidad de señales o sistemas externos que lo modifiquen. Para esto se contempló la idea de utilizar las funciones *Sin* y *Cos* que se encuentran dentro de la librería IEEE.Math_real.all la cual es una librería estándar, para esto se debe considerar el formato que utiliza la función, que en este caso es de tipo real, tipo de dato no es muy utilizado en las aplicaciones de la licenciatura por lo que se buscara información que ayude a conocer este tipo de funciones.

3.4.3. Funciones

El lenguaje en VHDL tiene muchas herramientas que permiten realizar diseños más completos, entre estas herramientas se encuentran las funciones (*function*) esta herramienta es muy similar a las funciones utilizadas en otros ambientes de programación como C o C++ por citar algunos ejemplos, se plantea generar un función que almacene el proceso de llenado de la memoria ROM para el caso de las señales de seno y coseno, de manera que el ancho de la ventana o de datos sea decidido por el usuario, beneficiando así el diseño de aplicaciones diferentes sin necesidad de reescribir el código. Esto también evitara la dependencia de sistemas externos para la formulación de estas memorias.

3.4.4. Memoria RAM

En las arquitecturas pasadas se utilizaba el software Matlab para la generación de una Memoria con los datos a utilizar lo que requería cierto conocimiento del programa de Matlab para editar las señales, estos no es viable pues se pretende utilizar la FFT en tiempo real por lo que se sustituirá este bloque por una memoria RAM para almacenar los datos que se utilizarán en la transformada además de contar con dos bloques de almacenamiento en la salida de la transformada para procesos que lo requieran.

3.4.5. Pruebas Simuladas

Se propone emular el funcionamiento de las transformadas con datos previamente adquiridos y mostrar el resultado además, sintetizar el trabajo con dos programas diferentes para comparar el espacio utilizado por la tarjeta FPGA con el IP core de la FFT y la simulación de los bloques en los que se trabajó de forma independiente.

CAPÍTULO 4

4. Pruebas y resultados

En este capítulo se presentan los resultados de las etapas propuestas en este trabajo.

En primer lugar se analizaron los diferentes programas en los cuales se había utilizado el proceso de la FFT, y la forma en que se aplicaba el algoritmo de la mariposa, en algunos de estos proyectos faltaban archivos y señales de inicio o comentarios que ayudarán a identificar diseño completo. Se decidió por un proyecto en el cual se utilizaba el diezmado en tiempo el cual se encontraba más completo que los anteriores, en base a este se prosiguió a identificar los archivos que podrían mejorarse y en dado caso reducir, un ejemplo de esto fue en las operaciones aritméticas dentro del algoritmo de la mariposa (multiplicación, resta y suma).

4.1. Casos de estudio

4.1.1. Algoritmo de la mariposa

Para el proceso del algoritmo de la mariposa anteriormente se utilizaban cuatro archivos, uno para cada operación aritmética, un archivo de suma, resta y multiplicación, más las memorias ROM de las señales seno y coseno que se usan como factor de fase, por lo que el proyecto completo contemplaba un total de trece archivos para el uso de la FFT. Para minimizar este número de archivos se realizaron las operaciones aritméticas dentro del archivo base del algoritmo de la mariposa reduciendo así el número de archivos a utilizar.

4.1.1.1. Multiplexor

El archivo del multiplexor fue eliminado del nuevo Core pues su función quedo obsoleta ya que las señales están conectadas directamente y se usan los contadores para acceder a las memorias RAM sin necesidad de cambiar la dirección de lectura y escritura por medio de este multiplexor.

4.1.2. Archivo Limitador

Otra modificación fue la de agregar una etapa más al proceso de la mariposa mediante un limitador el cual evita el sobre flujo de datos al final de las operaciones evitando así errores por desbordamiento, además de que realiza un redondeo para mantener los datos lo más exactos posible al momento de realizar los diferentes cambios de formato, ya que anteriormente se realizaba un truncamiento o corte manteniendo solo los bits más significativos para las operaciones lo que disminuía la exactitud y precisión de la FFT.

4.1.3. Funciones Seno y Coseno

Para eliminar la dependencia de otro programa para la generación de las señales de Seno y Coseno se planteó el uso de la librería “IEEE.Math_real.all” dentro del IP core, el uso de esta librería permite emplear las funciones especiales de “Sin “y “Cos” programadas dentro de esta librería estándar del lenguaje, el resultado de las funciones se guardan en una ROM que se crea en el momento de activación del Core en la Figura 21 se muestran las funciones generadas. Otra librería utilizada fue la “IEEE.numeric_std.all” la cual permite ajustar los formatos para las operaciones necesarias ya sea entero, o vector con signo o sin signo, según lo requiera, esto permitió disminuir aún más el número de archivos que se utiliza al final del proyecto. Cabe mencionar que los cambios hechos hasta esta sección han sido generados dentro de un solo archivo que es el que realiza las operaciones completas del algoritmo a excepción del limitador que es un archivo aparte que se interconecta a las salidas (*g0, h1, g1, h1*).

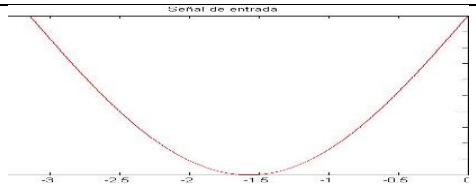
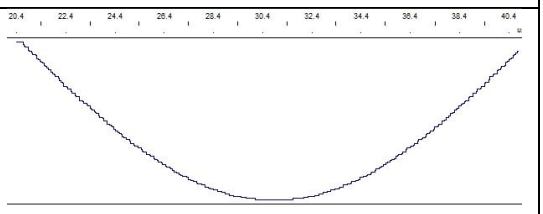
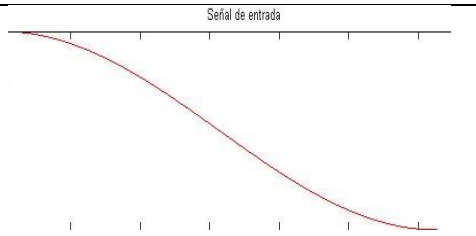
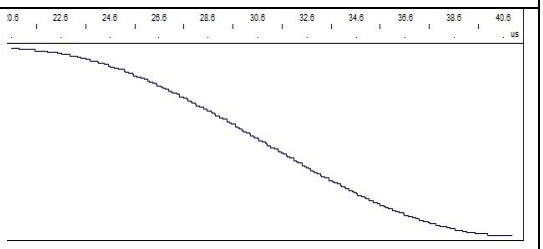
	Matlab	VHDL
Sin		
Cos		

Figura 21 Graficas del factor de fase

4.1.4. Mejora mediante el uso de diezmado en frecuencia

Otro aspecto que se planteó y se desarrolló fue el tipo de diezmado que utiliza el Core, pues de utilizar el algoritmo de diezmado en tiempo se optó por utilizar el diezmado en frecuencia, esto implicó el cambio en el orden de las operaciones aritméticas dentro del core para la correcta implementación de este proceso. En la Figura 22 se muestra una comparación de los bloques de cada tipo de operación y el aumento de etapas en la segunda imagen debido al limitador.

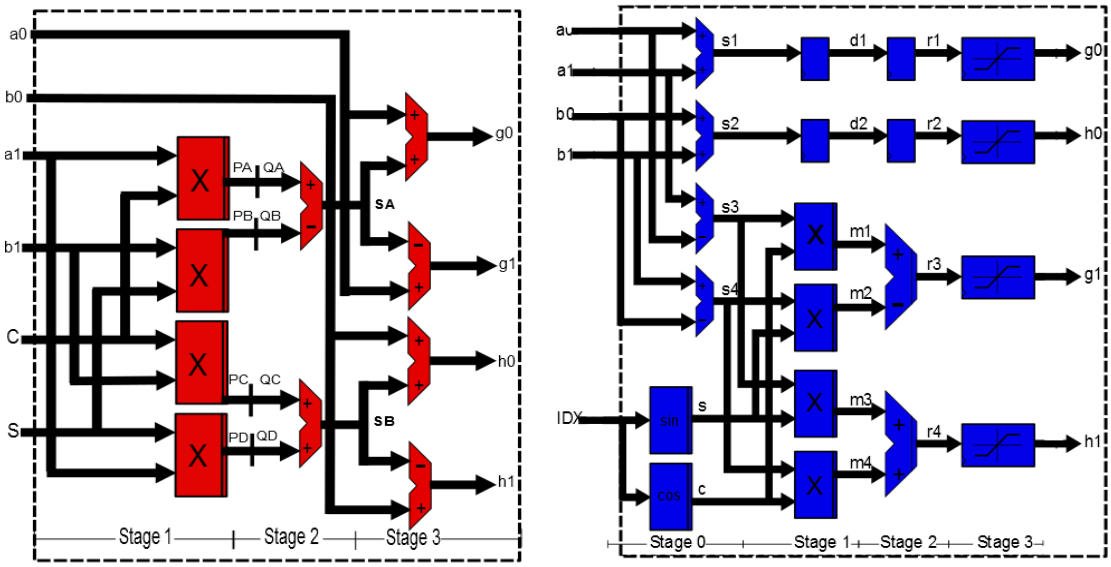


Figura 22 Bloques de operación Diezmado en tiempo (izquierda) y Diezmado en frecuencia (derecha)

4.1.5. IFFT

Como última adición al Core se le agregó una etapa final que permite recobrar la señal de entrada mediante los datos obtenidos por la FFT es decir aplicando la IFFT de un modo sencillo utilizando la misma herramienta de la FFT esto se realiza conjugando la señal obtenida de la FFT y realizar nuevamente el computo de la FFT, de esta manera se podrá comprobar la efectividad del Core, además de contar con ambos resultados en caso de necesitarlos, esto queda a consideración del diseñador.

4.2. Resultados

4.2.1. Pruebas simuladas del IP core

En esta sección se muestran las pruebas que se realizaron con el IP Core con dos señales de prueba y se comparan con la FFT de un programa matemático especializado. Para la primera prueba se simuló la señal de la fórmula 30 que es una señal sinusoidal simple de una frecuencia, en la Figura 23 se muestra la señal de entrada y el resultado obtenido mediante el uso del Core y al compararlo con la función FFT del programa MATLAB que se muestra en la Figura 24 se puede observar que el Core cumple con la tarea de mostrar el número de frecuencias encontrada dentro de esta señal

$$2^8 \sin(2\pi * 4) \quad (40)$$

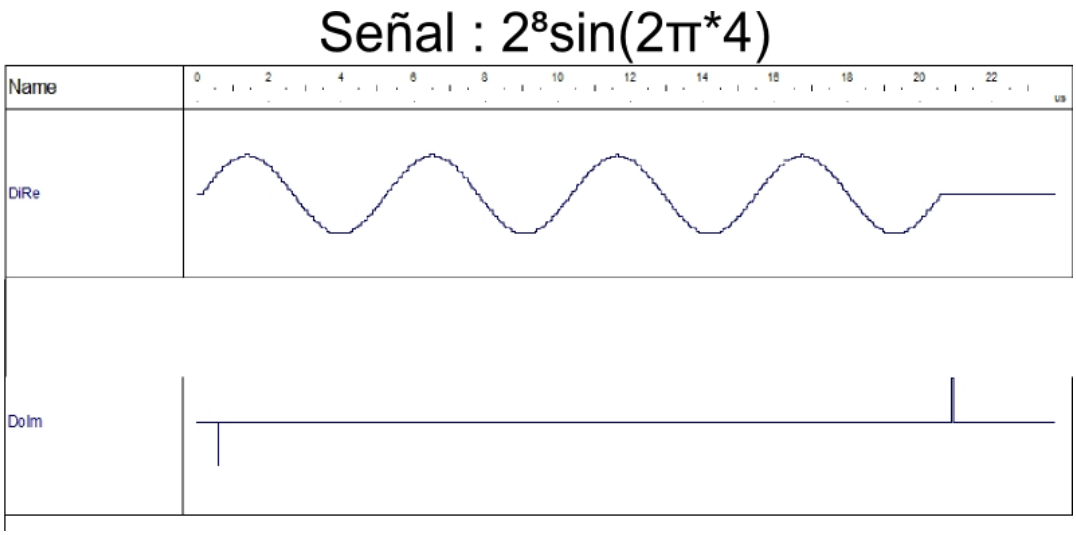


Figura 23 Simulación IP core

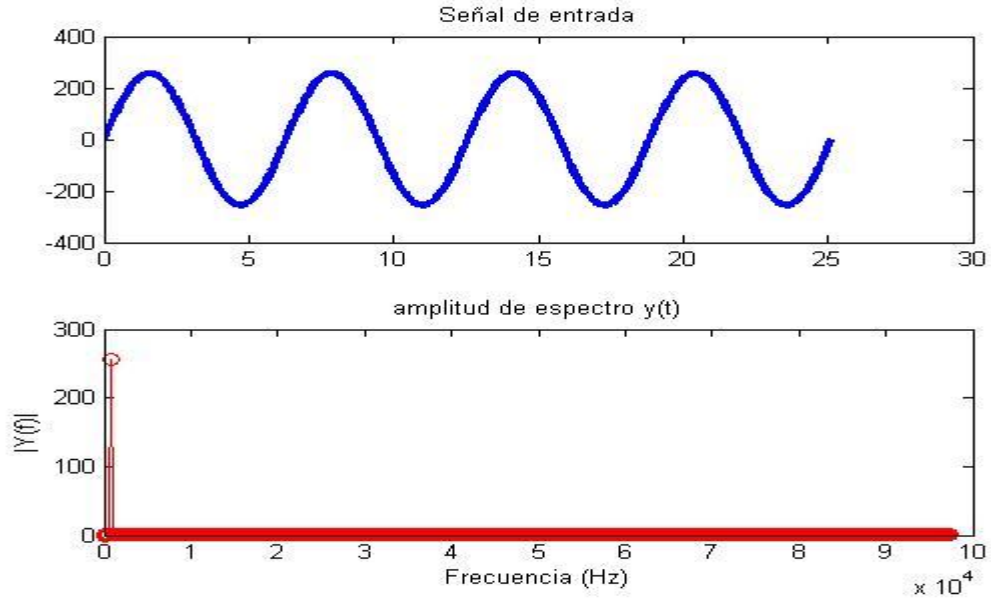


Figura 24 FFT mediante MATLAB

Para comprobar la sección de la IFFT se utilizó la misma señal obtenida de la FFT y en base a esta se recuperó la señal original. En la Figura 25 se muestra el resultado obtenido, así mismo se muestra en la Figura 26 la IFFT utilizando el mismo método que en el Core pero implementado en MATLAB.

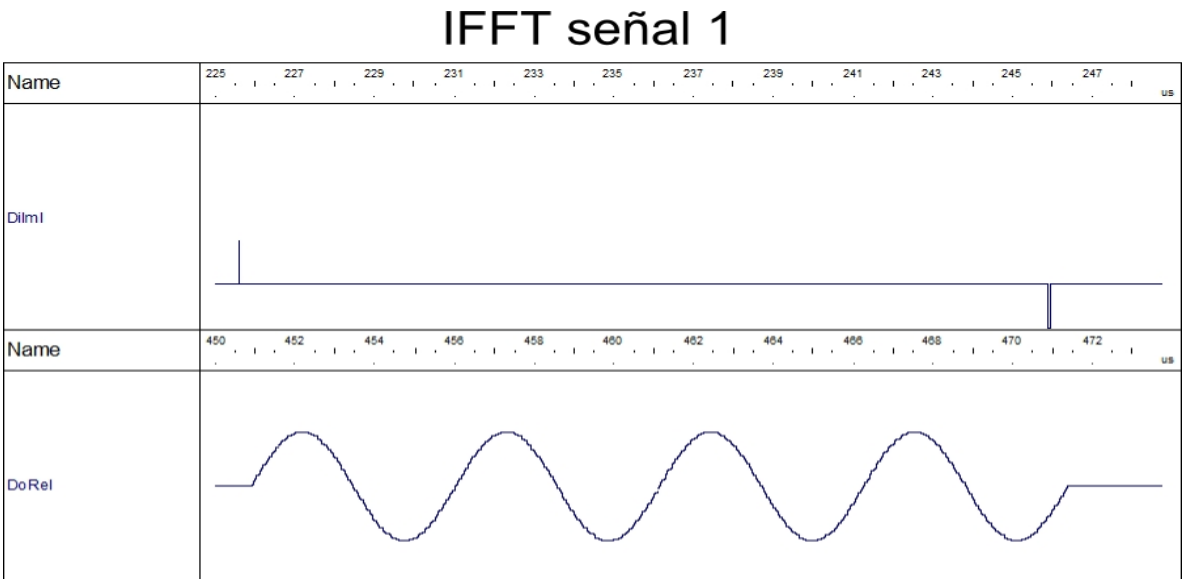


Figura 25 IFFT señal 1

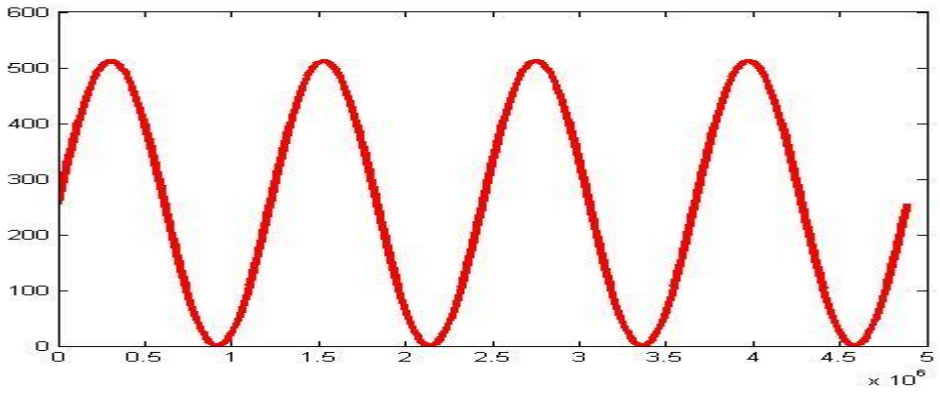


Figura 26 IFFT mediante MATLAB

Como se puede observar la señal reconstruida mediante la aplicación de la IFFT es igual a la señal de entrada, pasa lo mismo con la IFFT obtenida mediante MATLAB. Para comprobar que los resultados son correctos se realizó una segunda simulación con una ecuación diferente para probar la detección de 2 frecuencias en la Figura 27 se muestran los resultados obtenidos utilizando el Core y el programa MATLAB para comprobar el resultado obtenido en la Figura 28.

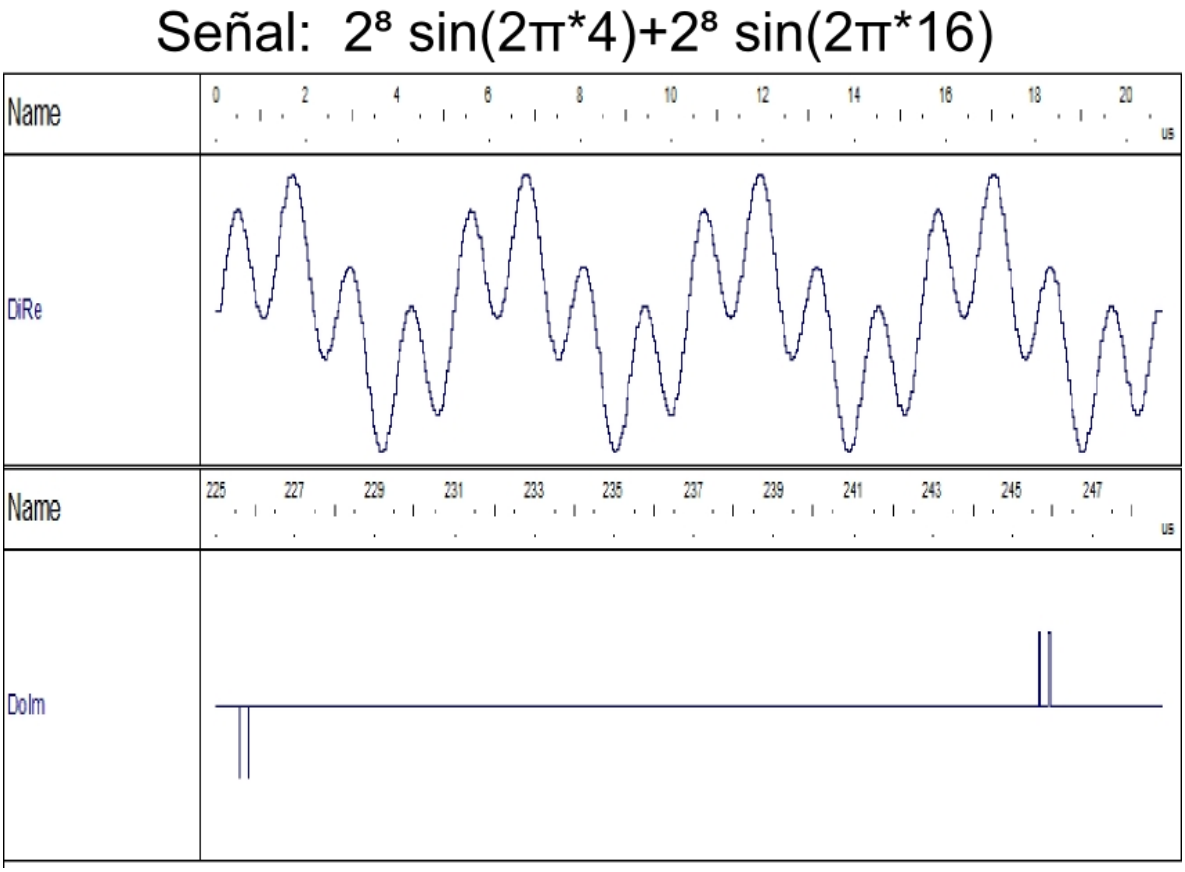


Figura 27 Señal 2 Core

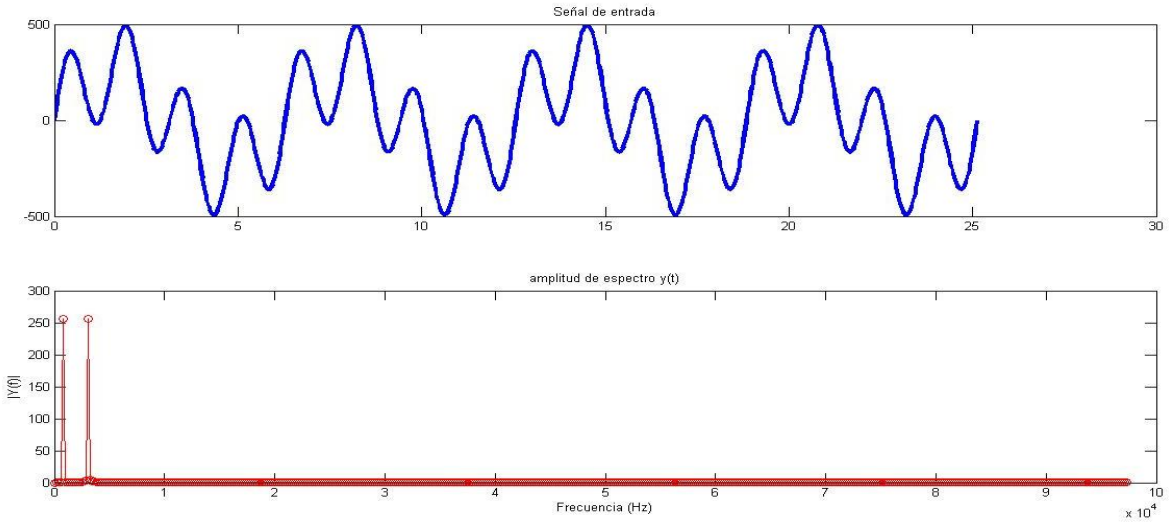


Figura 28 Señal 2 MATLAB

Se observa que el Core detecta las dos frecuencias dentro de la señal al igual que MATLAB por lo que el Core funciona de manera adecuada incluyendo la función de la IFFT. Para realizar la STFT solo se requiere utilizar el core en ciertos lapsos de tiempo,

esto para emular la función ventana que requiere la STFT para trabajar. Lo que dependerá del diseñador pues en base a las características que requiera ya sea precisión para identificar en tiempo cambio la frecuencia o conocer la frecuencia de forma más precisa, son cuestiones a decidir por el usuario.

4.2.2. STFT

Para la sección de la STFT se utilizó el Core de la FFT en diferentes secciones de tiempo para generar el ventaneo de la señal, para las pruebas se utilizaron la señal sinusoidal simple de la sección anterior la cual se muestra en la Figura 29, se observa a la FFT trabajando en las ventanas de tiempo.

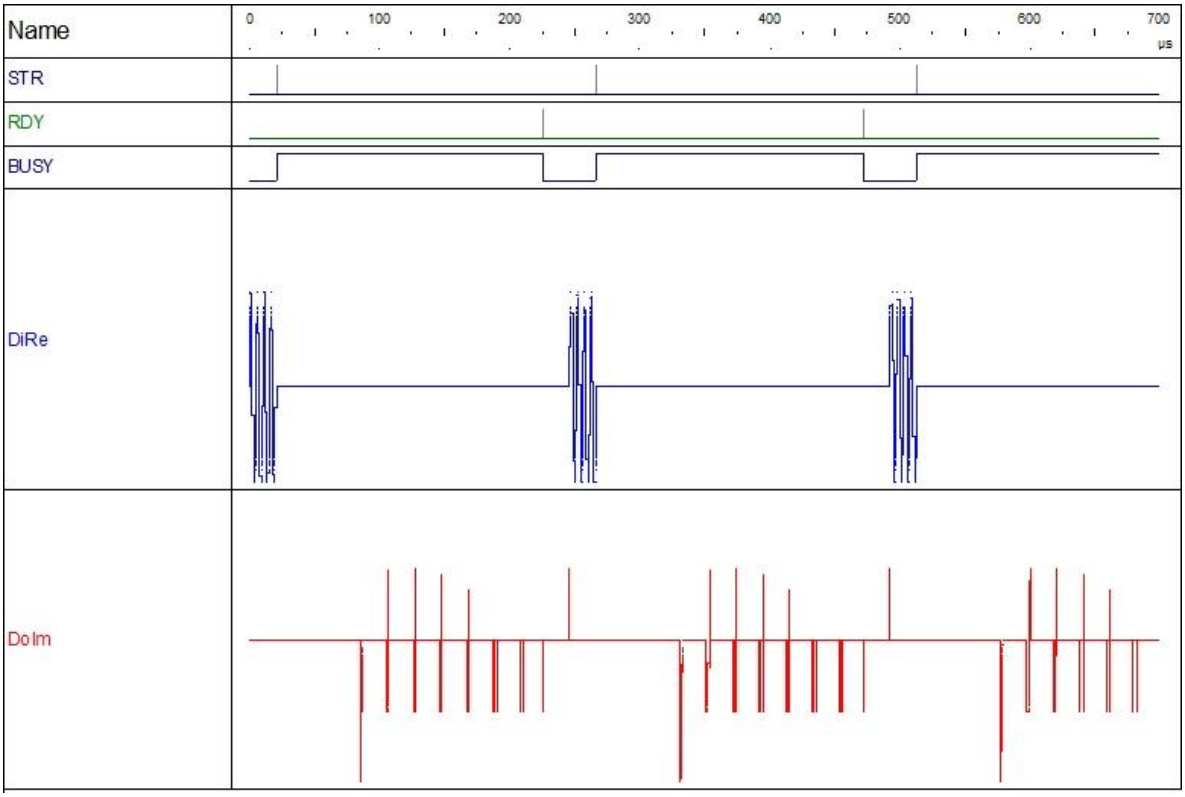


Figura 29 Ventaneo de la señal sinusoidal simple

El muestreo se realizó dando un tiempo de 100 nanosegundos para volver a realizar la FFT, para observar mejor la detección de frecuencias se unieron las dos señales de prueba de la sección anterior y se le aplico la FFT en la Figura 30 se muestra el resultado obtenido.

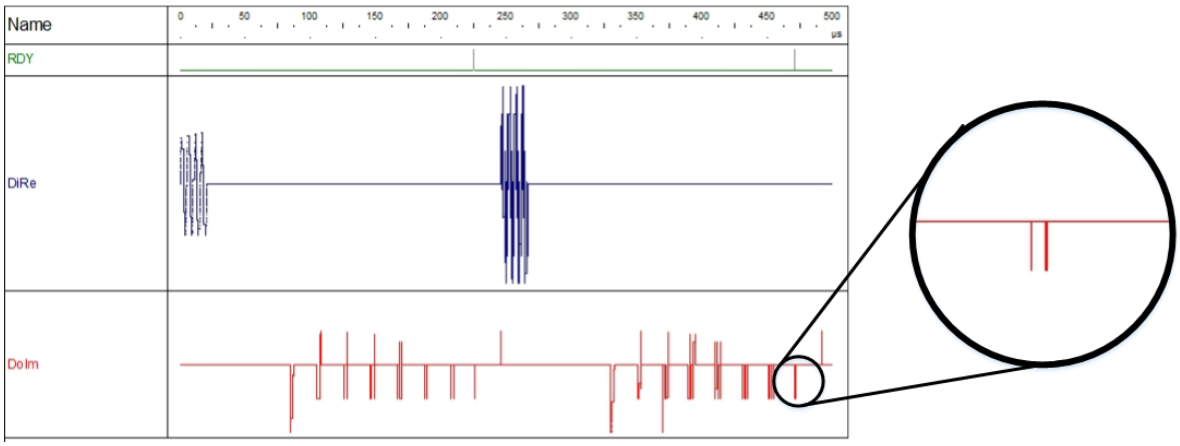


Figura 30 Ventaneo de señal compuesta

Al comparar el resultado con los obtenidos en la sección anterior se concluye que el resultado es el mismo por lo que la STFT está trabajando de forma correcta.

4.2.3. Programación en FPGA

Para observar la diferencia entre el proyecto base y el core propuesto en este trabajo se utilizó la tarjeta Spartan3E-1600 de Xilinx para programar el Core e identificar si se realizaron mejoras al momento de sintetizar y programar la FPGA. En la Tabla 7 se muestra el resultado del proyecto base y los datos obtenidos de la síntesis del proyecto.

Tabla 7 Proyecto base con 1024 puntos 12 bits, 38MHz, tarda 15370 ciclos.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	1359	14752	9%
Number of Slice Flip Flops	71	29504	0%
Number of 4 input LUTs	5625	29504	19%
Number of bonded IOBs	52	250	20%
Number of MULT18X18SIOs	4	36	11%
Number of GCLKs	1	24	4%

Los resultados de la síntesis de este proyecto utilizando 1024 con 12 bits fue que para realizar la FFT se necesitaron 15,370 ciclos, a comparación en la síntesis del Core se obtuvo una diferencia razonable en cuanto al número de elementos lógicos utilizados, esto se puedo observar en la Tabla 8 además de realizar la FFT en un numero de ciclos menor que en la tabla anterior.

Tabla 8 IP Core con 1024 puntos, 12 bits, 83MHz, tarda 10240 ciclos.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	587	14752	3%
Number of Slice Flip Flops	254	29504	0%
Number of 4 input LUTs	1109	29504	3%
Number of bonded IOBs	57	250	22%
Number of BRAMs	2	36	5%
Number of MULT18X18SIOs	4	36	11%
Number of GCLKs	1	24	4%

En cuanto al tiempo del proceso de la FFT se observa una clara reducción de tiempo esto al compararla con la Figura 31 y la Figura 32 que son las simulaciones realizadas con la misma señal sinusoidal primero por el proyecto base y la segunda por el IP core, en este ejemplo pasamos de 360 µs a 245 µs.

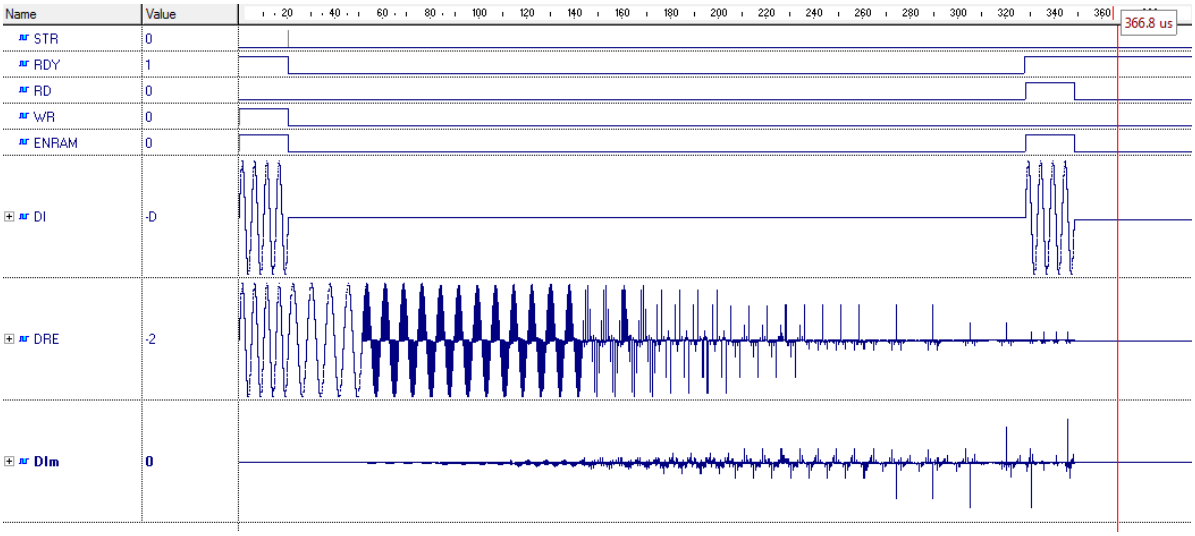


Figura 31 Simulación del proyecto base de la FFT

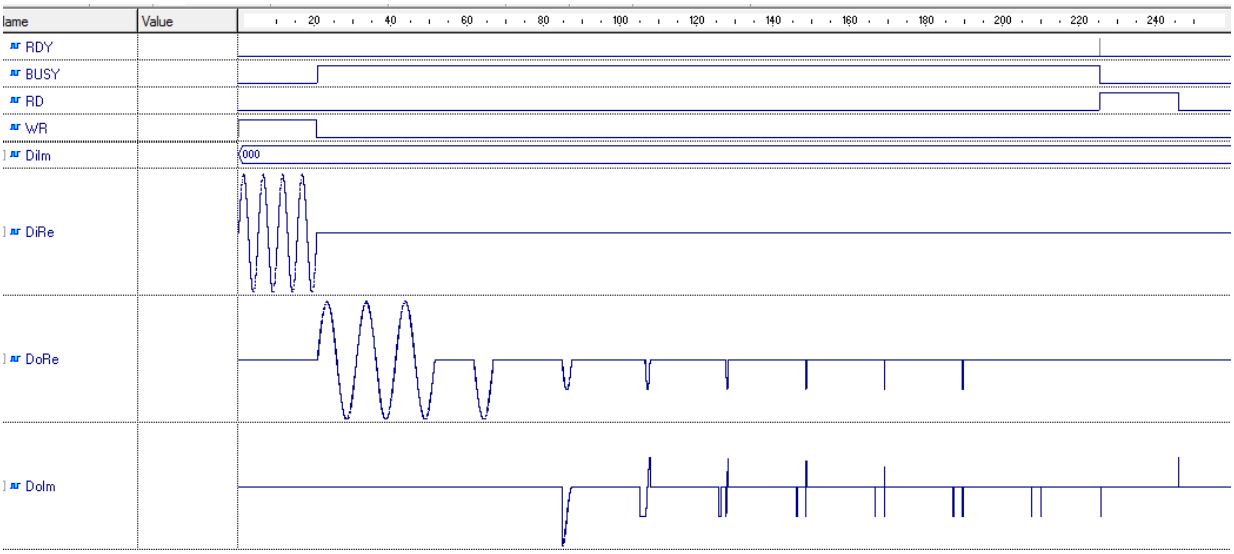


Figura 32 Simulación del IP Core de la FFT

De esta forma se corrobora que efectivamente el IP core muestra una gran mejora con respecto a la FFT tanto en tiempo como en reducción de elementos lógicos de esta forma ya solo es cuestión del diseñador los parámetros con los que quiere trabajar para la conexión del core con la aplicación a realizar.

4.2.4. Documentación

Para crear la primera documentación sobre el proyecto base de la FFT se utilizó el software doxywizard de Doxygen la opción de diseño experto, lo que permite modificar las características de los documentos creados, en la Figura 33 se muestran algunas de las opciones del compilador doxywizard, se debe escribir el nombre del proyecto y describir sus propiedades, se utilizaron también ciertas opciones que permiten mostrar el código completo además de la jerarquización.

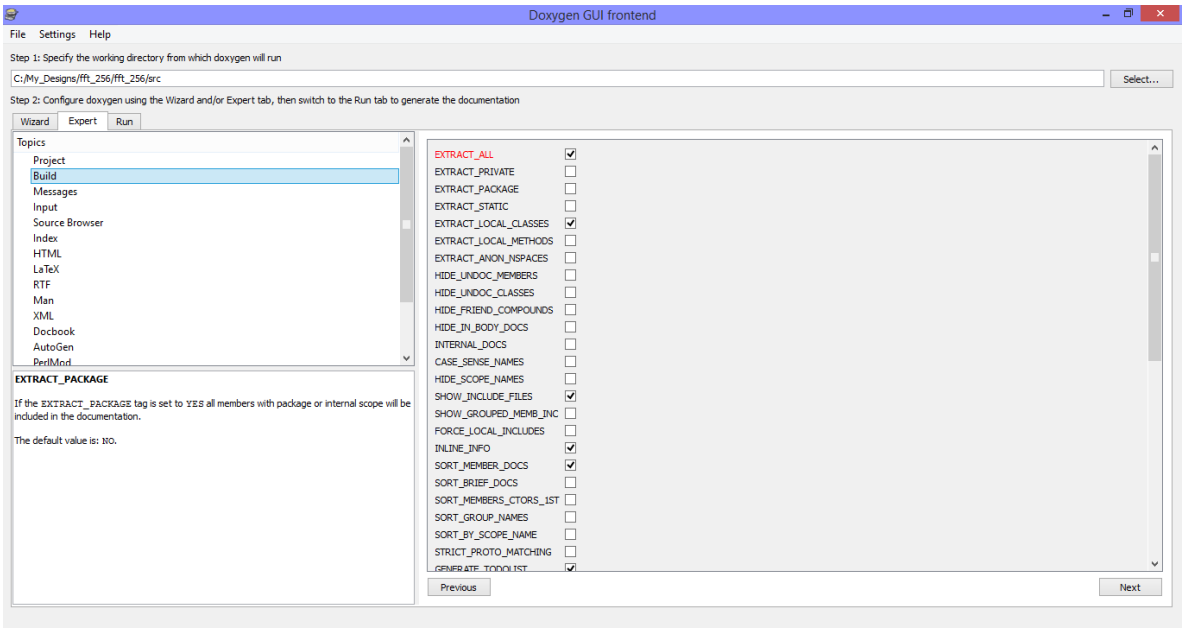


Figura 33 Opciones de compilación del software doxywizard

El resultado de este programa generó los documentos en LaTeX y HTML (HyperText Markup Language). Para los documentos generados en HTML nos brinda un menú interactivo entre los diferentes bloques, lo que facilita el análisis de la estructura del proyecto. En la Figura 34 se muestra un ejemplo:

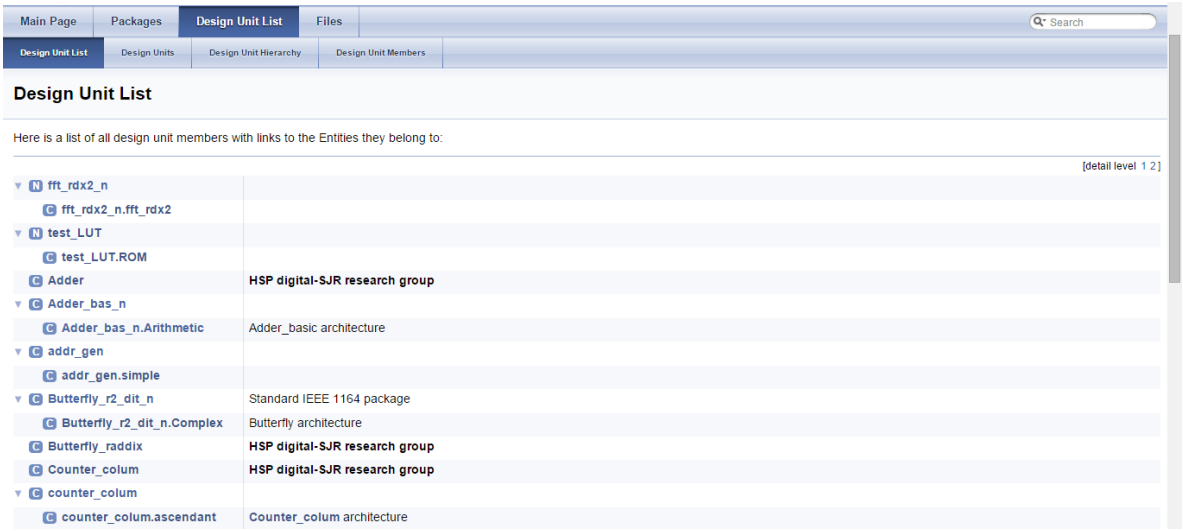


Figura 34 Archivo generado en HTML

En cuanto a los archivos en texto plano o LaTeX se genera un archivo tipo batch que genera las instrucciones para crear un archivo PDF con la documentación proporcionada en Doxywizard y en este caso se realizaron pequeñas modificaciones

después de la compilación del sistema mediante la edición directa del archivo LaTeX para probar la edición externa de los archivos, para evitar la compilación repetitiva se utilizó la herramienta de Tex Works para modificar los archivos en texto plano y generar de nuevo el documento PDF sin la necesidad de compilar todos los archivos de nuevo, en Figura 35 se muestra un ejemplo de esta documentación.

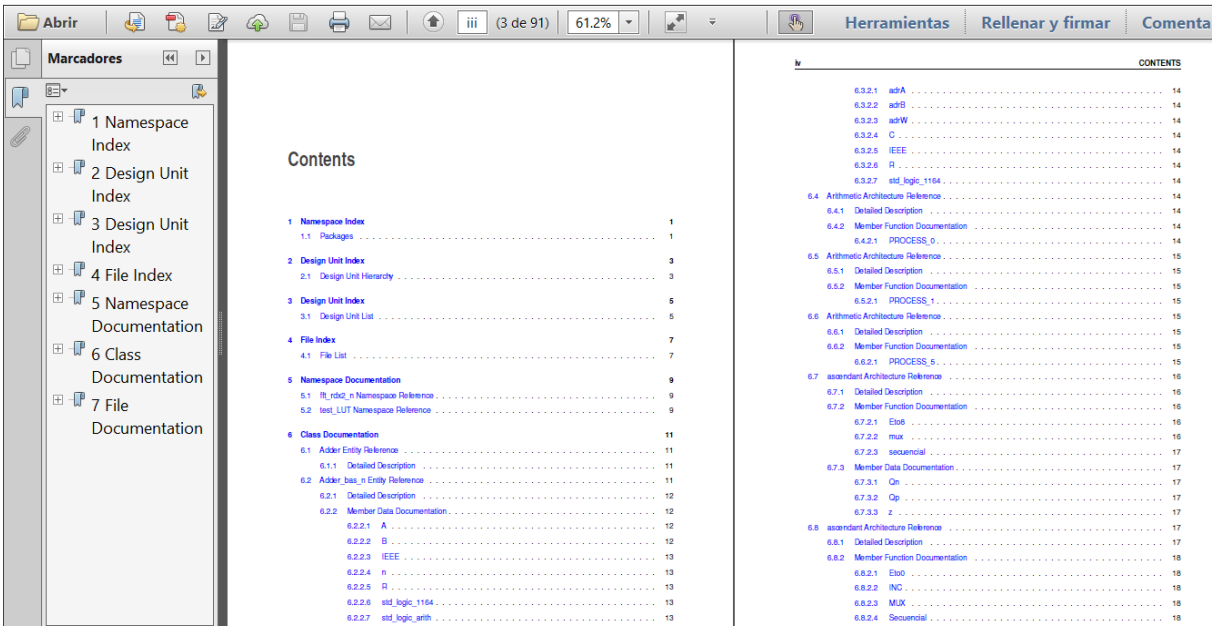


Figura 35 Documento PDF generado por LaTeX

4.2.5. Documentación Final

Para la documentación final del Core se procedió a realizar una edición más completa utilizando las herramientas de Doxywizard solamente, para esto se generaran dos tipos de archivos primero los archivos HTML que permiten una navegación más didáctica y a continuación se generaran los archivo de texto plano para la posterior creación del documento PDF.

CAPÍTULO 5

5. Conclusiones y prospectivas

En este trabajo se presenta la transformada rápida de Fourier en un IP Core desarrollado en base a FPGA, con la ventaja de poder utilizarlo en estructuras más complejas sin necesidad de editarlo. En comparación a otras estructuras se presentan mejoras en la velocidad de cálculo y el ahorro de recursos de la tarjeta al programarla, esto se muestra en el capítulo 4 también se tiene la ventaja de poder trabajar sin

necesidad de sistemas ajenos, esto al integrar las funciones de seno y coseno, las cuales no son muy utilizadas pero que forman parte de las librerías estándar del lenguaje. El Core que se presenta en este trabajo ha sido probado para corroborar su exactitud y el resultado se puede observar en el capítulo 4 en sus dos funciones tanto en la FFT como la IFFT, para su uso como STFT solo es necesario establecer las ventanas en las que se utilizará la FFT según el criterio del diseñador.

Otra ventaja de este Core es que se presenta una documentación especifica del Software, en arquitecturas anteriores no se contaba con los datos suficientes sobre los archivos, lo que generaba problemas para acoplarlas o probarlas, en este trabajo se utilizó el procesador de textos doxywizard para realizar un documento que contenga todos los datos del proyecto de forma más específica, un ejemplo es la separación entre procesos, variables y librerías. Este software ayuda mostrar la estructura del proyecto de forma más clara y completa. Se utilizó este procesador de texto sobre un editor de texto como Word ya que está más especializado hacia códigos de programación, además de ser un software libre y aunque a diferencia del editor no se puede observar los cambios directamente al editar, el resultado es de mejor calidad, así como el ahorro de tiempo para su creación.

Este trabajo tiene como finalidad ayudar a disminuir la carga de trabajo de diseñadores que necesiten realizar un procesamiento de datos como parte de un proyecto más específico o complejo. Quedando a futuro su uso en diferentes sistemas mecatrónicos ya sea en procesamiento de imágenes, calidad de la energía, vibraciones, etc. Dependerá del diseñador el correcto uso del Core.

Otro trabajo interesante seria el desarrollo de un manual del software doxywizard para VHDL en específico pues si bien existe un manual general y algunos ejemplos, estos están más enfocados a lenguajes como C, C+,C# y java. Por lo que los programadores en VHDL tienen más problemas para utilizar todas las herramientas que ofrece este software si no están familiarizados con él.

REFERENCIAS

Banerjee, A., Sundar, A., & Banerjee, S. (2001). FPGA realization of a CORDIC based FFT processor for biomedical signal processing. *Microprocessors and Microsystems*, 131–142.

Cabal-Yepez, E., Garcia-Ramirez, A., Romero-Troncoso, R., Garcia-Perez, A., & Osornio-Rios, R. (2013). Reconfigurable Monitoring System for Time-Frequency Analysis on Industrial Equipment Through STFT and DWT. *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS*, 760 - 771.

Cabal-Yepeza, E., Valtierra-Rodrigueza, M., Romero-Troncoso, R., Garcia-Pereza, A., Osornio-Riosb, R., Miranda-Vidalesc, H., & Alvarez-Salasc, R. (2012). FPGA-based entropy neural processor for online detection of multiple combined faults on induction motors. *Mechanical Systems and Signal Processing*, 123–130.

Conirinthios, M. (s.f.). *Sistems, Signasl, Transform and Digital Signal Processing with Matlab.* CRC Press.

Cortes, J., Mendoza, J., & Muriel, J. (2010). Alternativa al análisis en frecuencia de la FFT mediante el algoritmo Goertzel. *Scientia et Technica*, 217-222.

Frigo, M., & Johnson, S. (1998). FFTW: an adaptive software architecture for the FFT. *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on* (págs. 1381 - 1384). Seattle, WA: IEEE.

Gizopoulos, D., Paschalis, A., & Zorian, Y. (2004). *Embedded Processor-Based Self-Test.* Dordrecht: Kluwer Academic Publishers.

Heesch, D. v. (2014). *doxygen*. Obtenido de http://www.stack.nl/~dimitri/doxygen/

Kıymık, K. M., I., G., Dizibüyüka, A., & Akin, M. (2005). Comparison of STFT and wavelet transform methods in determining epileptic seizure activity in EEG signals for real-time application. *Computers in Biology and Medicine, 35*, 603–616.

Lavry Engineering, Inc. (2004). *Sampling Theory for Digital Audio.* Obtenido de Lavry engineering: http://lavryengineering.com/pdfs/lavry-sampling-theory.pdf

Molino, A., Martina, M., Vacca, F., Masera, G., Terreno, A., Pasquettaz, G., & D'Angelob, G. (2009). FPGA implementation of time–frequency analysis algorithms for laser welding monitoring. *Microprocessors and Microsystems*, 179-190.

Mora, O. G. (2010). *IPCORE controlador PID con comunicación USB basado en tecnologia FPGA.* San Juan del Río.

Nasser, K., & Sidharth, M. (2010). *Digital Signal Processing Laboratory: LabVIEW-Based FPGA Implementation.* Universal-Publishers.

Oppenheim, A. V., Schafer, R., & Buck, J. (1999). *Discrete-Time Signal Processing.* New Jersey: Prentice-Hall.

Proakis, J. G., & Manolakis, D. G. (1996). *Digital Signal Processing Principle, Algorithms and Applications.* New Jersey: Prentice- Hall International.

Qin, S., & Zhong, Y. (2004). Research on the unified mathematical model for FT, STFT and WT and its applications. *Mechanical Systems and Signal Processing*, 1335–1347.

Rangel-Magdaleno, J., Romero-Troncoso, R., Osornio-Rios, R., Cabal-Yepez, E., & Contreras-Medina, L. (2009). Novel Methodology for Online Half-Broken-Bar Detection on Induction Motors. *Instrumentation and Measurement, IEEE Transactions on*, 1690 - 1698.

Romero Troncoso, R. d. (2007). *Electrónica Digital y Lógica Programable .* Guanajuato: Universidad de Guanajuato.

Salazar, A., & Vergara, L. (2010). ICA mixtures applied to ultrasonic nondestructive classification of archaeological ceramics. *EURASIP Journal on Advances in Signal Processing, 2010*, 8.

Smith, S. W. (1999). *The Scientist and Engineer's Guide to Digital Signal Processing.* California: California Technical Publishing.

Vai, M. M. (2001). *VLSI Design.* Florida: CRC Press.

APÉNDICE

En esta sección se añaden la configuración utilizada para documentar el Core y el resultado obtenido mediante el software doxywizard, se mostraran los resultados más importantes del documento.

Configuración Doxywizard

A continuación se muestran las opciones utilizadas en el software doxywizard para generar la documentación actual del IP Core de la FFT.

Doxyfile 1.8.9.1

Project related configuration options

```
DOXYFILE_ENCODING      = UTF-8
PROJECT_NAME           = FFT_rdx2
PROJECT_NUMBER         = 7
PROJECT_BRIEF          = "Fast Fourier Transform & inverse"
PROJECT_LOGO  = C:/Users/JEANKARLO/Pictures/imagenes/fi.png
OUTPUT_DIRECTORY       = ../../files_one/complete
CREATE_SUBDIRS         = YES
ALLOW_UNICODE_NAMES    = NO
OUTPUT_LANGUAGE        = English
BRIEF_MEMBER_DESC      = YES
REPEAT_BRIEF           = YES
ABBREVIATE_BRIEF       = "The $name class" \
                         "The $name widget" \
                         "The $name file" \
                         is \
                         provides \
                         specifies \
                         contains \
                         represents \
```

ALWAYS_DETAILED_SEC = YES

INLINE_INHERITED_MEMB = NO

FULL_PATH_NAMES = YES

SHORT_NAMES = YES

JAVADOC_AUTOBRIEF = NO

QT_AUTOBRIEF = NO

MULTILINE_CPP_IS_BRIEF = NO

INHERIT_DOCS = YES

SEPARATE_MEMBER_PAGES = NO

TAB_SIZE = 4

OPTIMIZE_OUTPUT_FOR_C = NO

OPTIMIZE_OUTPUT_JAVA = NO

OPTIMIZE_FOR_FORTRAN = NO

OPTIMIZE_OUTPUT_VHDL = YES

MARKDOWN_SUPPORT = YES

AUTOLINK_SUPPORT = YES

BUILTIN_STL_SUPPORT = NO

CPP_CLI_SUPPORT = NO

SIP_SUPPORT = NO

IDL_PROPERTY_SUPPORT = YES

DISTRIBUTE_GROUP_DOC = YES

SUBGROUPING = YES

INLINE_GROUPED_CLASSES = YES

INLINE_SIMPLE_STRUCTS = NO

TYPEDEF_HIDES_STRUCT = NO

LOOKUP_CACHE_SIZE = 0

Build related configuration options

EXTRACT_ALL = YES

EXTRACT_PRIVATE = NO

EXTRACT_PACKAGE = NO

```
EXTRACT_STATIC         = NO
EXTRACT_LOCAL_CLASSES  = YES
EXTRACT_LOCAL_METHODS  = NO
EXTRACT_ANON_NSPACES   = NO
HIDE_UNDOC_MEMBERS     = NO
HIDE_UNDOC_CLASSES     = NO
HIDE_FRIEND_COMPOUNDS  = NO
HIDE_IN_BODY_DOCS      = NO
INTERNAL_DOCS          = NO
CASE_SENSE_NAMES       = YES
HIDE_SCOPE_NAMES       = NO
HIDE_COMPOUND_REFERENCE= NO
SHOW_INCLUDE_FILES     = YES
SHOW_GROUPED_MEMB_INC  = NO
FORCE_LOCAL_INCLUDES   = NO
INLINE_INFO            = YES
SORT_MEMBER_DOCS       = YES
SORT_BRIEF_DOCS        = NO
SORT_MEMBERS_CTORS_1ST = NO
SORT_GROUP_NAMES       = NO
SORT_BY_SCOPE_NAME     = NO
STRICT_PROTO_MATCHING  = NO
GENERATE_TODOLIST      = YES
GENERATE_TESTLIST      = YES
GENERATE_BUGLIST       = YES
GENERATE_DEPRECATEDLIST= YES
MAX_INITIALIZER_LINES  = 30
SHOW_USED_FILES        = YES
SHOW_FILES             = YES
SHOW_NAMESPACES        = YES
```

Configuration options related to warning and progress messages

```
QUIET                = NO
WARNINGS             = YES
WARN_IF_UNDOCUMENTED   = YES
WARN_IF_DOC_ERROR      = YES
WARN_NO_PARAMDOC       = NO
WARN_FORMAT            = "$file:$line: $text"
```

Configuration options related to the input files

```
INPUT                = .
INPUT_ENCODING         = UTF-8
FILE_PATTERNS          =
                     *.vhd \
                     *.vhdl \
RECURSIVE              = YES
EXCLUDE                = fft_example.m
EXCLUDE_SYMLINKS       = NO
EXAMPLE_PATTERNS       = *
EXAMPLE_RECURSIVE      = NO
IMAGE_PATH             = Address_gen.eps \
                     Address_gen.png \
                     FFT_algorithm.eps \
                     FFT_algorithm.jpg \
                     FFT_rd2.eps \
                     FFT_rd2.jpg \
                     RAM.eps \
                     RAM.png
FILTER_SOURCE_FILES    = NO
```

Configuration options related to source browsing

```
SOURCE_BROWSER         = YES
INLINE_SOURCES         = YES
```

```
STRIP_CODE_COMMENTS    = YES
REFERENCED_BY_RELATION = YES
REFERENCES_RELATION    = YES
REFERENCES_LINK_SOURCE = YES
SOURCE_TOOLTIPS        = YES
USE_HTAGS              = NO
VERBATIM_HEADERS       = YES
CLANG_ASSISTED_PARSING = NO
```

Configuration options related to the alphabetical class index

```
ALPHABETICAL_INDEX     = YES
COLS_IN_ALPHA_INDEX    = 3
```

Configuration options related to the HTML output

```
GENERATE_HTML          = YES
HTML_OUTPUT            = html
HTML_FILE_EXTENSION    = .html
HTML_EXTRA_FILES       = Address_gen.png \
                         FFT_algorithm.jpg \
                         FFT_rd2.jpg \
                         RAM.png
HTML_COLORSTYLE_HUE    = 220
HTML_COLORSTYLE_SAT    = 53
HTML_COLORSTYLE_GAMMA  = 117
HTML_TIMESTAMP         = YES
HTML_DYNAMIC_SECTIONS  = YES
HTML_INDEX_NUM_ENTRIES = 100
GENERATE_DOCSET        = NO
DOCSET_FEEDNAME        = "Doxygen generated docs"
DOCSET_BUNDLE_ID       = org.doxygen.Project
DOCSET_PUBLISHER_ID    = org.doxygen.Publisher
DOCSET_PUBLISHER_NAME  = Publisher
GENERATE_HTMLHELP      = NO
```

```
GENERATE_CHI           = NO
BINARY_TOC             = NO
TOC_EXPAND             = NO
GENERATE_QHP           = NO
QHP_NAMESPACE          = org.doxygen.Project
QHP_VIRTUAL_FOLDER     = doc
GENERATE_ECLIPSEHELP   = NO
ECLIPSE_DOC_ID         = org.doxygen.Project
DISABLE_INDEX          = NO
GENERATE_TREEVIEW      = YES
ENUM_VALUES_PER_LINE   = 4
TREEVIEW_WIDTH         = 250
EXT_LINKS_IN_WINDOW    = NO
FORMULA_FONTSIZE       = 10
FORMULA_TRANSPARENT    = YES
USE_MATHJAX            = YES
MATHJAX_FORMAT         = HTML-CSS
MATHJAX_RELPATH        = http://cdn.mathjax.org/mathjax/latest
SEARCHENGINE           = YES
SERVER_BASED_SEARCH    = NO
EXTERNAL_SEARCH        = NO
SEARCHDATA_FILE        = searchdata.xml
```

Configuration options related to the LaTeX output

```
GENERATE_LATEX         = YES
LATEX_OUTPUT           = latex
LATEX_CMD_NAME         = latex
MAKEINDEX_CMD_NAME     = makeindex
COMPACT_LATEX          = NO
PAPER_TYPE             = letter
LATEX_EXTRA_FILES      = Address_gen.eps \
                         FFT_algorithm.eps \
```

PDF_HYPERLINKS = YES USE_PDFLATEX = YES LATEX_BATCHMODE = YES LATEX_HIDE_INDICES = NO LATEX_SOURCE_CODE = YES LATEX_BIB_STYLE = plain
Configuration options related to the RTF output
GENERATE_RTF = NO RTF_OUTPUT = rtf COMPACT_RTF = NO RTF_HYPERLINKS = NO RTF_SOURCE_CODE = NO
Configuration options related to the man page output
GENERATE_MAN = NO MAN_OUTPUT = man MAN_EXTENSION = .3 MAN_LINKS = NO
Configuration options related to the XML output
GENERATE_XML = NO XML_OUTPUT = xml XML_PROGRAMLISTING = YES
Configuration options related to the DOCBOOK output
GENERATE_DOCBOOK = NO DOCBOOK_OUTPUT = docbook DOCBOOK_PROGRAMLISTING = NO
Configuration options for the AutoGen Definitions output
GENERATE_AUTOGEN_DEF = NO
Configuration options related to the Perl module output
GENERATE_PERLMOD = NO PERLMOD_LATEX = NO PERLMOD_PRETTY = YES

Configuration options related to the preprocessor
ENABLE_PREPROCESSING = YES MACRO_EXPANSION = NO EXPAND_ONLY_PREDEF = NO SEARCH_INCLUDES = YES INCLUDE_PATH = C:/Users/JEANKARLO/Documents/CORE/CORE/FFTnew SKIP_FUNCTION_MACROS = YES
Configuration options related to external references
ALLEXTERNALS = NO EXTERNAL_GROUPS = YES EXTERNAL_PAGES = YES PERL_PATH = /usr/bin/perl
Configuration options related to the dot tool
CLASS_DIAGRAMS = YES HIDE_UNDOC_RELATIONS = YES HAVE_DOT = YES DOT_NUM_THREADS = 0 DOT_FONTNAME = Helvetica DOT_FONTSIZE = 10 CLASS_GRAPH = YES COLLABORATION_GRAPH = YES GROUP_GRAPHS = YES UML_LOOK = NO UML_LIMIT_NUM_FIELDS = 10 TEMPLATE_RELATIONS = NO INCLUDE_GRAPH = YES INCLUDED_BY_GRAPH = YES CALL_GRAPH = NO CALLER_GRAPH = NO GRAPHICAL_HIERARCHY = YES DIRECTORY_GRAPH = YES

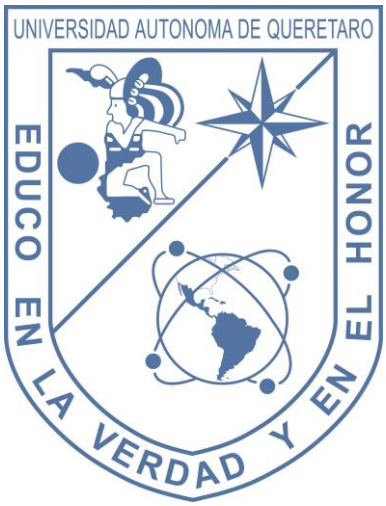
```
DOT_IMAGE_FORMAT       = svg
INTERACTIVE_SVG        = YES
DOT_PATH               = "../../../../../../Program Files (x86)/Graphviz2.38/bin"
DOT_GRAPH_MAX_NODES    = 50
MAX_DOT_GRAPH_DEPTH    = 0
DOT_TRANSPARENT        = NO
DOT_MULTI_TARGETS      = NO
GENERATE_LEGEND        = YES
DOT_CLEANUP            = YES
```

Documento del IP core de la FFT

En esta sección se presenta los resultados más relevantes obtenidos mediante el software doxywizard se observan las diferentes secciones que se generan y la separación de variables, estructuras, librerías y los índices obtenidos.

Chapter 1

IP CORE FFT &STFT



IP Core FFT

Generated by Doxygen 1.8.9.1
November 2015

Contents

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Design Unit Index

3.1 Design Unit Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 7

Class Documentation

7.1 Addres_Generator Entity Reference

7.1.1 Detailed Description

fftAddresGen description:

The addres generator fix the index to separate even from odd numbers.

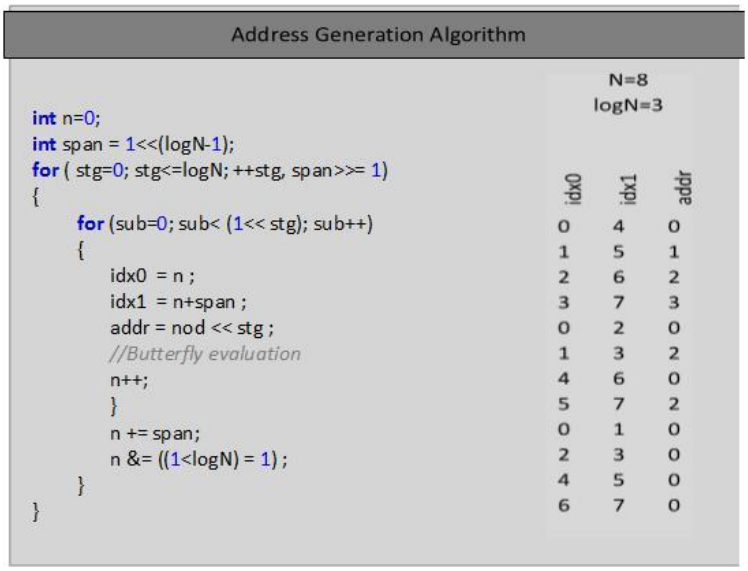


Figure 7.1: Generator

Detailed description of this

The documentation for this class was generated from the following file:

- fftAddresGen.vhd

7.2 counter Entity Reference

The documentation for this class was generated from the following file:

- fftCount.vhd

7.3 Dual_Ram Entity Reference

Generics dual port RAM.

7.3.1 Detailed Description

Generics dual port RAM.

This module implement a general dual port RAM

The documentation for this class was generated from the following file:

- fftDualRAM.vhd

7.4 fftAddresGen Architecture Reference

Components

FFT address generator

Author

Luis Morales Velazquez.
Jean Karlo Gomez Reyes.

Date

November 2015.

- fftCounter
- fftCount
- fftBarrelShifter

Constants

FFT address generator

Author

Luis Morales Velazquez.
Jean Karlo Gomez Reyes.

Date

November 2015.

- D1 std_logic_vector(m - 1 downto 0):=(others=>' 0 ')
- D2 std_logic_vector(n - 1 downto 0):=(others=>' 0 ')
- D3 std_logic_vector(n - 2 downto 0):=(others=>' 0 ')

Types

FFT address generator

Author

Luis Morales Velazquez.
Jean Karlo Gomez Reyes.

Date

November 2015.

- shift_typearray(naturalrange<>)ofstd_logic_vector(n - 2 downto 0)

Signals

FFT address generator

Author

Luis Morales Velazquez.
Jean Karlo Gomez Reyes.

Date

November 2015.

- ES std_logic
- EU std_logic
- EN std_logic
- OPC1 std_logic_vector(1 downto 0)
- OPC2 std_logic_vector(1 downto 0)
- OPC3 std_logic_vector(1 downto 0)
- K1 std_logic_vector(m - 1 downto 0)
- STG std_logic_vector(m - 1 downto 0)
- K2 std_logic_vector(n - 1 downto 0)
- IDX std_logic_vector(n - 1 downto 0)
- IDN std_logic_vector(n - 1 downto 0)
- IDS std_logic_vector(n - 1 downto 0)
- K3 std_logic_vector(n - 2 downto 0)
- NOD std_logic_vector(n - 2 downto 0)
- ADD std_logic_vector(n - 2 downto 0)
- QNS std_logic_vector(n - 1 downto 0)
- QPS std_logic_vector(n - 1 downto 0)
- QNM std_logic_vector(n - 1 downto 0)
- QPM std_logic_vector(n - 1 downto 0)

Instantiations

- stage **fftCounter**
- submat **fftCount**
- node **fftCounter**
- shifter **fftBarrelShifter**

7.4.1 Detailed Description

Addres Generator Architecture

Definition at line 46 of file fftAddresGen.vhd.

7.4.2 Member Data Documentation

7.4.2.1 ADD std_logic_vector(n - 2 downto 0) [Signal]

Definition at line 91 of file fftAddresGen.vhd.

7.4.2.2 D1 std_logic_vector(m - 1 downto 0):=(others=>' 0 ') [Constant]

Definition at line 83 of file fftAddresGen.vhd.

7.4.2.3 D2 std_logic_vector(n - 1 downto 0):=(others=>' 0 ') [Constant]

Definition at line 84 of file fftAddresGen.vhd.

7.4.2.4 D3 std_logic_vector(n - 2 downto 0):=(others=>' 0 ') [Constant]

Definition at line 85 of file fftAddresGen.vhd.

7.4.2.5 EN std_logic [Signal]

Definition at line 87 of file fftAddresGen.vhd.

7.4.2.6 ES std_logic [Signal]

Definition at line 87 of file fftAddresGen.vhd.

7.4.2.7 EU std_logic [Signal]

Definition at line 87 of file fftAddresGen.vhd.

7.4.2.8 fftBarrelShifter [Component]

Definition at line 74 of file fftAddresGen.vhd.

7.4.2.9 fftCount [Component]

Definition at line 63 of file fftAddresGen.vhd.

7.4.2.10 fftCounter [Component]

Definition at line 51 of file fftAddresGen.vhd.

7.4.2.11 IDN std_logic_vector(n - 1 downto 0) [Signal]

Definition at line 90 of file fftAddresGen.vhd.

7.4.2.12 IDS std_logic_vector(n - 1 downto 0) [Signal]

Definition at line 90 of file fftAddresGen.vhd.

7.4.2.13 IDX std_logic_vector(n - 1 downto 0) [Signal]

Definition at line 90 of file fftAddresGen.vhd.

7.4.2.14 K1 std_logic_vector(m - 1 downto 0) [Signal]

Definition at line 89 of file fftAddresGen.vhd.

7.4.2.15 K2 std_logic_vector(n - 1 downto 0) [Signal]

Definition at line 90 of file fftAddresGen.vhd.

7.4.2.16 K3 std_logic_vector(n - 2 downto 0) [Signal]

Definition at line 91 of file fftAddresGen.vhd.

7.4.2.17 NOD std_logic_vector(n - 2 downto 0) [Signal]

Definition at line 91 of file fftAddresGen.vhd.

7.4.2.18 node fftCounter [Instantiation]

Definition at line 116 of file fftAddresGen.vhd.

7.4.2.19 OPC1 std_logic_vector(1 downto 0) [Signal]

Definition at line 88 of file fftAddresGen.vhd.

7.4.2.20 OPC2 std_logic_vector(1 downto 0) [Signal]

Definition at line 88 of file fftAddresGen.vhd.

7.4.2.21 OPC3 std_logic_vector(1 downto 0) [Signal]

Definition at line 88 of file fftAddresGen.vhd.

7.4.2.22 QNM std_logic_vector(n - 1 downto 0) [Signal]

Definition at line 92 of file fftAddresGen.vhd.

7.4.2.23 QNS std_logic_vector(n - 1 downto 0) [Signal]

Definition at line 92 of file fftAddresGen.vhd.

7.4.2.24 QPM std_logic_vector(n - 1 downto 0) [Signal]

Definition at line 92 of file fftAddresGen.vhd.

7.4.2.25 QPS std_logic_vector(n - 1 downto 0) [Signal]

Definition at line 92 of file fftAddresGen.vhd.

7.4.2.26 shift_type array(natural range<>)of std_logic_vector(n - 2 downto 0) [Type]

Definition at line 48 of file fftAddresGen.vhd.

7.4.2.27 shifter fftBarrelShifter [Instantiation]

Definition at line 118 of file fftAddresGen.vhd.

7.4.2.28 stage fftCounter [Instantiation]

Definition at line 112 of file fftAddresGen.vhd.

7.4.2.29 STG std_logic_vector(m - 1 downto 0) [Signal]

Definition at line 89 of file fftAddresGen.vhd.

7.4.2.30 submat **fftCount** [Instantiation]

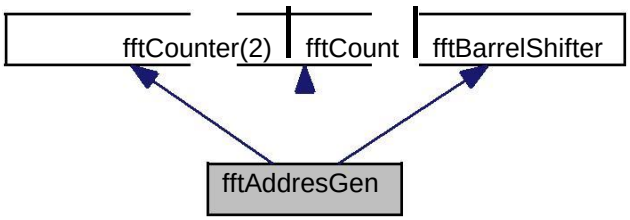
Definition at line 114 of file fftAddresGen.vhd.

The documentation for this class was generated from the following file:

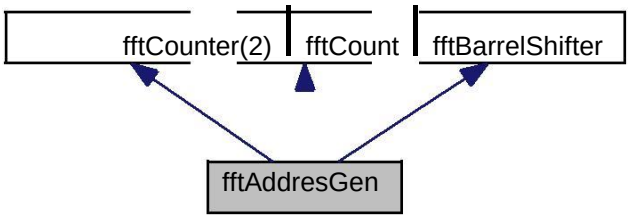
- fftAddresGen.vhd

7.5 fftAddresGen Entity Reference

Inheritance diagram for fftAddresGen:



Collaboration diagram for fftAddresGen:



Entities

- fftAddresGen architecture

Libraries

FFT address generator

Author

Luis Morales Velazquez.
Jean Karlo Gomez Reyes.

Date

November 2015.

- IEEE
  Standard IEEE library.

Use Clauses

FFT address generator

Author

Luis Morales Velazquez.
Jean Karlo Gomez Reyes.

Date

November 2015.

- STD_LOGIC_1164
  Standard IEEE 1164 package.
- NUMERIC_STD
  Standard IEEE Numeric package.

Generics

FFT address generator

Author

Luis Morales Velazquez.
Jean Karlo Gomez Reyes.

Date

November 2015.

- np **integer**:= **8**

Number of total points.

- n **integer:=** 3

  Number of bits to address np (log2(np)) => n = $\log_2(N)$.

- m **integer:=** 2

  Number of FFT stages.

Ports

FFT address generator

Author

Luis Morales Velazquez.
Jean Karlo Gomez Reyes.

Date

November 2015.

- RST **in STD_LOGIC**

  Number of total points.

- CLK **in STD_LOGIC**

  Number of total points.

- CS **in STD_LOGIC**

  Number of total points.

- ENI **in STD_LOGIC**

  Number of total points.

- EOC **out STD_LOGIC**

  Number of total points.

- IDX0 **out STD_LOGIC_VECTOR(n - 1 downto 0)**

  Number of total points.

- IDX1 **out STD_LOGIC_VECTOR(n - 1 downto 0)**

  Number of total points.

- ADDR **out STD_LOGIC_VECTOR(n - 2 downto 0)**

  Number of total points.

7.5.1 Detailed Description

fftAddres entity

Definition at line 25 of file fftAddresGen.vhd.

7.5.2 Member Data Documentation

7.5.2.1 ADDR out STD_LOGIC_VECTOR(n - 2 downto 0) [Port]

Number of total points.

Definition at line 43 of file fftAddresGen.vhd.

7.5.2.2 CLK in STD_LOGIC [Port]

Number of total points.

Definition at line 36 of file fftAddresGen.vhd.

7.5.2.3 CS in STD_LOGIC [Port]

Number of total points.

Definition at line 37 of file fftAddresGen.vhd.

7.5.2.4 ENI in STD_LOGIC [Port]

Number of total points.

Definition at line 38 of file fftAddresGen.vhd.

7.5.2.5 EOC out STD_LOGIC [Port]

Number of total points.

Definition at line 39 of file fftAddresGen.vhd.

7.5.2.6 IDX0 out STD_LOGIC_VECTOR(n - 1 downto 0) [Port]

Number of total points.

Definition at line 40 of file fftAddresGen.vhd.

7.5.2.7 IDX1 out STD_LOGIC_VECTOR(n - 1 downto 0) [Port]

Number of total points.

Definition at line 41 of file fftAddresGen.vhd.

7.5.2.8 IEEE [Library]

Standard IEEE library.

Definition at line 11 of file fftAddresGen.vhd.

7.5.2.9 m integer:= 2 [Generic]

Number of FFT stages.

Definition at line 33 of file fftAddresGen.vhd.

7.5.2.10 n integer:= 3 [Generic]

Number of bits to address np (log2(np)) => n = log2(N).

Definition at line 31 of file fftAddresGen.vhd.

7.5.2.11 np integer:= 8 [Generic]

Number of total points.

Definition at line 29 of file fftAddresGen.vhd.

7.5.2.12 NUMERIC_STD [Package]

Standard IEEE Numeric package.

Definition at line 15 of file fftAddresGen.vhd.

7.5.2.13 RST in STD_LOGIC [Port]

Number of total points.

Definition at line 35 of file fftAddresGen.vhd.

7.5.2.14 STD_LOGIC_1164 [Package]

Standard IEEE 1164 package.

Definition at line 13 of file fftAddresGen.vhd.

The documentation for this class was generated from the following file:

- fftAddresGen.vhd

7.6 fftBarrelShifter Architecture Reference

Barrel Shifter architecture declaration.

Types

- **shift_typearray(naturalrange<>)ofstd_logic_vector(n - 1 downto 0)**

  Array.

Signals

- DS **shift_type (0 to m)**

  Signal DS.
- DM **shift_type (0 to m - 1)**

  Sigmal DM.

7.6.1 Detailed Description

Barrel Shifter architecture declaration.

Definition at line 34 of file fftBarrelShifter.vhd.

7.6.2 Member Data Documentation

7.6.2.1 DM shift_type (0 tom - 1) [Signal]

Sigmal DM.

Definition at line 37 of file fftBarrelShifter.vhd.

7.6.2.2 DS shift_type (0 tom) [Signal]

Signal DS.

Definition at line 36 of file fftBarrelShifter.vhd.

7.6.2.3 shift_type array(naturalrange<>)ofstd_logic_vector(n - 1 downto 0) [Type]

Array.

Definition at line 35 of file fftBarrelShifter.vhd.

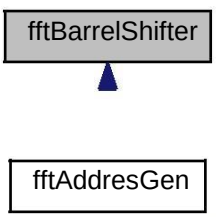
The documentation for this class was generated from the following file:

- fftBarrelShifter.vhd

7.7 fftBarrelShifter Entity Reference

Standard IEEE 1164 package Barrel Shifter entity declaration.

Inheritance diagram for fftBarrelShifter:



Entities

- fftBarrelShifter architecture

  Barrel Shifter architecture declaration.

Libraries

FFT address generator

Author

Luis Morales Velazquez.
Jean Karlo Gomez Reyes.

Date

November 2015.

- IEEE

  Adress generator.

Use Clauses

- STD_LOGIC_1164

  Standard IEEE library.

Generics

- n **integer**:= 5

  Number of data bits.
- m **integer**:= 3

  Number of shifter bits.

Ports

- SHF **in** **STD_LOGIC_VECTOR**(**m** - 1 **downto** 0)

  Shifter vector.
- Di **in** **STD_LOGIC_VECTOR**(**n** - 1 **downto** 0)

  Input vector shifter.
- Do **out** **STD_LOGIC_VECTOR**(**n** - 1 **downto** 0)

  Output vector shifter.

7.7.1 Detailed Description

Standard IEEE 1164 package Barrel Shifter entity declaration.

Definition at line 23 of file fftBarrelShifter.vhd.

7.7.2 Member Data Documentation

7.7.2.1 Di in STD_LOGIC_VECTOR(n - 1 downto 0) [Port]

Input vector shifter.

Definition at line 28 of file fftBarrelShifter.vhd.

7.7.2.2 Do out STD_LOGIC_VECTOR(n - 1 downto 0) [Port]

Output vector shifter.

Definition at line 30 of file fftBarrelShifter.vhd.

7.7.2.3 IEEE [Library]

Adress generator.

Definition at line 17 of file fftBarrelShifter.vhd.

7.7.2.4 m integer:= 3 [Generic]

Number of shifter bits.

Definition at line 25 of file fftBarrelShifter.vhd.

7.7.2.5 n integer:= 5 [Generic]

Number of data bits.

Definition at line 24 of file fftBarrelShifter.vhd.

7.7.2.6 SHF in STD_LOGIC_VECTOR(m - 1 downto 0) [Port]

Shifter vector.

Definition at line 27 of file fftBarrelShifter.vhd.

7.7.2.7 STD_LOGIC_1164 [Package]

Standard IEEE library.

Definition at line 19 of file fftBarrelShifter.vhd.

The documentation for this class was generated from the following file:

- fftBarrelShifter.vhd

7.8 fftButterflyR2DIF Entity Reference

Standard IEEE numerical real package The architecture of the algorithm is :

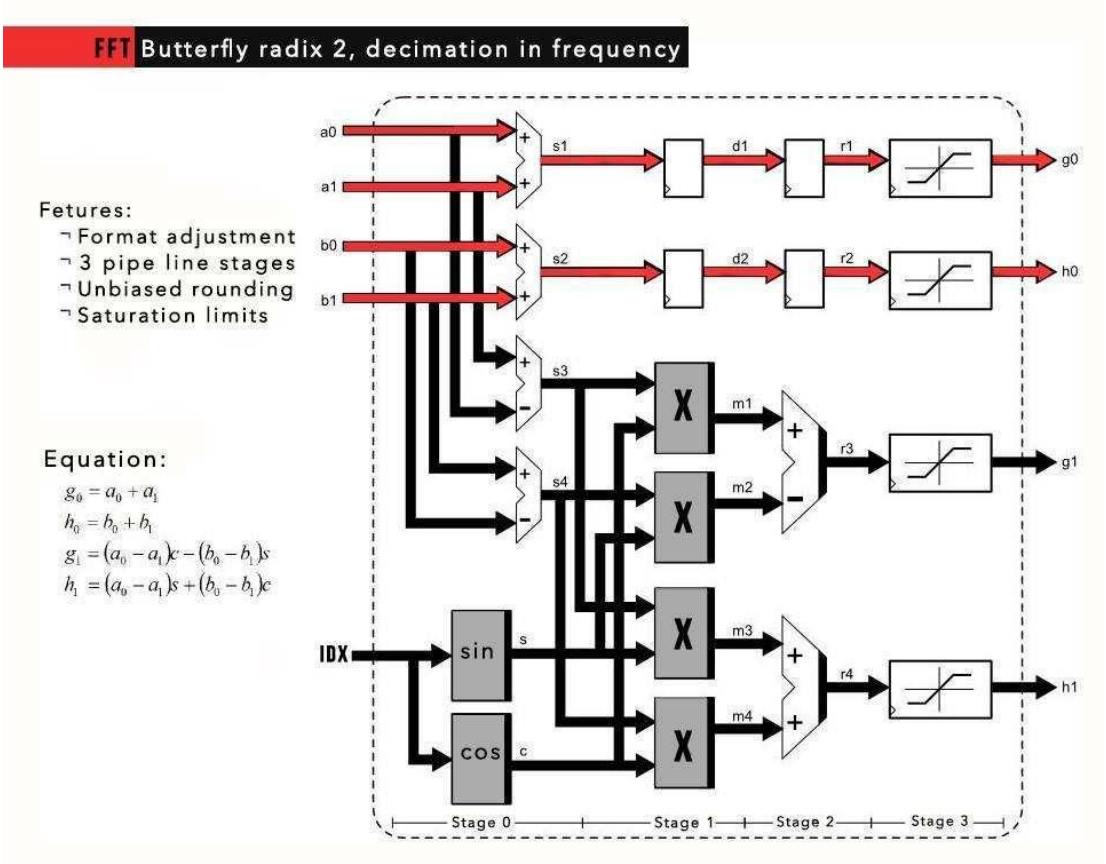
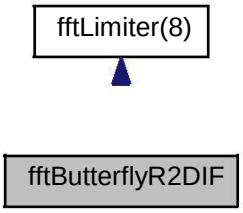


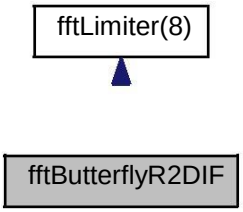
Figure 7.2: butterfly raddix 2

.

Inheritance diagram for fftButterflyR2DIF:



Collaboration diagram for fftButterflyR2DIF:



Entities

- fftButterflyR2DIF architecture

  architecture fftButterflyR2DIF of fftButterflyR2DIF is

Libraries

- IEEE

Use Clauses

- STD_LOGIC_1164

  Standard IEEE library.
- NUMERIC_STD

  Standard IEEE 1164 package.
- MATH_REAL

  Standard IEEE numeric package.

Generics

- e **integer:= 2**

  Format integer.
- f **integer:= 10**

  Format float.
- n **integer:= 8**

  Number of points.
- m **integer:= 2**

  Number of index bits.
- ifft **boolean:=false**

  Set inverse FFT.

7.11.2.4 IEEE [Library]

Standard IEEE library.

Definition at line 14 of file fftCount.vhd.

7.11.2.5 K in STD_LOGIC_VECTOR(n - 1 downto 0) [Port]

Standard IEEE library.

Definition at line 28 of file fftCount.vhd.

7.11.2.6 n integer:= 8 [Generic]

Standard IEEE library.

Definition at line 22 of file fftCount.vhd.

7.11.2.7 NUMERIC_STD [Package]

Standard IEEE numeric package.

Definition at line 18 of file fftCount.vhd.

7.11.2.8 OPC in STD_LOGIC_VECTOR(1 downto 0) [Port]

Standard IEEE library.

Definition at line 26 of file fftCount.vhd.

7.11.2.9 RST in STD_LOGIC [Port]

Standard IEEE library.

Definition at line 24 of file fftCount.vhd.

7.11.2.10 STD_LOGIC_1164 [Package]

Standard IEEE 1164 package.

Definition at line 16 of file fftCount.vhd.

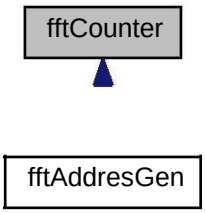
The documentation for this class was generated from the following file:

- fftCount.vhd

7.12 fftCounter Entity Reference

Counter entity declaration.

Inheritance diagram for fftCounter:



Entities

- fftCounter architecture

  Counter architecture declaration.

Libraries

FFT generic counter

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

Date

November 2015.

- IEEE
  Standard IEEE library.

Use Clauses

FFT generic counter

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

Date

November 2015.

- STD_LOGIC_1164
  Standard IEEE 1164 package.
- NUMERIC_STD
  Standard IEEE numeric package.

Generics

FFT generic counter

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

Date

November 2015.

- n **integer**:= 8
  Standard IEEE library.

Ports

FFT generic counter

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

---

Date

November 2015.

- RST **in** **STD_LOGIC**

  Standard IEEE library.
- CLK **in** **STD_LOGIC**

  Standard IEEE library.
- OPC **in** **STD_LOGIC_VECTOR(1 downto 0)**

  Standard IEEE library.
- D **in** **STD_LOGIC_VECTOR(n - 1 downto 0)**

  Standard IEEE library.
- K **in** **STD_LOGIC_VECTOR(n - 1 downto 0)**

  Standard IEEE library.
- E **out** **STD_LOGIC**

  Standard IEEE library.
- Q **out** **STD_LOGIC_VECTOR(n - 1 downto 0)**

  Standard IEEE library.

7.12.1 Detailed Description

Counter entity declaration.

Definition at line 20 of file fftCounter.vhd.

7.12.2 Member Data Documentation

7.12.2.1 CLK in STD_LOGIC [Port]

Standard IEEE library.

Definition at line 24 of file fftCounter.vhd.

7.12.2.2 D in STD_LOGIC_VECTOR(n - 1 downto 0) [Port]

Standard IEEE library.

Definition at line 26 of file fftCounter.vhd.

7.12.2.3 E out STD_LOGIC [Port]

Standard IEEE library.

Definition at line 28 of file fftCounter.vhd.

7.12.2.4 IEEE [Library]

Standard IEEE library.

Definition at line 13 of file fftCounter.vhd.

7.12.2.5 K in STD_LOGIC_VECTOR(n - 1 downto 0) [Port]

Standard IEEE library.

Definition at line 27 of file fftCounter.vhd.

7.12.2.6 n integer:= 8 [Generic]

Standard IEEE library.

Definition at line 21 of file fftCounter.vhd.

7.12.2.7 NUMERIC_STD [Package]

Standard IEEE numeric package.

Definition at line 17 of file fftCounter.vhd.

7.12.2.8 OPC in STD_LOGIC_VECTOR(1 downto 0) [Port]

Standard IEEE library.

Definition at line 25 of file fftCounter.vhd.

7.12.2.9 Q out STD_LOGIC_VECTOR(n - 1 downto 0) [Port]

Standard IEEE library.

Definition at line 30 of file fftCounter.vhd.

7.12.2.10 RST in STD_LOGIC [Port]

Standard IEEE library.

Definition at line 23 of file fftCounter.vhd.

7.12.2.11 STD_LOGIC_1164 [Package]

Standard IEEE 1164 package.

Definition at line 15 of file fftCounter.vhd.

The documentation for this class was generated from the following file:

- fftCounter.vhd

7.13 fftCounter Architecture Reference

Counter architecture declaration.

Signals

FFT generic counter

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

Date

November 2015.

- Qn **std_logic_vector(n - 1 downto 0)**
- Qp **std_logic_vector(n - 1 downto 0)**

7.13.1 Detailed Description

Counter architecture declaration.

Definition at line 34 of file fftCounter.vhd.

7.13.2 Member Data Documentation

7.13.2.1 Qn std_logic_vector(n - 1 downto 0) [Signal]

Definition at line 35 of file fftCounter.vhd.

7.13.2.2 Qp std_logic_vector(n - 1 downto 0) [Signal]

Definition at line 35 of file fftCounter.vhd.

The documentation for this class was generated from the following file:

- fftCounter.vhd

7.14 fftDualRAM Entity Reference

fftDualRam entity

Entities

- fftDualRAM architecture
  fftDualRAM arqchitecture

Libraries

FFT Dual Ram

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes

Date

November 2015

- IEEE
  Standard IEEE library.

Use Clauses

FFT Dual Ram

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes

Date

November 2015

- STD_LOGIC_1164
  Standard IEEE 1164 package.
- NUMERIC_STD
  Standard IEEE numeric package.

Generics

FFT Dual Ram

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes

Date

November 2015

- n **integer**:= 8
  Standard IEEE library.
- m **integer**:= 10
  Standard IEEE library.

Ports

FFT Dual Ram

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes

Date

November 2015

- CLK **in** **STD_LOGIC**

  Standard IEEE library.

- WRA **in** **STD_LOGIC**

  Standard IEEE library.

- ADA **in** **STD_LOGIC_VECTOR**(**m** - 1 **downto** 0)

  Standard IEEE library.

- DiA **in** **STD_LOGIC_VECTOR**(**n** - 1 **downto** 0)

  Standard IEEE library.

- DoA **out** **STD_LOGIC_VECTOR**(**n** - 1 **downto** 0)

  Standard IEEE library.

- WRB **in** **STD_LOGIC**

  Standard IEEE library.

- ADB **in** **STD_LOGIC_VECTOR**(**m** - 1 **downto** 0)

Standard IEEE library.

- DiB in **STD_LOGIC_VECTOR(n - 1 downto 0)**

  Standard IEEE library.

- DoB out **STD_LOGIC_VECTOR(n - 1 downto 0)**

  Standard IEEE library.

7.14.1 Detailed Description

fftDualRam entity

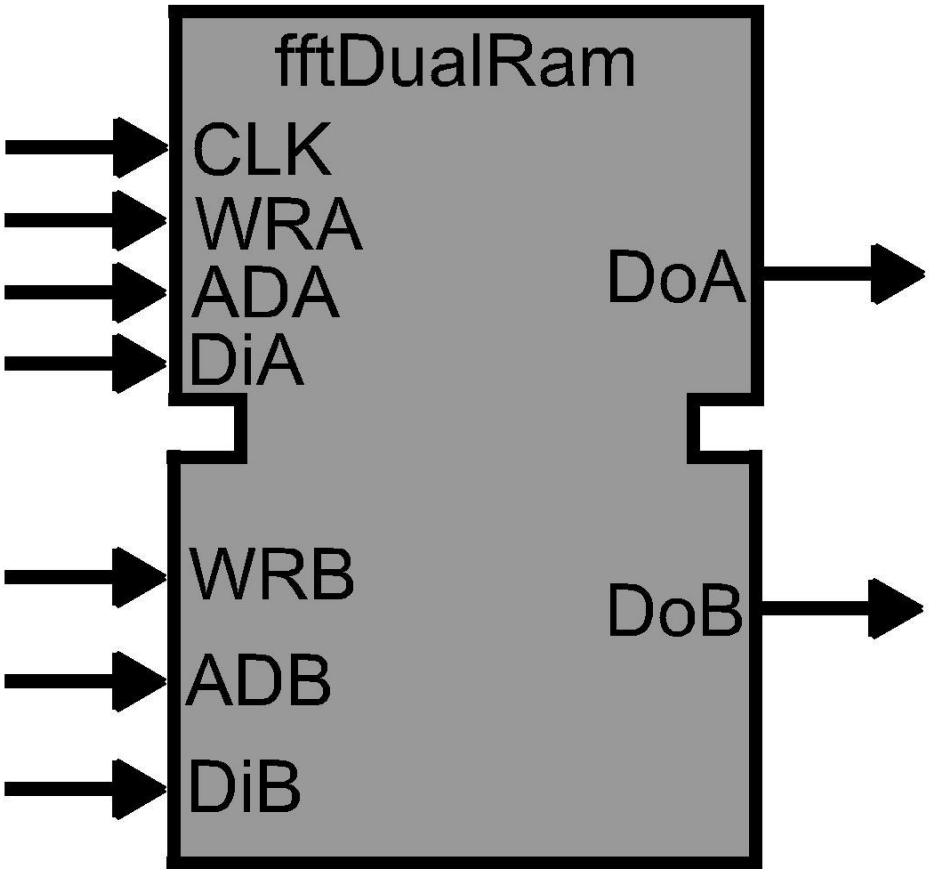


Figure 7.4: Dual RAM block

The fftDualRAM works with the clock signal and stores the two inputs "A" and "B" in two RAM each, which may be

the real part or the imaginary part, this makes it easier and faster to store and access data for the FFT process. The other signals are :

- inputs
  1. CLK : Clock
  2. WRA : Write flag for RAM "A"
  3. ADA : Addres in RAM "A"
  4. DiA : Input Data for RAM "A"
  5. WRB : Write flag for RAM "B"
  6. ADB : Addres in RAM "B"
  7. DiB : Input Data for RAM "B"
- outputs
  1. DoA : Output Data form RAM "A"
  2. DoB : Output Data form RAM "B"

Definition at line 44 of file fftDualRAM.vhd.

7.14.2 Member Data Documentation

7.14.2.1 ADA in STD_LOGIC_VECTOR(m - 1 downto 0) [Port]

Standard IEEE library.

Definition at line 50 of file fftDualRAM.vhd.

7.14.2.2 ADB in STD_LOGIC_VECTOR(m - 1 downto 0) [Port]

Standard IEEE library.

Definition at line 54 of file fftDualRAM.vhd.

7.14.2.3 CLK in STD_LOGIC [Port]

Standard IEEE library.

Definition at line 48 of file fftDualRAM.vhd.

7.14.2.4 DiA in STD_LOGIC_VECTOR(n - 1 downto 0) [Port]

Standard IEEE library.

Definition at line 51 of file fftDualRAM.vhd.

7.14.2.5 DiB in STD_LOGIC_VECTOR(n - 1 downto 0) [Port]

Standard IEEE library.

Definition at line 55 of file fftDualRAM.vhd.

7.14.2.6 DoA out STD_LOGIC_VECTOR(n - 1 downto 0) [Port]

Standard IEEE library.

Definition at line 52 of file fftDualRAM.vhd.

7.14.2.7 DoB out STD_LOGIC_VECTOR(n - 1 downto 0) [Port]

Standard IEEE library.

Definition at line 57 of file fftDualRAM.vhd.

7.14.2.8 IEEE [Library]

Standard IEEE library.

Definition at line 13 of file fftDualRAM.vhd.

7.14.2.9 m integer:= 10 [Generic]

Standard IEEE library.

Definition at line 46 of file fftDualRAM.vhd.

7.14.2.10 n integer:= 8 [Generic]

Standard IEEE library.

Definition at line 45 of file fftDualRAM.vhd.

7.14.2.11 NUMERIC_STD [Package]

Standard IEEE numeric package.

Definition at line 17 of file fftDualRAM.vhd.

7.14.2.12 STD_LOGIC_1164 [Package]

Standard IEEE 1164 package.

Definition at line 15 of file fftDualRAM.vhd.

7.14.2.13 WRA in STD_LOGIC [Port]

Standard IEEE library.

Definition at line 49 of file fftDualRAM.vhd.

7.14.2.14 WRB in STD_LOGIC [Port]

Standard IEEE library.

Definition at line 53 of file fftDualRAM.vhd.

The documentation for this class was generated from the following file:

- fftDualRAM.vhd

7.15 fftDualRAM Architecture Reference

fftDualRAM arqchitecture

FFT Dual Ram

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes

Date

November 2015

Types

- **mem_typearray(naturalrange<>)ofstd_logic_vector(n - 1 downto 0)**

  Process Data A RAM.

Processes

- PortA**(CLK)**

  Process Data A RAM.
- PortB**(CLK)**

  Process Data B RAM.

Shared Variables

- MEM **shared mem_type (0 to 2 m - 1)**

  Process Data A RAM.

7.15.1 Detailed Description

fftDualRAM arqchitecture

Definition at line 60 of file fftDualRAM.vhd.

7.15.2 Member Function Documentation

7.15.2.1 PortA(CLK) [Process]

Process Data A RAM.

Definition at line 65 of file fftDualRAM.vhd.

```
00065  PortA: process( CLK)                    begin
00066                if rising_edge( CLK) then
00067                      DoA <=  MEM(to_integer(unsigned( ADA)));
00068                    if  WRA='1' then
00069                          MEM(to_integer(unsigned( ADA))) :=  DiA;
00070                    end if;
00071                end if;
00072          end process;
00073
```

7.15.2.2 PortB(CLK) [Process]

Process Data B RAM.

Definition at line 75 of file fftDualRAM.vhd.


```
00075  PortB: process( CLK)                    begin
00076                if rising_edge( CLK) then
00077                      DoB <=  MEM(to_integer(unsigned( ADB)));
00078                    if  WRB='1' then
00079                          MEM(to_integer(unsigned( ADB))) :=  DiB;
00080                    end if;
00081                end if;
00082          end process;
00083
```

7.15.3 Member Data Documentation

7.15.3.1 MEM sharedmem_type (0 to 2 m - 1) [Shared Variable]

Process Data A RAM.

Definition at line 62 of file fftDualRAM.vhd.

7.15.3.2 mem_type array(naturalrange<>)ofstd_logic_vector(n - 1 downto 0) [Type]

Process Data A RAM.

Definition at line 61 of file fftDualRAM.vhd.

The documentation for this class was generated from the following file:

- fftDualRAM.vhd

7.16 fftFFTr2dif Entity Reference

FFT's were first discussed by Cooley and Tukey (1965), although Gauss had actually described the critical factorization step as early as 1805 (Bergland 1969, Strang 1993). A discrete Fourier transform can be computed using an FFT by means of the Danielson-Lanczos lemma if the number of points N is a power of two. If the number of points N is not a power of two, a transform can be performed on sets of points corresponding to the prime factors of N which is slightly degraded in speed. An efficient real Fourier transform algorithm or a fast Hartley transform (Bracewell 1999) gives a further increase in speed by approximately a factor of two. Base-4 and base-8 fast Fourier transforms use optimized code, and can be 20-30% faster than base-2 fast Fourier transforms. prime factorization is slow when the factors are large, but discrete Fourier transforms can be made fast for N=2, 3, 4, 5, 7, 8, 11, 13, and 16 using the Winograd transform algorithm (Press et al. 1992, pp. 412-413, Arndt).

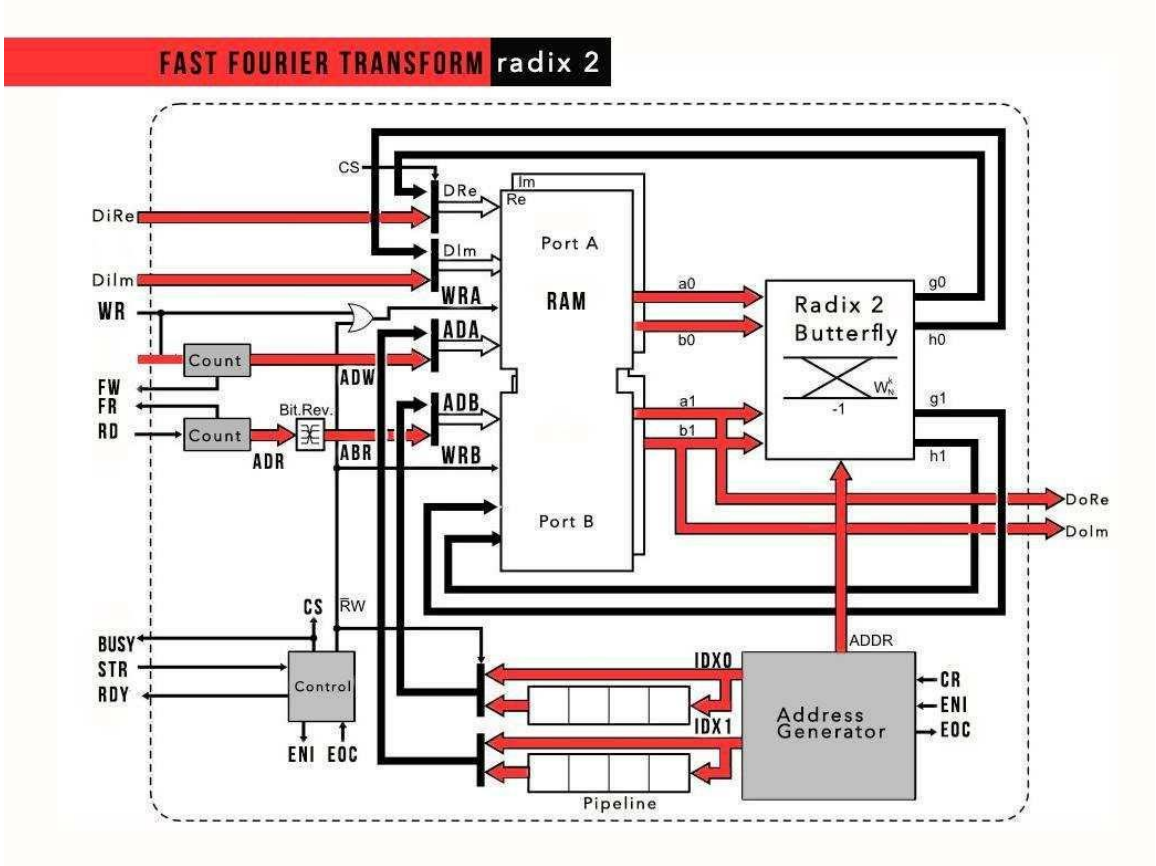


Figure 7.5: FFT Structure

- Ports

1. RST : Reset
2. CLK : Clock
3. STR : Start pulse
4. RDY : Ready pulse
5. BUSY : Busy flag
6. FR : Full read counter flag
7. FW : Full write counter flag
8. RD : Read signal
9. WR : Write signal
10. CLR : Clear read and write counters
11. DiRe : Input real data
12. DiIm : Input imaginary data
13. DoRe : Output real data
14. DoIm : Output imaginary data

Libraries

Radix 2 (DIF) FFT and IFFT core

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

Date

November 2015

- IEEE
  Standard IEEE library.

Use Clauses

Radix 2 (DIF) FFT and IFFT core

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

Date

November 2015

- STD_LOGIC_1164
  Standard IEEE 1164 package.
- NUMERIC_STD
  Standard IEEE numeric package.

Generics

Radix 2 (DIF) FFT and IFFT core

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

Date

November 2015

- np **integer:= 256**
  Number of points.
- e **integer:= 2**
  Integer format.
- f **integer:= 10**
  Float format.
- ifft **boolean:=false**
  IFFT Signal.

Ports

Radix 2 (DIF) FFT and IFFT core

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

Date

November 2015

- RST **in STD_LOGIC**
  Global core asynchronous reset(active high) -- input.
- CLK **in STD_LOGIC**
  Global core clock signal -- input.
- STR **in STD_LOGIC**
  Start pulse -- input.
- RDY **out STD_LOGIC**

Ready pulse -- output.

- BUSY out STD_LOGIC

  Busy flag --output.
- FR out STD_LOGIC

  Full read counter flag -- output.
- FW out STD_LOGIC

  Full write counter flag --output.
- RD in STD_LOGIC

  Read signal --input.
- WR in STD_LOGIC

  Write signal --input.
- CLR in STD_LOGIC

  Clear read and write counters --input.
- DiRe in STD_LOGIC_VECTOR(e + f - 1 downto 0)

  Input real data --input vector.
- DiIm in STD_LOGIC_VECTOR(e + f - 1 downto 0)

  Input complex data --input vector.
- DoRe out STD_LOGIC_VECTOR(e + f - 1 downto 0)

  Output real data --output vector.
- DoIm out STD_LOGIC_VECTOR(e + f - 1 downto 0)

7.16.1 Detailed Description

FFT's were first discussed by Cooley and Tukey (1965), although Gauss had actually described the critical factorization step as early as 1805 (Bergland 1969, Strang 1993). A discrete Fourier transform can be computed using an FFT by means of the Danielson-Lanczos lemma if the number of points N is a power of two. If the number of points N is not a power of two, a transform can be performed on sets of points corresponding to the prime factors of N which is slightly degraded in speed. An efficient real Fourier transform algorithm or a fast Hartley transform (Bracewell 1999) gives a further increase in speed by approximately a factor of two. Base-4 and base-8 fast Fourier transforms use optimized code, and can be 20-30% faster than base-2 fast Fourier transforms. prime factorization is slow when the factors are large, but discrete Fourier transforms can be made fast for N=2, 3, 4, 5, 7, 8, 11, 13, and 16 using the Winograd transform algorithm (Press et al. 1992, pp. 412-413, Arndt).

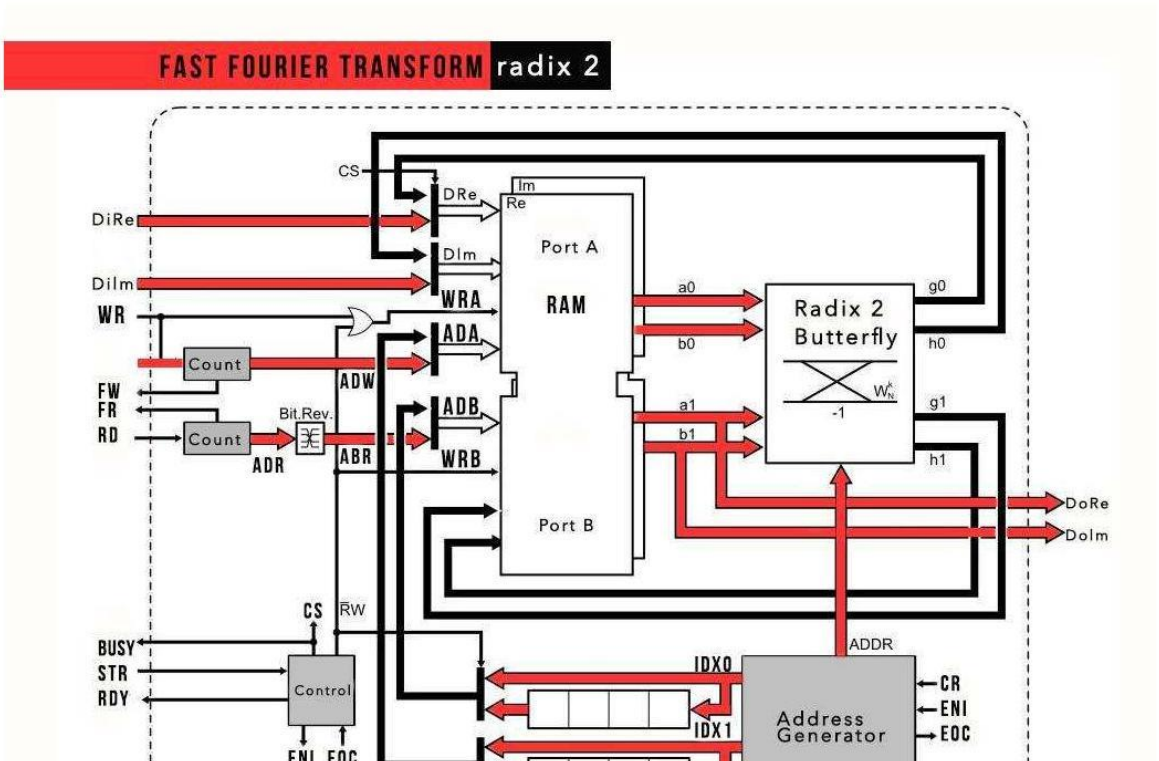


Figure 7.6: FFT Structure

- Ports
    1. RST : Reset
    2. CLK : Clock
    3. STR : Start pulse
    4. RDY : Ready pulse
    5. BUSY : Busy flag
    6. FR : Full read counter flag
    7. FW : Full write counter flag
    8. RD : Read signal
    9. WR : Write signal
    10. CLR : Clear read and write counters
    11. DiRe : Input real data
    12. DiIm : Input imaginary data
    13. DoRe : Output real data
    14. DoIm : Output imaginary data

7.16 fftFFTr2dif Entity Reference

Module to perform the Fast Fourier Transform (FFT) and its inverse,it configurable via generic list to be used with a wide range of points and adjustable numerical format, the input and output data are in the same numerical format. The FFT is properly normalized, no further division is needed. It uses unbiased rounding for the calculations. This Core allowed to work with the following points :4,16,32,64,128,256,512,1024,2048,4096,8192,16384. The Core was verified using Active-HDL simulator with a test bech with two signals as inputs.

Attention

Configuration

np: number of points (4,16,32,64,128,256,512,1024,2048,4096,8192,16384,...)

The following table shows the synthesis results of the core in the FPGA used. In this case its a Xilinx Spartan 3E-1600.

Logic Utilization	Used	Available	Utilization
Number of Slices	1359	14752	9%
Number of Slices Flip Flops	71	29504	0%
Number of 4 inputs LUTs	5625	29504	19%
Number of bonded IOBs	52	250	20%
Number of MULT18X18SIOs	4	36	11%
Number of GLCKs	1	24	4%

Table 1: Synthesis Results

The above results where obtained using the Xilinx ISE Webpack 14.7

Definition at line 82 of file fftFFTr2dif.vhd.

7.16.2 Member Data Documentation

7.16.2.1 BUSY out STD_LOGIC [Port]

Busy flag --output.

Definition at line 93 of file fftFFTr2dif.vhd.

7.16.2.2 CLK in STD_LOGIC [Port]

Global core clock signal -- input.

Definition at line 90 of file fftFFTr2dif.vhd.

7.16.2.3 CLR in STD_LOGIC [Port]

Clear read and write counters --input.

Definition at line 98 of file fftFFTr2dif.vhd.

7.16.2.4 DiIm in STD_LOGIC_VECTOR(e +f - 1 downto 0) [Port]

Input complex data --input vector.

Definition at line 100 of file fftFFTr2dif.vhd.

7.16.2.5 DiRe in STD_LOGIC_VECTOR(e +f - 1 downto 0) [Port]

Input real data --input vector.

Definition at line 99 of file fftFFTr2dif.vhd.

7.16.2.6 DoIm out STD_LOGIC_VECTOR(e +f - 1 downto 0) [Port]

Output complex data --output vector

Definition at line 104 of file fftFFTr2dif.vhd.

7.16.2.7 DoRe out STD_LOGIC_VECTOR(e +f - 1 downto 0) [Port]

Output real data --output vector.

Definition at line 101 of file fftFFTr2dif.vhd.

7.16.2.8 e integer:= 2 [Generic]

Integer format.

Definition at line 84 of file fftFFTr2dif.vhd.

7.16.2.9 f integer:= 10 [Generic]

Float format.

Definition at line 85 of file fftFFTr2dif.vhd.

7.16.2.10 FR out STD_LOGIC [Port]

Full read counter flag -- output.

Definition at line 94 of file fftFFTr2dif.vhd.

7.16.2.11 FW out STD_LOGIC [Port]

Full write counter flag --output.

Definition at line 95 of file fftFFTr2dif.vhd.

7.16.2.12 IEEE [Library]

Standard IEEE library.

Definition at line 16 of file fftFFTr2dif.vhd.

7.16.2.13 ifft boolean:=false [Generic]

IFFT Signal.

Definition at line 86 of file fftFFTr2dif.vhd.

7.16.2.14 np integer:= 256 [Generic]

Number of points.

Definition at line 83 of file fftFFTr2dif.vhd.

7.16.2.15 NUMERIC_STD [Package]

Standard IEEE numeric package.

Definition at line 20 of file fftFFTr2dif.vhd.

7.16.2.16 RD in STD_LOGIC [Port]

Read signal --input.

Definition at line 96 of file fftFFTr2dif.vhd.

7.16.2.17 RDY out STD_LOGIC [Port]

Ready pulse -- output.

Definition at line 92 of file fftFFTr2dif.vhd.

7.16.2.18 RST in STD_LOGIC [Port]

Global core asynchronous reset(active high) -- input.

Definition at line 89 of file fftFFTr2dif.vhd.

7.16.2.19 STD_LOGIC_1164 [Package]

Standard IEEE 1164 package.

Definition at line 18 of file fftFFTr2dif.vhd.

7.16.2.20 STR in STD_LOGIC [Port]

Start pulse -- input.

Definition at line 91 of file fftFFTr2dif.vhd.

7.16.2.21 WR in STD_LOGIC [Port]

Write signal --input.

Definition at line 97 of file fftFFTr2dif.vhd.

The documentation for this class was generated from the following file:

- fftFFTr2dif.vhd

7.17 fftLimiter Architecture Reference

Limiter arcchitecture declaration.

FFT limiter and unbiased rounded

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

Date

November 2015.

Constants

- UL **std_logic_vector(er - el downto 0):=(others=>' 0 ')**
- LL **std_logic_vector(er - el downto 0):=(others=>' 1 ')**

Processes

- Rounding**(FRC , FRR , RAW)**
- Limiter**(RND , ENT)**

Signals

- RND **std_logic_vector(er + fr - 1 downto fr - fl)**
- ENT **std_logic_vector(er - el downto 0)**
- LM **std_logic_vector(el + fl - 1 downto 0)**
- FRR **std_logic_vector(fr - fl - 1 downto 0)**
- FRC **std_logic_vector(fr - fl - 1 downto 0)**

7.17.1 Detailed Description

Limiter arcchitecture declaration.

Definition at line 46 of file fftLimiter.vhd.

7.17.2 Member Function Documentation

7.17.2.1 Limiter(RND , ENT) [Process]

Definition at line 90 of file fftLimiter.vhd.

```
    Limiter: process( RND, ENT)00090                begin
00091                   if ( UL /=  ENT) AND ( RND( er+ fr-1) = '0') then
00092                       LM( el+ fl-1) <= '0';
00093                   LM( el+ fl-2 downto 0) <= (others=>'1');
00094               elsif ( LL /=  ENT) AND ( RND( er+ fr-1) = '1') then
00095                       LM( el+ fl-1) <= '1';
```



```
00096                   LM( el+ fl-2 downto 0) <= (others=>'0');
00097               else
00098                   LM <=  RND( fr+ el-1 downto  fr- fl);
00099               end if;
00100           end process;
00101
```

7.17.2.2 Rounding(FRC , FRR , RAW) [Process]

Definition at line 81 of file fftLimiter.vhd.

```
end generate;00081
```

7.17.3 Member Data Documentation

7.17.3.1 ENT std_logic_vector(er -el downto 0) [Signal]

Definition at line 48 of file fftLimiter.vhd.

7.17.3.2 FRC std_logic_vector(fr -fl - 1 downto 0) [Signal]

Definition at line 81 of file fftLimiter.vhd.

7.17.3.3 FRR std_logic_vector(fr -fl - 1 downto 0) [Signal]

Definition at line 81 of file fftLimiter.vhd.

7.17.3.4 LL std_logic_vector(er -el downto 0):=(others=>' 1 ') [Constant]

Definition at line 51 of file fftLimiter.vhd.

7.17.3.5 LM std_logic_vector(el +fl - 1 downto 0) [Signal]

Definition at line 49 of file fftLimiter.vhd.

7.17.3.6 RND std_logic_vector(er +fr - 1 downtofr -fl) [Signal]

Definition at line 47 of file fftLimiter.vhd.

7.17.3.7 UL std_logic_vector(er -el downto 0):=(others=>' 0 ') [Constant]

Definition at line 50 of file fftLimiter.vhd.

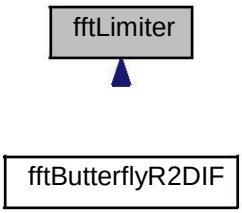
The documentation for this class was generated from the following file:

- fftLimiter.vhd

7.18 fftLimiter Entity Reference

Numerical format adjustment using unbiased rounding (Banker's method) and saturation limits to avoid format overflow.Limiter entity declaration.

Inheritance diagram for fftLimiter:



Entities

- fftLimiter architecture

  Limiter arcchitecture declaration.

Libraries

FFT limiter and unbiased rounded

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

Date

November 2015.

- IEEE

  Standard IEEE library.

Use Clauses

FFT limiter and unbiased rounded

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

---

Date

November 2015.

- STD_LOGIC_1164
  Standard IEEE 1164 package.
- NUMERIC_STD
  Standard IEEE Numeric package.

Generics

FFT limiter and unbiased rounded

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

Date

November 2015.

- er **integer**:= 5
  Standard IEEE library.
- fr **integer**:= 5
  Standard IEEE library.
- el **integer**:= 3
  Standard IEEE library.
- fl **integer**:= 2
  Standard IEEE library.

Ports

FFT limiter and unbiased rounded

Author

Luis Morales Velazquez
Jean Karlo Gomez Reyes Company: Universidad Autonoma de Queretaro

Date

November 2015.

- CLK **in** **STD_LOGIC**
  Standard IEEE library.
- RAW **in** **STD_LOGIC_VECTOR**(**er** + **fr** - 1 **downto** 0)

Standard IEEE library.

- LIM out STD_LOGIC_VECTOR(el + fl - 1 downto 0)

  Standard IEEE library.

7.18.1 Detailed Description

Numerical format adjustment using unbiased rounding (Banker's method) and saturation limits to avoid format overflow.Limiter entity declaration.

This unit performs the format adjustment for the input signal RAW into the output signal LIM.

When the number in the input is beyond the range of the output format the output is saturated at the maximum or minimum values in the given format.

Considerations: There are two restrictions for the input parameters: er + fr > el + fl and fr >= fl.

Parameters

in	er.fr	Input format
out	el.fl	Output format

Definition at line 34 of file fftLimiter.vhd.

7.18.2 Member Data Documentation

7.18.2.1 CLK in STD_LOGIC [Port]

Standard IEEE library.

Definition at line 40 of file fftLimiter.vhd.

7.18.2.2 el integer:= 3 [Generic]

Standard IEEE library.

Definition at line 37 of file fftLimiter.vhd.

7.18.2.3 er integer:= 5 [Generic]

Standard IEEE library.

Definition at line 35 of file fftLimiter.vhd.

7.18.2.4 fl integer:= 2 [Generic]

Standard IEEE library.

Definition at line 38 of file fftLimiter.vhd.

7.18.2.5 fr integer:= 5 [Generic]

Standard IEEE library.

Definition at line 36 of file fftLimiter.vhd.

7.18.2.6 IEEE [Library]

Standard IEEE library.

Definition at line 9 of file fftLimiter.vhd.

7.18.2.7 LIM out STD_LOGIC_VECTOR(el +fl - 1 downto 0) [Port]

Standard IEEE library.

Definition at line 42 of file fftLimiter.vhd.

7.18.2.8 NUMERIC_STD [Package]

Standard IEEE Numeric package.

Definition at line 13 of file fftLimiter.vhd.

7.18.2.9 RAW in STD_LOGIC_VECTOR(er +fr - 1 downto 0) [Port]

Standard IEEE library.

Definition at line 41 of file fftLimiter.vhd.

7.18.2.10 STD_LOGIC_1164 [Package]

Standard IEEE 1164 package.

Definition at line 11 of file fftLimiter.vhd.

The documentation for this class was generated from the following file:

- fftLimiter.vhd

Index