UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

Ingeniería Eléctrica – Instrumentación

Desarrollo y puesta en marcha de un sistema embebido en FPGA para el procesamiento de datos en 2D con potenciales aplicaciones en óptica adaptativa.

TESIS QUE PARA OPTAR POR EL GRADO DE: MAESTRO EN INGENIERÍA

PRESENTA:
William Hidber Cruz

TUTOR PRINCIPAL
SALVADOR CUEVAS CARDONA, INSTITUTO DE ASTRONOMÍA UNAM

MÉXICO, D. F. Febrero 2013

JURADO ASIGNADO:

PRESIDENTE: DR. QURESHI NASER

SECRETARIO: DR. BRUCE DAVIDSON NEIL CHARLES

VOCAL: DR. CUEVAS CARDONA SALVADOR

1 ER. SUPLENTE: DRA. ROSETE AGUILAR MARTHA

2 D O. SUPLENTE: DR. SANDOVAL ROMERO GABRIEL EDUARDO

LUGAR O LUGARES DONDE SE REALIZÓ LA TESIS: INSTITUTO DE ASTRONOMÍA

TUTOR DE TESIS:

DR. SALVADOR CUEVAS CARDONA

FIRMA

Dedicatoria:

A mis padres, hermano y amigos de la maestría. A los profesores que me apoyaron en la realización de la tesis.

Agradecimientos:

Al CONACYT por el apoyo económico en la realización de la maestría.

Índice i
Introducción iii

Capítulo 1.- Óptica Adaptativa

1.1.- Introducción. 1
1.2 Sistemas de óptica adaptativa. 2
1.5.1.- Sensores de Frente de onda. 3
1.5.1.2.- Sensor de curvatura. 4
1.5.1.4.- Reconstrucción del frente de onda. 6
1.5.1.4.1.- Método basado en Zernike. 6
1.5.1.4.2.- Método basado en el Laplaciano. 7
1.5.2.- Corrección de frente de onda. 9

Capítulo 2._ Análisis de Fourier

2.1.- Introducción. 11
2.3.- La transformada de Fourier en Tiempo Discreto. 11
2.4.- La Transformada Discreta de Fourier en 2D. 12
2.5.- FFT. 15
2.5.2.- Radix-2 en decimación en el tiempo. 15
2.5.3.- Radix-2 en decimación en frecuencia. 16

Capítulo 3._ Dispositivos Lógicos Programables

3.1.- Introducción. 19
3.2.- FPGA. 19
3.2.1.- Arquitectura. 20
3.2.1.1.- Bloques lógicos configurables. 21
3.2.1.2.- Bloques de entrada/salida (IOB, Input/Output Blocks). 21
3.2.1.3.- Interconexiones programables. 22
3.2.1.4.- Bloques de propósito especial. 23
3.4.2.- Aplicaciones. 24

Capítulo 4._ Implementación de sistemas en FPGAs.

4.1.- Introducción. 27
4.2.- Metodologías de modelado. 29
4.3.- Diagramas ASM. 30
4.3.1 Bloques ASM. 32
4.4._ VHDL. 32
4.4.1.- Ventajas del uso de VHDL. 34
4.4.2.- Niveles de abstracción en VHDL. 34
4.5.- Herramientas CAD para el diseño de sistemas en FPGA. 35
4.5.1.- Síntesis. 37
4.5.2.- Simulación funcional. 38
4.5.3.- Diseño físico. 38
4.5.4.- Simulación de tiempo. 39

Capítulo 5._ Implementación del sistema

5.1.- Introducción. 41

5.2.- Modelado del sistema. 42
5.2.1.- Condición para que el sistema pueda ser empleado en sistemas de Óptica Adaptativa. 46
5.3.- Módulo de control de la SRAM. 46
5.4- UART RS-232. 51
5.4.1.- Módulo ctrl UART RX. 52
5.4.1.1.- Submódulo UART RX. 55
5.4.1.2.- Submódulo de conversión a punto flotante. 57
5.4.1.3.- Submódulo de direccionamiento de memoria. 59
5.4.2.- Módulo ctrl UART TX. 61
5.4.2.1.- Submódulo UART TX. 64
5.4.2.2.- FIFO. 65
5.4.2.3.- Submódulo de direccionamiento de memoria. 66
5.5.- Módulo FFT (core FFT). 66
5.5.1.- Implementación del core FFT. 68
5.6.- Módulo de control del core FFT. 71
5.6.1.- Descripción de los submódulos que integran el módulo de control de la FFT. 72
5.6.2.- Integración de los submódulos. 76
5.6.3.- Análisis de tiempo del módulo de control de la FFT. 78
5.7.- Integración del sistema (Módulo de control principal del sistema). 80
5.7.1.- Análisis de tiempo y relación con un sistema de óptica adaptativa. 83

Capítulo 6._ Pruebas y resultados

6.1.- Introducción. 85
6.2.- Pruebas realizadas al sistema. 85
6.2.1.- Prueba de transmisión de datos PC – FPGA. 87
6.2.1.1.- Implementación física. 90
6.2.2.- Prueba de recepción de datos PC – FPGA. 91
6.2.2.1.- Implementación física. 94
6.2.3.- Prueba de procesamiento de datos. 95
6.2.3.1.- Implementación física. 98
6.3.- Resultados. 99

Conclusiones. 109
Proyecciones a futuro. 110

Anexos

A.- Hoja de datos de la FPGA Spartan 3E. 114
B.- Diagramas de comparación de tiempo en el procesado de la FFT en la FPGA. 118
C._ Uso de los bloques DCM de la FPGA Spartan 3E. 138
D._ Cartas ASM de los módulos implementados. 140
E.- Comparación del tiempo de procesamiento del FPGA, CPU y GPU. 153

Referencias 155
Bibliografía 156

Introducción

Un frente de onda producido por una fuente puntual a millones de kilómetros, al llegar a la tierra, se ve perturbado por la turbulencia atmosférica. A causa de éste problema, las imágenes que se observan por un telescopio terrestre no son nítidas, ya que las aberraciones que se introdujeron al frente de onda producen que la imagen observada presente distorsiones. Este problema se soluciona por medio de la óptica adaptativa, la cual tiene como fin compensar en tiempo real el frente de onda incidente aberrado por la atmósfera. La corrección del frente de onda se realiza por medio de un espejo deformable, el cual está controlado por un *sistema de control*. Este sistema recibe señales de un sensor de frente de onda y decide cómo manejar esos datos. Una etapa importante es la de recuperación del frente de onda medido, recuperación hecha a partir de los datos del sensor de frente de onda. Los datos producidos por la recuperación del frente de onda son manejados por el sistema de control. Existen varios métodos para realizar la recuperación del frente de onda, uno de éstos se realiza resolviendo la ecuación diferencial producida por los datos de un sensor conocido como "sensor de curvatura", utilizando como recurso la FFT. Debido al alto consumo de recursos que se necesitan para resolver la FFT y además de que el cálculo se debe realizar en tiempo real, es necesario implementar un dispositivo embebido que lleve a cabo esta función.

Por lo tanto el objetivo de la presente tesis es: Planear, diseñar, implementar y validar un subsistema que esté destinado específicamente a realizar el cálculo de la FFT de una imagen (por medio de FPGAs), el cual tiene como fin ser parte de un sistema más complejo para realizar la recuperación de un frente de onda, teniendo aplicaciones en óptica adaptativa. Este subsistema, además, debe contar con una interfaz de comunicación para la transmisión de los datos a procesar y los ya procesados, necesaria si los datos se quieren transmitir a otros sistemas de procesamiento de datos.

Un sistema en tiempo real es aquel que interactúa con el mundo exterior donde el tiempo es un factor importante, la reacción del sistema a eventos externos debe ocurrir durante su evolución, donde el correcto funcionamiento del sistema no sólo depende del resultado lógico que devuelve, sino también depende del tiempo en que se produce ese resultado. Existen muchos dispositivos que son empleados para aplicaciones en tiempo real. Sin embargo, para aplicaciones de procesamiento de señales hay dos que se emplean con mayor frecuencia: los DSPs (del inglés *Digital Signal Processor*) y los FPGAs (del inglés *Field Programmable Gate Array*). Éstos últimos han ido evolucionando muy rápidamente, tanto que un FPGA puede sustituir a varios DSPs. En el área de

Astronomía empieza a ser común el uso de los FPGAs para implementar sistemas embebidos en tiempo real. Se han propuesto sistemas que son capaces de realizar el cálculo de la FFT (del inglés *Fast Fourier Transform*) para posibles aplicaciones generales. Tal es el caso de Tomasz S. Czajkowski et al [1], quienes en una FPGA implementan diferentes algoritmos para el cálculo de la FFT, con el fin de hacer una comparación entre ellos, llegando a la conclusión de que los multiplicadores incluidos en el FPGA son los que limitan la precisión del cálculo. La aplicación directa que ellos proponen es en Astrofísica, sin precisar en qué área será empleado. En el caso de la FFT en dos dimensiones, una aplicación directa se encuentra en el procesamiento de imágenes, pero no necesariamente con aplicaciones en el área de Astronomía. Se han desarrollado más aplicaciones en el área Biomédica. I. S. Uzun et al [2], proponen emplear los FPGAs en el cálculo de la FFT, ya que la FFT es un algoritmo que necesita un alto empleo de recursos computacionales, demostrando que es mejor emplear los FPGAs sobre los DSPs para el procesamiento de imágenes. Carlos Lujan Ramirez et al [3] realizan la misma comparación de dispositivos, FPGAs vs DSPs, y llegan a la misma conclusión: un FPGA es mejor que un DPS para el procesamiento digital de imágenes. A parte de que el algoritmo de la FFT en dos dimensiones emplea muchos recursos computacionales, también se requiere de más tiempo para efectuar el algoritmo, por lo que Jung Sub Kim et al [4] proponen un diseño en paralelo para efectuar el algoritmo de la FFT en el FPGA y de esta forma realizarlo en menor tiempo. Las anteriores propuestas únicamente evalúan la implementación de la FFT en una FPGA, para aplicaciones más específicas para utilizar el algoritmo ya implementado en la FPGA podemos encontrar la propuesta de J. M. Rodríguez-Ramos et al [5], quienes utilizan la implementación de la FFT en dos dimensiones en una FPGA para recuperar un frente de onda a partir de los datos que entrega la cámara CADAFIS. Éstos mismos investigadores evalúan el desempeño que presenta el sistema implementado sobre un FPGA comparado con el sistema implementado sobre una GPU (del inglés *Graphics Processing Unit*), argumentando que el mejor desempeño se presenta utilizando FPGAs. Hay pocas aplicaciones en el uso de los FPGAs en Astronomía (como se comentó antes, apenas se empiezan a utilizar éstos dispositivos en esta área). Sin embargo hay que tomar muy en cuenta a los FPGAs cuando se desee implementar sistemas en tiempo real, ya que los autores antes mencionados ponen énfasis en que los FPGAs son muy útiles en aplicaciones donde el tiempo de respuesta es un factor importante.

En las siguientes hojas el trabajo está dividido en 6 capítulos, cuyo contenido es el siguiente:

Capítulo 1. Se da una breve explicación sobre lo que son los sistemas de óptica adaptativa, con el fin de especificar el uso y la ubicación que se la dará al sistema implementado.

Capítulo 2. El objetivo de la tesis consiste en desarrollar un sistema que permita efectuar la FFT. En este capítulo se explica en que consiste la Transformada de Fourier discreta en 1D y 2D. También se describe el algoritmo para obtener la FFT en 1D.

Capítulo 3. En este capítulo se describe de forma general la arquitectura de un FPGA,.

Capítulo 4. Se detalla la metodología que se sigue para implementar un sistema en un FPGA, así como también los lenguajes de programación que permiten realizar el modelado de un sistema para que pueda ser implementado en éstos dispositivos. Es importante mencionar esta metodología ya que fue la que se siguió para el desarrollo del sistema implementado.

Capítulo 5. Este capítulo constituye la parte central de la tesis, ya que aquí se explica el desarrollo de la implementación del sistema propuesto.

Capítulo 6. Se describen las pruebas realizadas al sistema implementado (con el fin de comprobar que se cumplió el objetivo planteado) y los resultados obtenidos.

Para finalizar el trabajo se detallan las conclusiones a las que se llegaron una vez finalizado el desarrollo del sistema además de las recomendaciones para dar continuación al presente trabajo.

Capítulo 1

Óptica Adaptativa

1.1 INTRODUCCIÓN

En la época actual la Astronomía está dominada por los grandes telescopios terrestres, esto se debe al menor coste que representa su construcción comparado con los telescopios espaciales. Sin embargo, debido a las propiedades de absorción, dispersión y turbulencia que presenta la atmósfera terrestre se dificulta realizar mediciones precisas de los objetos observados por estos telescopios.

La luz captada por los telescopios tiene su origen en una fuente puntual, la cual es una ubicación sencilla donde los rayos se propagan igualmente en todas direcciones (fuente isotrópica). El frente de onda generado por una fuente puntual simplemente es una esfera con radio R y su centro está ubicado en el punto de origen de las ondas. En el espacio libre y a una distancia suficiente de la fuente, los rayos dentro de un área pequeña de un frente de onda esférica, son casi paralelos, por lo tanto, entre más lejos este de la fuente, más se parece la propagación de ondas a un frente de onda plano. Sin embargo, apenas un frente de onda atraviesa la atmosfera terrestre, sufre distorsiones, producto de que el índice de refracción del aire no es constante, ya que depende de las condiciones de temperatura, presión y densidad, las cuales varían dependiendo de la altitud terrestre. Además de que pude haber una mezcla de masas de aire con diferentes características físicas, provocando cambios bruscos en el índice de refracción. Como consecuencia de esto, las imágenes captadas por los telescopios se vuelven borrosas y se introducen ciertas aberraciones debido al constante cambio en el índice de refracción.

Desde la última mitad del siglo XX se han realizado grandes investigaciones en técnicas que ayuden a compensar los efectos indeseados producidos por la atmósfera, una de estas técnicas es la óptica adaptativa.

La óptica adaptativa es utilizada para mejorar la capacidad de los sistemas ópticos mediante la compensación activa de las aberraciones. Estas aberraciones reducen la intensidad y claridad de las imágenes tomadas por los telescopios. La esencia de la óptica adaptativa reside en su capacidad

para medir, y en última instancia, la de corregir la dirección de los rayos de luz perturbados por la turbulencia atmosférica [6].

1.2 SISTEMAS DE ÓPTICA ADAPTATIVA

Antes de describir en que consiste un sistema de óptica adaptativa, es importante definir con exactitud el tipo de sistemas al que nos vamos a referir, porque en la literatura se suelen confundir los conceptos de óptica activa y óptica adaptativa. Se toma el concepto más difundido entre la comunidad astronómica. Ambos tipos de sistemas corrigen aberraciones que cambian con el tiempo, como consecuencia de esto, es evidente que tales sistemas no pueden ser estáticos, como por ejemplo, una lente divergente que corrige aberraciones fijas de un ojo miope, sino que han de tener elementos que varíen con el tiempo, al igual que las aberraciones que corrigen. Los sistemas de óptica activa corrigen distorsiones que varían con frecuencia inferior a 2 Hz. Los sistemas de óptica adaptativa compensan distorsiones cuya frecuencia es superior a 2 Hz [9].

En las últimas décadas los sistemas de óptica adaptativa han estado en constante evolución. En la Figura 1.1 se ilustra la configuración más común de los sistemas de óptica adaptativa. Este sistema contiene tres elementos principales, los cuales son: el sensor de frente de onda, un espejo de corrección (para la corrección de las aberraciones de bajo orden, además se puede presentar un espejo complementario para la corrección de aberraciones de órdenes superiores) y el sistema de control. Otros subsistemas, ajenos al sistema de óptica adaptativa, son: el telescopio y el sensor CCD (del inglés *Charge Coupled Device*). Los frentes de onda aberrados inciden en el telescopio, el cual envía las señales al sistema de óptica adaptativa, para que posteriormente los datos obtenidos se manden al CCD.

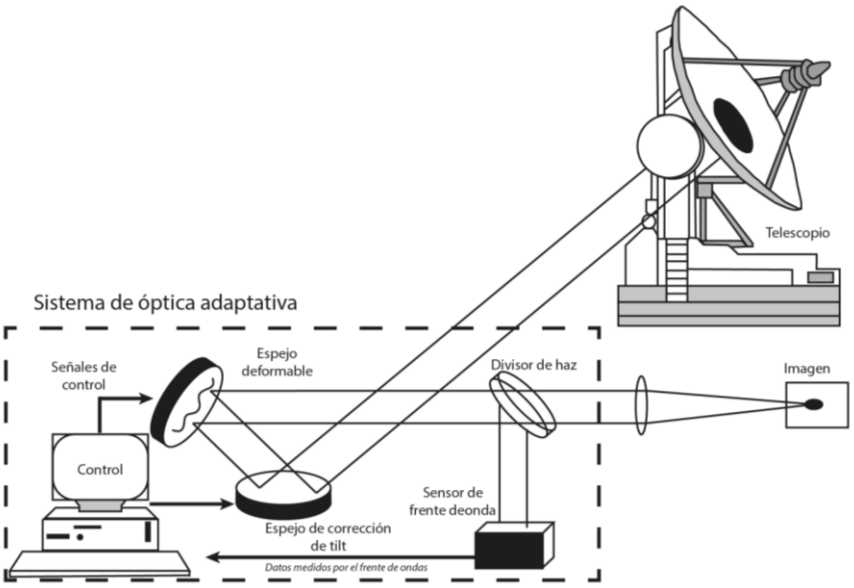


Figura 1.1._ Sistema de óptica adaptativa.

El sensor de frente de onda es ofrecido como una caja negra que acepta un haz de luz y da como salida señales eléctricas relacionadas con la fase del rayo de entrada. El espejo deformable es un espejo con una serie de actuadores en su parte de atrás que deforman el espejo para que tome una determinada forma. El sistema de control es un procesador de señales que traduce las señales provenientes del sensor de frente de onda a comandos de control para el espejo deformable, con el fin de compensar el rayo incidente. Dentro del sistema de control hay un proceso que a menudo se llama "reconstrucción", donde las señales son usadas para reconstruir la fase proveniente de las señales del sensor de frente de onda.

La forma en que un sistema de óptica adaptativa funciona es simple (la forma en la que se implementa es donde reside la mayor parte del reto en los sistemas de óptica adaptativa). Para un sistema de adquisición de imágenes, como puede ser un telescopio, primero debemos medir la fase del haz de luz entrante, posteriormente se determina que tan distorsionado está, con el fin de calcular una compensación, por último se aplica esta compensación al espejo deformable. El proceso anterior se repite según vaya cambiando la distorsión en la fase de los rayos entrantes.

El sistema de control de óptica adaptativa plasmará en el espejo deformable la forma de la aberración, excepto que lo hará con el signo contrario. Debido a que el sistema busca una perturbación, realiza una corrección y luego mira otra vez a la corrección realizada para aplicar una nueva corrección, el sistema es conocido como de lazo cerrado, por lo que un sistema de óptica adaptativa siempre opera en lazo cerrado.

1.2.1 SENSORES DE FRENTE DE ONDA

Estos dispositivos permiten medir con la suficiente rapidez los frentes de onda incidentes, con el fin de aplicar correcciones en tiempo real.

Cada sensor de frente de onda consiste en los siguientes componentes principales:

- **Dispositivo óptico:** El cual transforma las aberraciones en variaciones de intensidad de luz (a diferencia de las ondas de radio, la fase de las ondas ópticas no se puede medir directamente).
- **Detector:** Transforma la intensidad de la luz en una señal eléctrica. Esta señal tiene un ruido intrínseco debido a la naturaleza de los fotones de la luz, pero puede también contener una contribución de ruido del detector.
- **Reconstructor:** Es necesario para convertir las señales en las aberraciones de fase. El cálculo debe ser lo suficientemente rápido, lo que significa, prácticamente, que solo reconstructores lineales son útiles. Un reconstructor lineal típicamente realiza la multiplicación de una matriz.

En la realidad no es posible medir directamente la fase del frente de onda en las longitudes de onda ópticas en las que se trabaja, ya que no existe un detector que responda a las frecuencias temporales implicadas. Generalmente se utilizan métodos indirectos para trasladar la información relacionada con la fase de las ondas en señales de intensidad que puedan ser procesadas. Los sensores más utilizados en óptica adaptativa son los siguientes,

a) Sensor Shack-Hartmann.
b) Sensor de curvatura.
c) Interferómetro de desplazamiento lateral.

De estos tres sensores únicamente se describirá el sensor de curvatura, ya que es necesario entender su funcionamiento para fundamentar el desarrollo de la tesis. Para mayor información de los otros sensores se puede consultar los libros de (Tyson R. K., 2000) y (H.I. Campbell, 2006).

1.2.1.1 SENSOR DE CURVATURA

Este sensor fue desarrollado por Roddier para hacer mediciones de la curvatura (la señal que mide es proporcional a la curvatura) del frente de onda. La Figura 1.2 ilustra la idea básica del funcionamiento del sensor de curvatura. El telescopio de longitud focal *f*, crea la imagen en el plano focal. Este sensor consiste en dos detectores de imagen ubicados fuera de foco, uno ubicado entre la pupila del telescopio y el foco, y el otro más alejado que el plano focal. El primer detector registra la distribución de irradiancia[1] en el plano P_1 a una distancia l antes del plano focal y el segundo registra la distribución de irradiancia en un plano P_2 a la misma distancia l después del plano focal. La curvatura de un frente de onda produce un exceso de iluminación en un plano y una pérdida de iluminación en el otro plano. El sensor de curvatura obtiene medidas de los dos planos desenfocados, y devuelve una señal que se obtiene restando las intensidades en puntos simétricos de cada plano.

[1] Magnitud utilizada para describir la potencia incidente por unidad de superficie de todo tipo de radiación electromagnética.

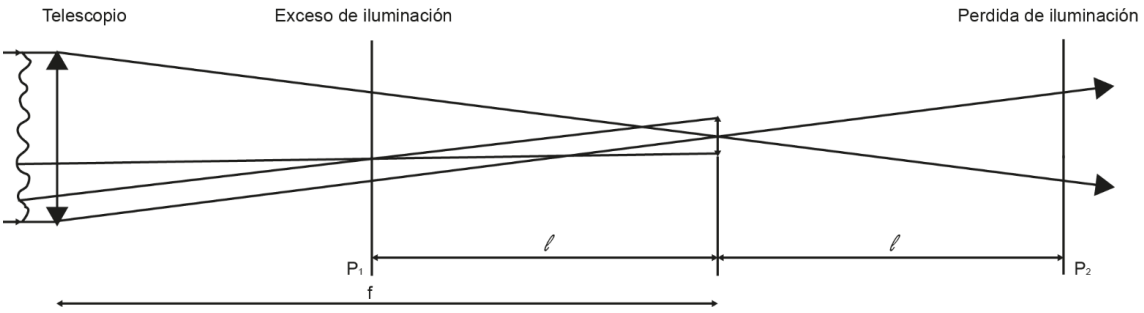


Figura 1.2._ Principio de funcionamiento del sensor de curvatura,

Para un análisis cuantitativo del funcionamiento de este sensor se parte de la ecuación del transporte de la irradiancia.

$$\frac{\partial I}{\partial z} = -\frac{\lambda}{2\pi}\left(\vec{\nabla} I \cdot \vec{\nabla}\phi + I\vec{\nabla}^2\phi\right) \tag{1.1}$$

Esta ecuación determina la propagación de la iluminación $I(x,y)$ de un plano a otro, en función de la fase $\phi(x,y)$, la cual es válida para la óptica geométrica. A partir de esta ecuación se llega a la señal medida por el sensor, la cual es la diferencia normalizada entre la iluminación $I_1(\vec{r})$ e $I_2(\vec{r})$ en los planos P_1 y P_2, y está dada por,

$$s(r) = \frac{I_1(\vec{r}) - I_2(-\vec{r})}{I_1(\vec{r}) + I_2(-\vec{r})} = \frac{\lambda f(f-\ell)}{2\pi\ell}\left[\frac{\partial\varphi}{\partial n}\left(\frac{\vec{fr}}{\ell}\right)\delta_c - \nabla^2\delta\left(\frac{\vec{fr}}{\ell}\right)\right] \tag{1.2}$$

El valor $\frac{\partial\varphi}{\partial n}$ es la primera derivada del frente de onda en los bordes, δ_c es un impulso de distribución lineal alrededor del borde de la pupila y ∇^2 es el Laplaciano del frente de ondas incidente. En la Figura 1.3 se ilustra un ejemplo de las intensidades obtenidas en un sensor de curvatura, en el plano P_1 y P_2 y la diferencia de estas dos imágenes.

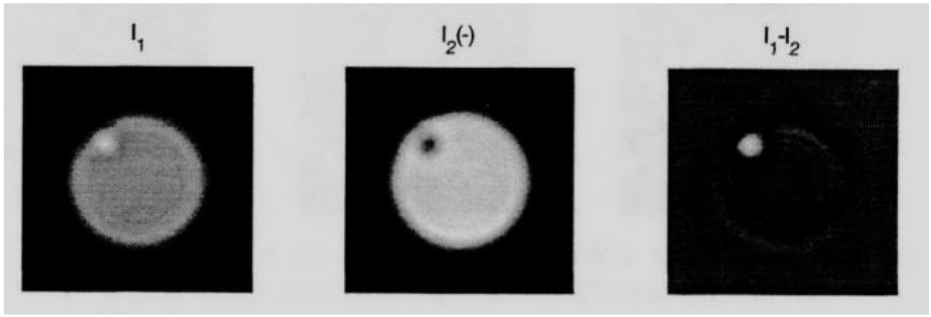


Figura 1.3._ Datos obtenidos en un sensor de curvatura.

1.2.1.2 RECONSTRUCCIÓN DEL FRENTE DE ONDA

Los sensores mencionados anteriormente realizan mediciones sobre los frentes de onda y producen señales que representan a estos frentes, es tarea del "reconstructor" saber el significado de estas señales para realizar la reconstrucción adecuada, además de que el sistema de control debe determinar cómo tratar estos datos para poder transmitirlos adecuadamente al sistema corrector. En los siguientes párrafos, el problema de calcular la forma del frente de onda a partir de los datos proporcionados por un sensor de frente de onda se explicara de una manera general, se aborda más en explicar la manera en la que se recupera un frente de onda a partir de los datos proporcionados por un sensor de curvatura y la aplicación de la Transformada de Fourier para recuperar el frente de onda.

1.2.1.2.1 MÉTODO BASADO EN ZERNIKE

Los datos recuperados por el sensor de frente de ondas pueden ser representados como un vector S. El dato desconocido (frente de onda) es un vector ϕ, que se puede especificar como valores de fase en un arreglo, o, más frecuentemente, como coeficientes de Zernike (polinomios que permiten realizar la representación geométrica de las aberraciones). Se supone que la relación entre las mediciones y las incógnitas es lineal, al menos en la primera aproximación. La forma más general de una relación lineal está dada por la multiplicación de matrices,

$$S = A\phi$$

Donde la matriz A es llamada matriz de interacción. En los sistemas de óptica adaptativa la matriz de interacción es determinada experimentalmente: todas las posibles señales (modos de Zernike) son aplicados al espejo deformable y la reacción que presente a estas señales es almacenada. Una matriz de reconstrucción B realiza la operación inversa, la recuperación del frente de onda a partir de las mediciones obtenidas es:

$$\phi = BS$$

El número de mediciones es por lo regular mayor que el número de incógnitas, por lo que la solución se puede buscar utilizando el método de mínimos cuadrados. En el método de mínimos cuadrados lo que hacemos es buscar un vector de fase ϕ que mejor coincida con los datos. El reconstructor resultante es:

$$B = (A^t A)^{-1} A^T$$

1.2.1.2.2 MÉTODO BASADO EN EL LAPLACIANO

Los datos obtenidos por un sensor de curvatura están regidos por la Ecuación 1.2, la solución a este sistema de ecuaciones diferenciales se obtiene a partir de la solución de la ecuación de Poisson con las condiciones iniciales de Neumann, como lo demuestra Noll [11]. Sin embargo, Roddier [12] presenta un nuevo método para la recuperación del frente de onda basado en la FFT (del inglés, *Fast Fourier Transform*), ya que el operador Laplaciano es equivalente a una multiplicación por $u^2 + v^2$ en el dominio de Fourier. Como una primera aproximación, la señal se toma para ser el Laplaciano del frente de onda, y la primera estimación del frente de onda se obtiene resolviendo en el dominio de Fourier. En la Figura 1.4 se ilustra el algoritmo del proceso.

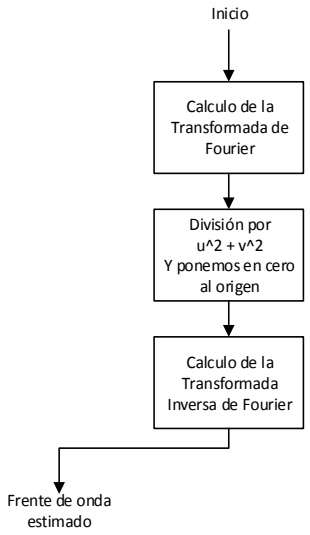


Figura 1.4._ Algoritmo para obtener una aproximación del frente de ondas en el dominio de Fourier [12]. Es posible aplicar este algoritmo ya que el operador Laplaciano es equivalente a una multiplicación por $u^2 + v^2$ en el dominio de Fourier.

Con los datos obtenidos del sensor calculamos su Transformada de Fourier, dividimos entre $u^2 + v^2$, poniendo ceros en el origen, *u* y *v* son las coordenadas en el dominio de Fourier, posteriormente obtenemos la Transformada Inversa de Fourier de la división obtenida anteriormente. El frente de onda obtenido es únicamente una aproximación, ya que no estamos tomando los límites de la pupila. En un análisis más a fondo hay que tomar en cuenta este dato y se puede encontrar en el trabajo elaborado por Roddier [12].

Las condiciones de contorno se toman en cuenta al forzar a cero la derivada radial dentro de una banda estrecha alrededor de los límites. Esto es equivalente a la resolución de la ecuación de Poisson con una pendiente cero en el borde radial como una condición de contorno de las

ecuaciones de Neumann. En la Figura 1.5 se muestra el algoritmo completo tomando las consideraciones anteriores.

Por la facilidad de implementar el algoritmo de la FFT en dispositivos lógicos programables, este método es muy viable para acoplarlo a un sistema embebido que obtenga de manera automática la recuperación del frente de onda.

La tesis está enfocada a validar la implementación de la FFT en una FPGA (del inglés, Field Programmable Gate Array), con el fin de que en un proyecto a futuro se complete el algoritmo para la recuperación del frente de onda y este sea implementado en un sistema embebido, para aplicaciones en óptica adaptativa.

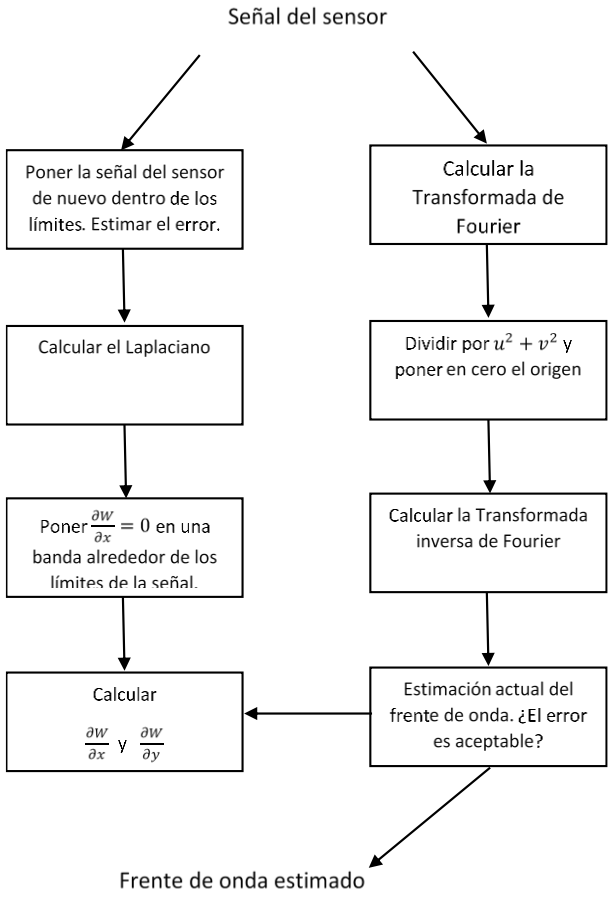


Figura 1.5._ Algoritmo para obtener una aproximación del frente de ondas en el dominio de Fourier [12], tomando en cuenta las condiciones iniciales.

1.2.2 CORRECCIÓN DEL FRENTE DE ONDA

Un frente de onda puede ser corregido utilizando muchos de los mismos mecanismos que causan su distorsión. Por ejemplo, una imagen de un espejo distorsionado puede ser corregida mediante la distorsión de otro espejo en la forma adecuada para compensar las variaciones de frente de onda. Las variaciones locales en el índice de refracción, que son la causa de una aberración, se puede invertir o conjugar, para proporcionar la corrección. La corrección para un frente de onda en la mayoría de los sistemas de óptica adaptativa está basada en el principio de la adición de un dispositivo de corrección para el haz de luz. Los dispositivos más comunes para realizar la compensación es la de los espejos activos. Hay que recordar que los sistemas de óptica adaptativa trabajan en lazo cerrado, donde el WFS *(Wave-Front Sensor*, sensor de frente de onda) mide el frente de onda incidente y envía los comandos correspondientes al dispositivo control y este los manda al sistema de compensación, Figura 1.6.

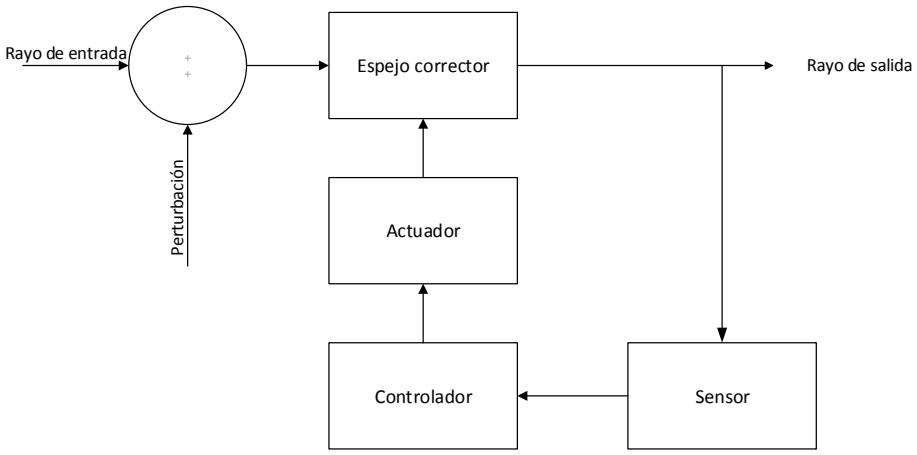


Figura 1.6._ Control en lazo cerrado.

Los espejos aptos para utilizar en los sistemas de óptica adaptativa son los espejos segmentados y los espejos deformables con superficies continuas.

Espejos segmentados. Estos espejos, en su totalidad, consisten en una serie de pequeños espejos espaciados estrechamente, con capacidades para la corrección de Tip/Tilt o Pistón en los frentes de onda, los modos de orden superior de corrección se pueden aplicar mediante la determinación de la contribución individual de cada uno de los segmentos. Una seria desventaja que se presenta en los espejos segmentados es que entre cada segmento existe una discontinuidad, la cual puede producir

resultados inesperados, como son, la pérdida de energía que se escapa entre las discontinuidades, además de que éstas actúan como una rejilla, produciendo efectos de difracción indeseados.

Espejos deformables con superficies continuas. Estos espejos son en general superficies continuas con dispositivos mecánicos (actuadores), que permiten inducirle deformaciones al espejo. Los actuadores que realizan la deformación pueden ser continuos, como los espejos de membrana o espejos bimorfos, o discretos, como pueden ser los que se colocan en los bordes del espejo, los cuales imparten momentos de flexión, o estar colocados perpendicularmente a la superficie para inducir la deformación. Los actuadores utilizados están clasificados en dos tipos, actuadores de fuerza y actuadores de desplazamiento.

Capítulo 2

Análisis de Fourier

2.1 Introducción

La Transformada de Fourier (TF) es una herramienta ampliamente utilizada en las ciencias físicas para el análisis de señales. Su principal objetivo es generar una función que muestra el contenido de frecuencia de una señal. Como resultado, ciertas características de la señal son más fácilmente analizadas o detectadas en el dominio de la frecuencia que en el dominio espacial o temporal.

FFT (del inglés *Fast Fourier Transform*) es la abreviatura usual de un eficiente algoritmo que permite calcular rápidamente la Transformada Discreta de Fourier (TDF) y su inversa (con pequeños cambios al algoritmo). La FFT es de gran importancia en una amplia variedad de aplicaciones digitales, desde el tratamiento digital de señales y filtrado digital en general, hasta el análisis de imágenes digitales. Antes de abordar por completo la TF en dos dimensiones, se dará un pequeño repaso a la TF en una dimensión, dando sus principales características, tanto para señales continuas como discretas.

2.2 La Transformada de Fourier en Tiempo Discreto

Las señales discretas, como su nombre lo indica, son señales que están definidas solamente en instantes discretos del tiempo.

Para el análisis de señales aperiódicas de energía finita se emplea la Transformada de Fourier en Tiempo Discreto (TFTD). Este tipo de señales involucran un intervalo de frecuencias que abarcan de $-\infty$ a $+\infty$. En la Tabla 2.1 se describen las ecuaciones de síntesis y análisis para la TFTD.

Tabla 2.1._ Análisis de señales periódicas en tiempo discreto, TFTD.

Ecuación de síntesis Transformada inversa	$x(n) = \frac{1}{2\pi}\int_{2\pi} X(w)e^{jwn}dw$
Ecuación de análisis Transformada directa	$X(w) = \sum_{n=-\infty}^{\infty} x(n)e^{-jwn}$

La TDF es un caso particular de la TFTD para secuencias de longitud finita, el cual al evaluarse el espectro solamente en unas frecuencias concretas se obtiene un espectro discreto. Se define entonces a la TDF de la siguiente forma:

Tabla 2.2._ Análisis de señales periódicas en tiempo discreto, TDF.

Transformada inversa	$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{j2\pi(nk/N)}$
Transformada directa	$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi(nk/N)}$

La exponencial $e^{-j2\pi/N}$ en muchos casos es representada como *W*, donde ésta constante es conocida comúnmente como *factor twiddle*.

2.2.1 La Transformada Discreta de Fourier en 2D

La TF y otras trasformadas en el espacio frecuencial en dos dimensiones tienen aplicación en el análisis de imágenes por diversas razones. Algunas de ellas tienen que ver con los propósitos de mejorar las imágenes, o seleccionar ciertas características o estructuras de interés para hacer mediciones sobre la imagen a tratar. La extensión de 1 dimensión a 2 dimensiones para la TDF es sencilla, la TDF en 2D está definida como,

$$X[k_1, k_2] = \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x[n_1, n_2]W_{N_1}^{n_1k_1}W_{N_2}^{n_2k_2} \qquad (2.1)$$

$$k_1 = 0, 1, \dots, N_1 - 1 \quad y \quad k_2 = 0, 1, \dots, N_2 - 1$$

Y la Transformada Discreta de Fourier Inversa (TDFI) se define como,

$$x[n_1, n_2] = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X[k_1, k_2] W_{N_1}^{-n_1 k_1} W_{N_2}^{-n_2 k_2} \tag{2.2}$$

$$n_1 = 0, 1, \dots, N_1 - 1 \quad y \quad n_2 = 0, 1, \dots, N_2 - 1$$

Donde para ambas ecuaciones W está definida como,

$$W_{N_1} = e^{-j2\pi/N_1} \qquad W_{N_2} = e^{-j2\pi/N_2}$$

Donde para la Ecuación 2.1, $x[n_1, n_2]$ es una secuencia discreta en 2 dimensiones de longitud uniforme en el dominio espacial, y para la Ecuación 2.2, $X[k_1, k_2]$ son los coeficientes de la TDF en el dominio frecuencial en 2 dimensiones. La TDF en 2D puede ser reescrita separando sus argumentos como,

$$X[k_1, k_2] = \sum_{n_2=0}^{N_2-1} \left(\sum_{n_1=0}^{N_1-1} x[n_1, n_2] W_{N_1}^{n_1 k_1} \right) W_{N_2}^{n_2 k_2} \tag{2.3}$$

$$k_1 = 0, 1, \dots, N_1 - 1 \quad y \quad k_2 = 0, 1, \dots, N_2 - 1$$

Y

$$x[n_1, n_2] = \frac{1}{N_2} \sum_{k_2=0}^{N_2-1} \left(\frac{1}{N_1} \sum_{k_1=0}^{N_1-1} X[k_1, k_2] W_{N_1}^{-n_1 k_1} \right) W_{N_2}^{-n_2 k_2} \tag{2.4}$$

$$n_1 = 0, 1, \dots, N_1 - 1 \quad y \quad n_2 = 0, 1, \dots, N_2 - 1$$

La Ecuación 2.3 se puede analizar como una TDF en 1D de la secuencia $x[n_1, n_2]$ de las columnas seguido por la TDF en 1D de los renglones. Similarmente, la Ecuación 2.4 es la TDFI en 1-D de las columnas seguido por la TDFI en 1D de los renglones. En la Figura 2.1 se ilustra este proceso.

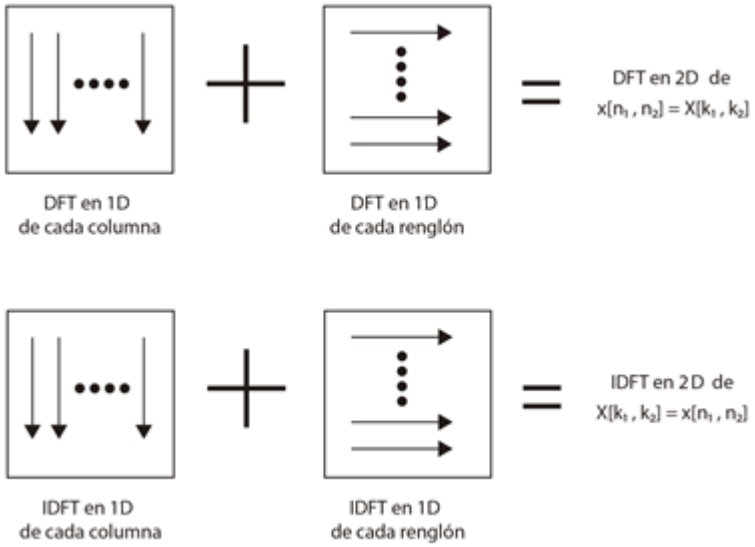


Figura 2.1._ Obtención de la TDF y la TDFI en 2D.

El proceso descrito anteriormente se puede intercambiar, obteniendo primero la TDF de los renglones seguido por la TDF de las columnas. Lo mismo para la TDFI, se obtiene primero la TDFI de los renglones seguido por la TDFI de las columnas, las Ecuaciones (2.5) y (2.6) muestran este reordenamiento.

$$X[k_1, k_2] = \sum_{n_1=0}^{N_1-1} \left(\sum_{n_2=0}^{N_2-1} x[n_1, n_2] W_{N_2}^{n_2 k_2} \right) W_{N_1}^{n_1 k_1} \tag{2.5}$$

$$K_1 = 0, 1, \dots, N_1 - 1 \quad y \quad k_2 = 0, 1, \dots, N_2 - 1$$

$$x[n_1, n_2] = \frac{1}{N_1} \sum_{k_1=0}^{N_1-1} \left(\frac{1}{N_2} \sum_{k_2=0}^{N_2-1} X[k_1, k_2] W_{N_2}^{-n_2 k_2} \right) W_{N_1}^{-n_1 k_1} \tag{2.6}$$

$$n_1 = 0, 1, \dots, N_1 - 1 \quad y \quad n_2 = 0, 1, \dots, N_2 - 1$$

El procedimiento anterior es consecuencia de la propiedad de separabilidad que presenta la TDF en 2D.

2.3 FFT

La TDF es una de las herramientas más importantes en el procesamiento digital de señales. El cálculo de la TDF requiere un gran número de multiplicaciones y adiciones complejas. Para N puntos, la TDF requiere N^2 multiplicaciones complejas y $N(N-1)$ adiciones complejas. Por ejemplo, para el cálculo de 8 puntos se requieren $8^2 = 64$ multiplicaciones complejas y $8(8-1) = 56$ sumas complejas. Si $N = 1024$, entonces se requieren $(1024)^2$ de multiplicaciones y $(1024)(1023)$ sumas, aproximadamente 1 millón de multiplicaciones complejas y sumas. Los algoritmos de la FFT (del inglés *Fast Fourier Transform*) reducen la carga computacional para calcular la TDF. Los algoritmos más populares para la obtención de la FFT están basados en el algoritmo de Cooley-Tukey y son los de radix-4 y radix-2., de los cuales el último es el más empleado. En la tabla 2.3 se muestra una comparación en la eficiencia del cálculo de la FFT.

Tabla 2.3 Comparación del esfuerzo computacional de la DFT y la FFT.

N	DFT – N^2 Multiplicaciones complejas y operaciones de suma	FFT - $N \log_2 N$ Multiplicaciones complejas y operación de suma	Esfuerzo computacional de la FFT comparada con la DFT (%)
8	64	24	37.50
32	1024	160	15.62
256	65536	1048	3.12
1024	1048576	10240	0.98
4096	16777216	49152	0.29

2.3.1 RADIX-2 EN DECIMACIÓN EN EL TIEMPO

Este algoritmo reordena la ecuación de la TDF en dos partes: la suma de los valores discretos con índices pares $n = [0, 2, 4, \dots, N-2]$ y la suma de los valores discretos con índices impares $n = [1, 3, 5, \dots, N-1]$ como se muestra en la siguiente ecuación.

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi(nk/N)} \tag{2.7}$$

$$X[k] = \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-\left(j\frac{2\pi(2n)k}{N}\right)} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) e^{-\left(j\frac{2\pi(2n+1)k}{N}\right)} \tag{2.8}$$

$$X[k] = \sum_{n=0}^{\frac{N}{2}-1} x(2n)e^{-\left(j\frac{2\pi nk}{\frac{N}{2}}\right)} + e^{-\left(j\frac{2\pi k}{N}\right)} \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)e^{-\left(j\frac{2\pi nk}{\frac{N}{2}}\right)} \quad (2.9)$$

$$X[k] = TDF_{\frac{N}{2}}\left[[x(0), x(2), \dots, x(N-2)]\right] + W_N^k TDF_{\frac{N}{2}}\left[[x(1), x(3), \dots, x(N-1)]\right] \quad (2.10)$$

Este algoritmo es conocido como decimación en el tiempo porque los datos que se reordenan son las muestras en el tiempo y Radix-2 porque son dos grupos los que se forman (pares e impares). Las operaciones que se realizan en este algoritmo se conocen como *operaciones de mariposa* y el diagrama de la Figura 2.2 muestra este algoritmo para 8 puntos.

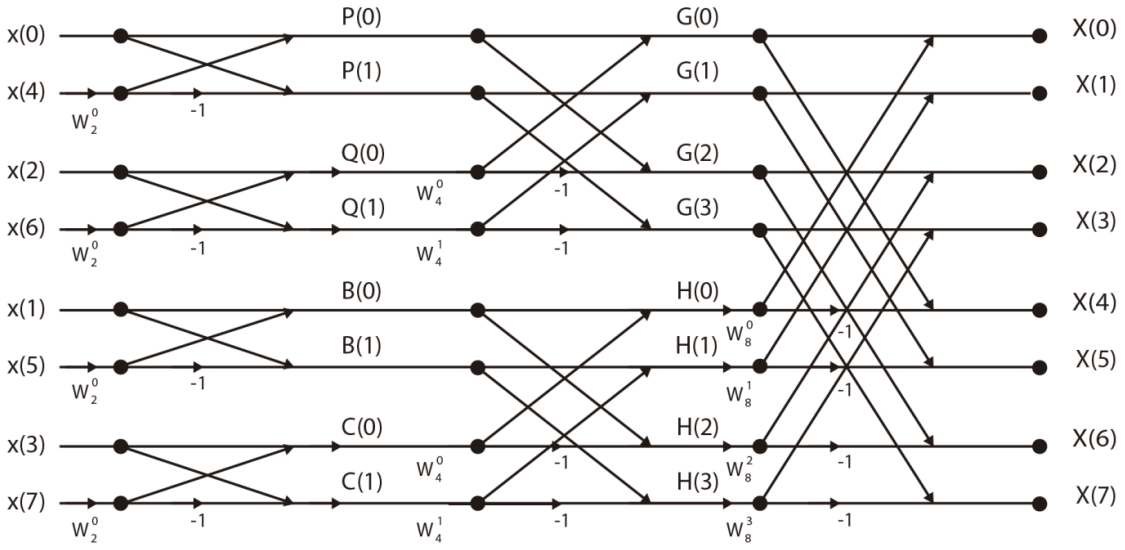


Figura 2.2._ Diagrama de mariposa del algoritmo Radix-2 en decimación en el tiempo para la obtención de la FFT con N = 8.

Cada proceso de decimación requiere $\frac{N}{2}$ multiplicaciones complejas. El número total de multiplicaciones complejas requeridas para completar todo el proceso es de $\frac{N}{2}\log_2 N$, por lo tanto el numero de operaciones se reduce de N^2 a $\frac{N}{2}\log_2 N$.

2.3.2 RADIX-2 EN DECIMACIÓN EN FRECUENCIA

El algoritmo de decimación en frecuencia para la obtención de la FFT está basado en la descomposición de la TDF de N-puntos, en la cual se va dividiendo la secuencia de salida hasta obtener una TDF de dos puntos. El reordenamiento de la TDF se realiza separando los valores con índice par de los datos discretos de la frecuencia $X[k] = [0,2,4, \dots, N-2]$ ylos valores con índice impar de los datos discretos de la frecuencia $X[k] = [1,3,5, \dots, N-1]$. El diagrama de la Figura 2.3 muestra este algoritmo para 8 puntos.

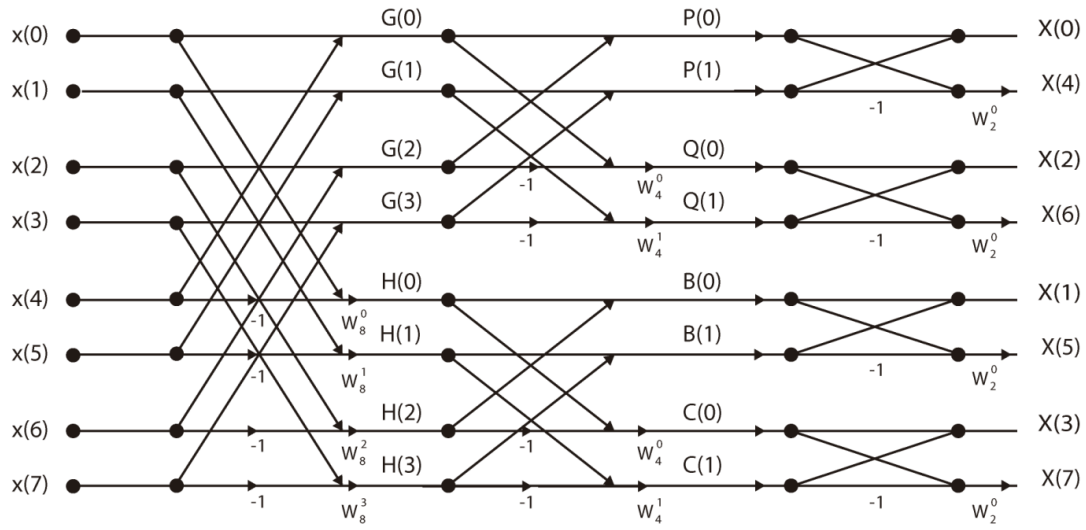


Figura 2.3._ Diagrama de mariposa para la obtención de la FFT en decimación en frecuencia.

Este algoritmo es llamado *decimación en frecuencia* porque las muestras que se separan en grupos de dos son los datos discretos de la frecuencia. El número total de multiplicaciones para el algoritmo Radix-2 en decimación en frecuencia es de $\frac{N}{2}\log_2 N$ multiplicaciones complejas.

Capítulo 3

Dispositivos lógicos programables

3.1 INTRODUCCIÓN

El rápido avance que se presenta en la industria de la electrónica ha hecho posible ir mejorando la tecnología para la fabricación de dispositivos y como consecuencia se ha ido mejorando la escala de integración de los circuitos integrados, dando como resultado la miniaturización de ciertos equipos electrónicos. Los circuitos integrados permiten que miles de componentes y circuitos puedan ser empacados en un pequeño espacio. Es posible diseñar circuitos de gran escala sin necesidad de que su estructura sea fija, gracias a los dispositivos lógicos programables. Un dispositivo lógico programable (o PLD, del inglés *Programable Logic Device*) es el nombre general que se le da a los circuitos integrados digitales capaces de ser programados para proporcionar una amplia variedad de funciones lógicas diferentes. Los PLDs se dividen en dos categorías: SPLD (del inglés, *Simple Programmable Logic Devices*) y CPLD (del inglés, *Complex Programmable Logic Devices*). Para grandes diseños, mapear el circuito a una CPLD puede convertirse en algo difícil y puede resultar en un uso inadecuado de los recursos que nos brinda el CPLD. Por estas razones los fabricantes de dispositivos programables buscaron otra alternativa para los dispositivos, basados en el uso de celdas pequeñas para programar e implementar la lógica combinacional y funciones de almacenamiento (comunicados por medio de interconexiones programables), dando lugar a los dispositivos FPGA (del inglés *Field Programmable Gate Array*).

3.2 FPGA

Los FPGAs son dispositivos prefabricados que pueden ser programados eléctricamente para implementar cualquier tipo de circuito lógico o sistema. Un FPGA es esencialmente una unidad de procesamiento de hardware que se puede reconfigurar "*in situ*". Las principales ventajas que proporciona un FPGA se listan a continuación.

- Utilización eficiente de recursos. El programador de un FPGA puede decidir los componentes del FPGA con los que quiere implementar un sistema.
- Posibilidad de implementar sistemas muy complejos en un solo chip.
- Desarrollo de sistemas de alta frecuencia. La elección de un FPGA para aplicaciones de tratamiento de señal se debe a su alta frecuencia de trabajo.
- Programables "*in situ*" (actualizables en campo).

Cada fabricante de FPGAs tiene su propia arquitectura para sus dispositivos, pero en general todos son una variación mínima (principalmente en el cambio de nombre de los componentes) de la arquitectura propuesta inicialmente por Xilinx® (cabe señalar que el FPGA que se utilizó en la tesis fue una de esta empresa), que se explica en las siguientes páginas.

3.2.1 ARQUITECTURA

De forma general un FPGA consiste en un arreglo de bloques lógicos programables de diferentes tipos. En la Figura 3.1 se muestra la estructura básica de un FPGA.

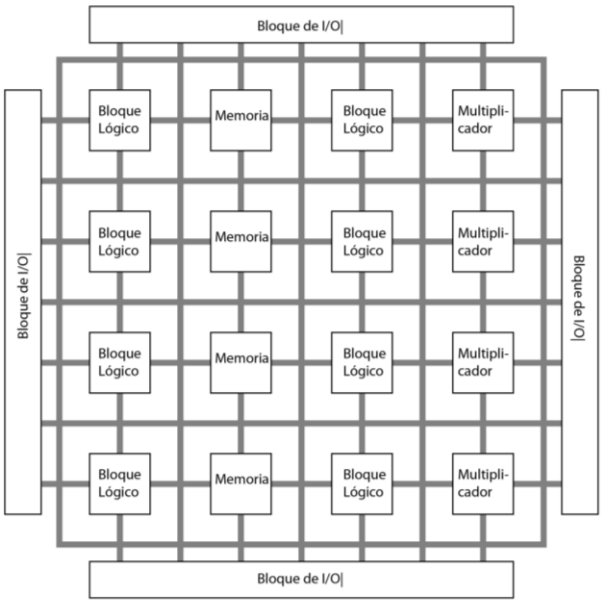


Figura 3.1._ Estructura general de un FPGA.

Los FPGAs están estructurados con tres principales tipos de bloques: los bloques entrada/salida, bloques lógicos configurables (CLB, del inglés *Configurable Logic Block*) y bloques

de interconexión. Los bloques lógicos están colocados en un arreglo matricial, los cuales están conectados entre sí por medio de los bloques de interconexión (los cuales son programables y están organizados con canales de enrutamiento verticales y horizontales). A su vez, estos dos tipos de bloques están rodeados por los bloques de entrada/salida, los cuales sirven para conectar el chip con el exterior.

Adicionalmente, la arquitectura de un FPGA incluye bloques embebidos de alto nivel, tales como multiplicadores y bloques de memoria RAM. Estos bloques pueden ser programados para implementar, ya sea lógica secuencial o combinacional.

3.2.1.1 BLOQUES LÓGICOS CONFIGURABLES

Estos bloques contienen la lógica programable de un FPGA. Un bloque lógico por lo regular consiste en tablas de consulta, multiplexores, compuertas y flip-flops. La tabla de consulta es una tabla de verdad almacenada en una SRAM, y proporciona las funciones de un circuito combinacional del bloque lógico. Se utiliza la sección de lógica combinacional, junto con varios multiplexores programables, para configurar las ecuaciones de entrada del flip-flop y la salida del bloque lógico.

3.2.1.2 BLOQUES DE ENTRADA/SALIDA (IOB, INPUT/OUTPUT BLOCKS)

Estos bloques se encuentran en el perímetro del chip, los cuales conectan los bloques lógicos con los pines externos del FPGA. Cada IOB puede ser usado para implementar varios estándares para señales en modo común, tales como LVCMOS (del inglés *Low Voltage Complementary Metal Oxide Semiconductor*, trabaja con un voltaje de 3.3 V) o LVTTL (del inglés *Low Voltage Transistor-Transistor Logic*, trabaja con un voltaje de 3.3 V). Además de que los IOB pueden ser emparejados con un IOB adyacente para formar señales diferenciales, tales como LVDS (del inglés *Low-Voltage Differential Signaling*).

Además los FPGAs cuentan con bloques especializados para comunicarse con dispositivos externos, estos bloques son los transceivers. Los transceivers usan un par de señales diferenciales para transmitir (TX) y otro par para recibir (RX), los cuales operan a velocidades altísimas (a Gigabits por segundo).

3.2.1.3 INTERCONEXIONES PROGRAMABLES

Todos los bloques internos en un FPGA están conectados por medio de cables con interruptores programables. Existen tres tipos de conexión para los componentes: conexiones locales, conexiones de interruptores y conexiones largas. En conjunto, las conexiones y los cables de conexión se conocen como recursos de enrutamiento. En la Figura 3.2 se muestran los tres tipos de conexiones. El recurso de enrutamiento local conecta “localmente” los bloques lógicos vecinos. Este tipo de conexión permite crear funciones lógicas complejas que un sólo CLB no sería capaz de realizar, pero varios CLB sí. El segundo recurso de enrutamiento son los interruptores, los cuales forman una matriz en donde los cables horizontales y verticales se interceptan. La matriz de interruptores nos da la posibilidad de conectar una matriz con otra, y esta a su vez con otra, así sucesivamente según lo necesitemos, los cuales eventualmente se conectan a los bloques lógicos, pudiendo de esta manera conectar bloques lógicos que se encuentran alejados entre sí y creando rutas de 90° o 180°. La desventaja de este tipo de enrutamiento es que por cada conexión que se realiza a una matriz de interruptores se va creando un retardo, que al final se acumula, llegando incluso a ser tan grande que haga ineficiente a nuestro sistema, debido a que el retardo puede que sea mayor comparándolo con la respuesta lógica de otros componentes implementados. El tercer tipo de enrutamiento es por medio de conexiones largas, estas conexiones permiten conectar dos bloques lógicos que se encuentran a una distancia considerable, esto sin crear un retardo significativo.

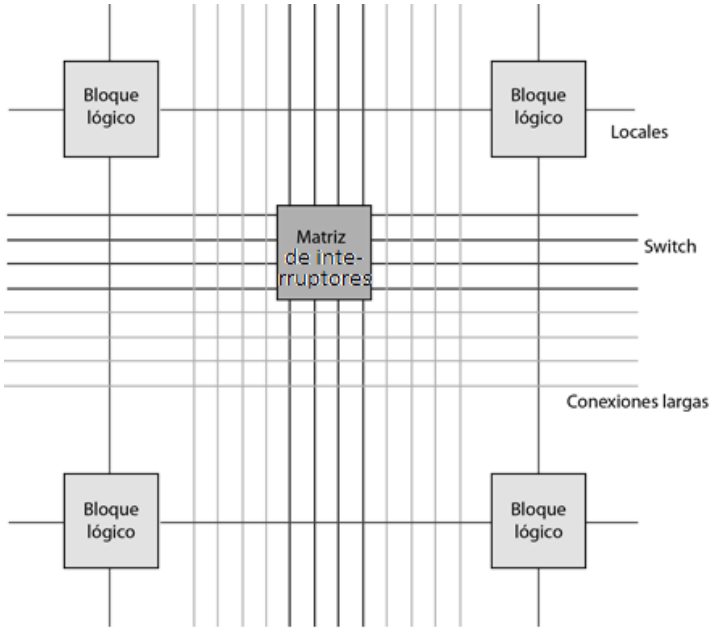


Figura 3.2._ Recursos de enrutamiento.

3.2.1.4 BLOQUES DE PROPÓSITO GENERAL

Además de los bloques vistos anteriormente, los fabricantes de FPGAs han añadido bloques adicionales a sus dispositivos. Estos bloques se añaden con el objetivo de hacer más eficiente el FPGA, mejorando el uso de los recursos con los que se cuenta. Los componentes más comunes que se agregan a los FPGA se describen a continuación.

Bloques de RAM: Muchos diseñadores necesitan ocupar memoria adicional de la que ofrecen los elementos lógicos por si solos, para solucionar este problema los fabricantes agregan un chip de memoria fija en el FPGA, conocido como bloque de RAM (BRAM, del inglés *Block Random-Access Memory*). La cantidad de memoria en un BRAM depende del dispositivo. Si el dispositivo contiene varios BRAM, estos pueden combinarse para crear un BRAM más grande. Además de que se pueden conectar los BRAM a diferentes relojes, lo que da como resultado el producir datos a diferentes frecuencias.

Bloques DSP: Los bloques DSP (del inglés *Digital Signal Processor*) están formados principalmente por multiplicadores, sumadores/restadores y elementos que realizan algunas otras operaciones bit a bit (tales como AND, OR, NOT o NAND). Estas operaciones se podrían diseñar por medio de los elementos lógicos, sin embargo, en términos de rendimiento, es más útil el utilizar los bloques DSP. Los bloques DSP se pueden combinar para realizar operaciones más complejas, tales como operaciones en coma flotante (de presión doble y simple), como pueden ser sumas, restas, multiplicaciones o divisiones.

DCM: La mayoría de los dispositivos FPGAs trabajan con un único reloj externo, el cual es fijo. Sin embargo estos FPGAs cuentan con bloques llamados DCM (*Digital Clock Manager*), los cuales nos permiten ajustar el reloj a diferentes frecuencia, esto con el fin de que el sistema pueda operar con diferentes ciclos de reloj. Por ejemplo si tenemos un reloj de 100 MHz podemos utilizar lógica digital para hacer una división del reloj a 25 MHz y de esta forma bajamos la frecuencia del reloj principal. La principal ventaja que nos brindan los DCM es que los relojes generados tendrán un bajo Jitter, además de que podemos hacer un cambio de base en el reloj, en la mayoría de los DCM podemos desfasar el reloj 90°, 180° ó 270°.

Procesadores: Muchas FPGAS recientes, como la Virtex 4 ó 5, se les agrega un procesador embebido. El uso de procesadores reduce el uso de recursos del FPGA y el consumo de potencia dependiendo del sistema que se esté diseñando. Por lo regular se ocupan procesadores del tipo RISC, tales como el IBM PowerPC 405 o PowerPC 440. Para los FPGAs que no cuentan con un procesador es posible grabarles uno haciendo uso de los *soft processors*.

3.2.2 APLICACIONES

En sus inicios los FPGAs eran utilizados para la implementación de circuitos de conexión (interfaces) entre bloques lógicos, máquinas de estado de mediana complejidad y procesos que requerían un procesamiento muy limitado, no muy complejo. Conforme los FPGAs se fueron haciendo más complejos, sus usos se fueron ampliando, abarcando diferentes mercados, como son los de telecomunicaciones, automotriz, video, redes de datos, entre otros. Los FPGAs se utilizan a menudo para crear prototipos de diseños ASIC (del inglés *Application-Specific Integrated Circuit*) o para proporcionar una plataforma sobre la cual verificar la implementación física de nuevos algoritmos. Debido al bajo costo que implica desarrollar sobre ellos y corto tiempo de lanzamiento al mercado, los FPGAs están cada vez más cerca de encontrar su camino en productos finales, lo que los hacen un competidor directo con los ASIC. El resultado final es que los FPGAs de hoy en día se pueden utilizar para implementar casi cualquier cosa. Para ser más específicos, los FPGAs están presentes en cuatro principales segmentos de mercado: ASIC, DSP, aplicaciones para microcontroladores embebidos y sistemas que comprenden la capa física [13].

En la Figura 3.3 se aprecia la distribución de las aplicaciones del uso de los FPGAs en el año 2008 [14].

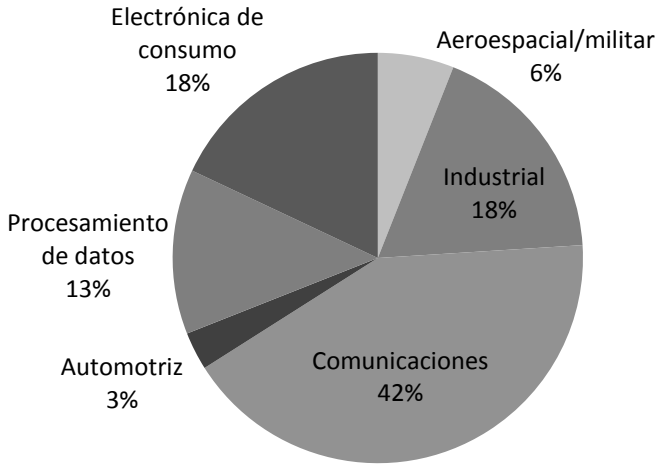


Figura 3.3 Distribución de las aplicaciones en FPGAs.

Como se observa en la gráfica, la principal aplicación de los FPGAs está orientada al procesamiento digital de señales (DSP), la cual es empleada en comunicaciones, imagenología, etc. La elección de un FPGA para aplicaciones de tratamiento de señal se debe a su alta frecuencia de trabajo, a su capacidad de procesamiento en paralelo, y su bajo precio en comparación con los ASICs. De esta aplicación se derivan una gran variedad de aplicaciones de los FPGAs, como pueden ser:

- Sistemas de visión artificial.
- Sistemas de imágenes médicas.
- Radio definida por software.
- Codificación y encriptación.
- Radioastronomía.
- Reconocimiento de voz.
- Aeronáutica y defensa.

Capítulo 4

Implementación de sistemas en FPGAs

4.1 INTRODUCCIÓN

Para que el diseño de un sistema se realice de manera correcta (ya sea en un FPGA o en cualquier otro dispositivo) debemos ajustarnos a un algoritmo, cuyo objetivo es guiarnos a través de las diversas etapas del ciclo de desarrollo. Este algoritmo se muestra en forma de diagrama de flujo en la Figura 4.1.

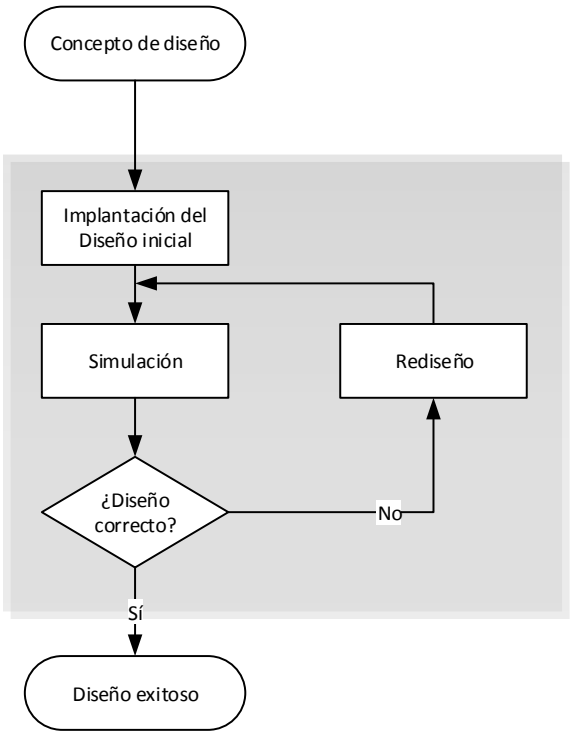


Figura 4.1._ Ciclo de diseño básico para el desarrollo de un sistema en FPGA.

Podemos dividir el proceso de diseño, en especial para el FPGA, en dos pasos:

1. Si se supone que se tiene un concepto primario acerca de lo que hay que lograr en el proceso de diseño, el primer paso consiste en generar un diseño inicial. Este paso requiere mucho esfuerzo, ya que comprende la total asimilación de la meta a la que se quiere llegar por parte del diseñador, y como se va a llegar a esa meta. Una vez que se ha comprendido lo anterior, se realiza un primer esbozo del sistema. Se dice que se requiere mucho esfuerzo porque el grueso de los diseños tiene ciertas metas específicas que sólo se alcanzan por medio del conocimiento, las habilidades y la intuición del diseñador. Además de que el diseñador debe tener muy en claro el tipo de metodología de modelado que va a emplear y el tipo de diagramas de flujo (como pueden ser los diagramas de estado, si es lógica secuencial) con los que se va a apoyar para realizar el esbozo del sistema.
2. El siguiente paso es la implantación del diseño (el cual es elaborado a partir del esbozo realizado en el paso anterior, en el caso de un FPGA, con un leguaje de descripción de hardware) y simulación de este. Para que la simulación tenga éxito es preciso tener adecuadas condiciones de entrada, las cuales deben poder aplicarse al diseño que se simula y más tarde al producto final que se someterá a pruebas. Al aplicar estas condiciones de entrada el simulador intenta comprobar que el producto diseñado se desempeñará como se requiere según las especificaciones del producto original. Si la simulación revela algunos errores hay que cambiar el diseño a fin de superarlos. La versión rediseñada se simula de nuevo para determinar si los errores desaparecieron. Este ciclo se repite hasta que la simulación indica un buen diseño. Un diseñador prudente dedica esfuerzos considerables a remediar los errores durante la simulación porque éstos suelen ser mucho más difíciles de corregir si se descubren tarde en el proceso de diseño. Aun así, algunos de ellos no se detectan durante la simulación, por lo que hay que enfrentarlos en etapas posteriores del ciclo de desarrollo.

El conjunto de herramientas de hardware y software que ayudan en el proceso de diseño de sistemas electrónicos reciben el nombre de herramientas para Automatización de Diseño Electrónico (EDA, del inglés *Electronic design automation*). El uso de herramientas EDA facilita el diseño de sistemas electrónicos, debido a que automatizan funciones como la descripción del sistema, la simulación, el prototipado y la producción final. Las herramientas EDA software, también conocidas como herramientas de Diseño Asistido por Computadora (CAD, del inglés *Computer-Aided Design*) o simplemente EDA-CAD, tienen por función ayudar a diseñar hardware a través de software. Por otro lado, las herramientas EDA hardware tienen por función facilitar el diseño de prototipos con base en un circuito integrado especifico.

La mayoría de las empresas dedicadas a la elaboración de FPGAs tienen sus propios sistemas CAD, estos programas ayudan a la implementación de sistemas electrónicos robustos para sus tarjetas.

En los párrafos siguientes se hará una breve descripción de las metodologías de modelado y los diagramas de estado, en forma de cartas ASM (del inglés, *Algorithmic State Machine*), los cuales son de vital importancia, como se explicó anteriormente, para que el diseñador desarrolle el primer esbozo de un sistema digital (primer paso en el diseño de hardware en el FPGA), además de que el sistema que se desarrolló en la tesis, al ser en su mayor parte secuencial, hace uso de estos diagramas. El segundo paso en el diseño puede ser completado con ayuda de un sistema CAD, por lo que también se describirán las herramientas que componen estos sistemas. Sin embargo antes de adentrarnos a los sistemas CAD, tenemos que tener bien claro en qué consisten los lenguajes de descripción de hardware y en especial del lenguaje VHDL (del inglés, VHSIC *Hardware Description Language*, y VHSIC, *Very High Speed Integrated Circuit*), ya que este lenguaje fue el que se ocupó en el desarrollo del proyecto.

4.2 METODOLOGÍAS DE MODELADO

Las metodologías son utilizadas para realizar el diseño conceptual de un sistema digital y son independientes de sistema CAD y el HDL empleados (pero muy ocupadas en el lenguaje VHDL). Las metodologías más empleadas son las siguientes:

Metodología de diseño ascendente: En esta metodología, el circuito o sistema que se pretende realizar comienza con la descripción de los componentes básicos, utilizando primitivas de esos componentes, que más tarde se agrupan en diversos módulos, y estos a su vez se agrupan para formar módulos de mayor complejidad y así sucesivamente hasta formar un sólo bloque que represente el sistema completo. No implica una estructura jerárquica de los elementos que componen el sistema, esta estructuración se realiza tras la descripción del circuito, y por lo tanto, no resulta necesaria. El flujo de diseño en la metodología ascendente es bastante ineficiente, por lo que se considera como una forma de diseñar poco apta y en el caso de diseños grandes, con la unión de más de mil componentes, puede generar diseños que no funcionen adecuadamente, con dificultades en la detección de errores y anomalías de funcionamiento.

Metodología de diseño descendente: La metodología de diseño descendente es un proceso que consiste en capturar una idea en un nivel de abstracción alto e implementarla; partiendo de esa descripción abstracta y después ir incrementando el nivel de detalle según sea necesario. El sistema inicial se va subdividiendo en módulos, estableciendo una jerarquía. Cada módulo se subdivide cuantas veces sea necesario hasta llegar a los componentes primarios del diseño. Dentro de las ventajas del uso de la metodología descendente se encuentran que:

1) Incrementa la productividad del diseño, permite especificar funcionalmente en un nivel de abstracción alto sin considerar la implementación del mismo a nivel compuertas.
2) Incrementa la reutilización del diseño, se utilizan tecnologías genéricas en el proceso de diseño. Esto permite reutilizar los diseños al cambiar la tecnología de implementación.
3) Una detección de errores rápida, con más tiempo dedicado a la definición y al diseño.

4.3 DIAGRAMAS ASM

Al diseñar un sistema, muchas veces se requiere que este describa operaciones secuenciales, los diagramas ASM son un tipo especial de diagramas para describir este tipo de operaciones y son muy ocupados por los diseñadores que programan en VHDL. El diagrama ASM describe la secuencia de los eventos, es decir, el orden en como ocurren los eventos con respecto al tiempo, así como la relación de temporización entre los estados de un control secuencial y de los acontecimientos que se producen al pasar de un estado a otro (los eventos que están sincronizados con los cambios de estado). El diagrama ASM se compone de tres elementos básicos:

1) El cuadro de estado.
2) El cuadro de decisión.
3) El cuadro condicional.

Un estado en la secuencia de control se indica con un cuadro de estado, como se observa en la Figura 4.2. La forma del cuadro de estado es rectangular dentro de la cual se escriben operaciones de registro o nombres de señal de salida que el control genera mientras se encuentra en ese estado. El estado recibe un nombre simbólico, el cual se coloca en la esquina superior izquierda de la casilla. El código binario (opcional) asignado al estado se coloca en la esquina superior de la derecha.

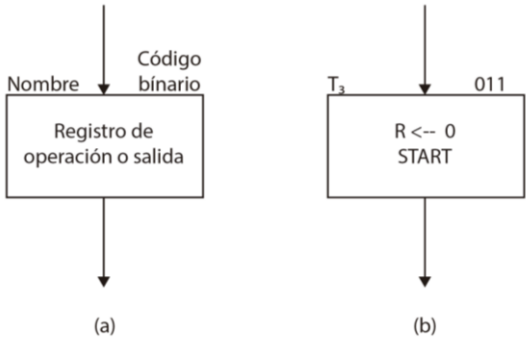


Figura 4.2._ Cuadro de estado, a) Descripción general, b) Ejemplo específico.

En la Figura 4.2b se muestra un ejemplo específico de un cuadro de estado. El estado tiene el nombre simbólico T_3, y el código binario asignado a él es 011. Dentro del cuadro se escribe la operación del registro R ← 0, la cual indica que el registro R se despeja a 0 cuando el sistema está en el estado T_3. El nombre START dentro de la casilla puede indicar una señal de salida que inicia cierta operación.

El cuadro de decisión escribe el efecto de una entrada en el subsistema de control. Es una casilla con forma de rombo con dos o más trayectorias de salida, como se muestra en la Figura 4.3. La condición de entrada que va a probarse está escrita dentro de la casilla. Una trayectoria de salida se toma si la condición es cierta y la otra cuando la condición es falsa. Las dos trayectorias se indican por 1 y 0.

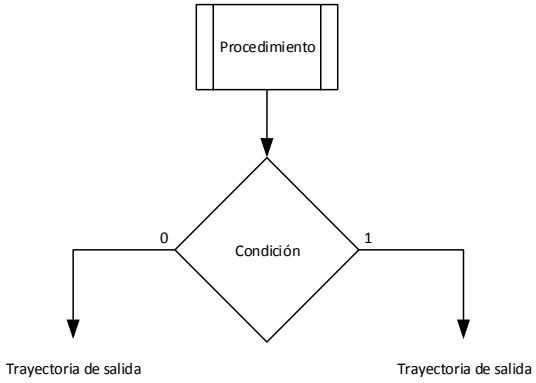


Figura 4.3._ Cuadro de decisión.

El tercer elemento, el cuadro condicional, es de uso exclusivo en los diagramas ASM. La forma de este elemento se puede observar en la Figura 4.4a, se diferencia del cuadro de estado por su forma ovalada.

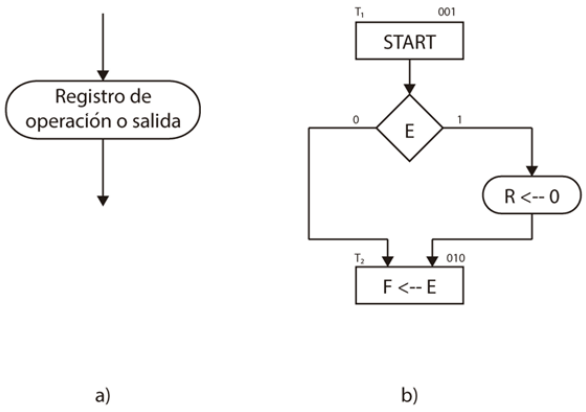


Figura 4.4._ a) Representación del cuadro condicional, b) Ejemplo de uso del cuadro condicional.

La trayectoria de entrada al cuadro condicional debe llegar desde una de las trayectorias de salida del cuadro de decisión. Las operaciones de registro o salidas listadas dentro de la casilla de condición son del tipo Mealy, por lo cual se generan durante un estado dado. En la Figura 4.4b se ilustra un ejemplo de uso de un cuadro condicional. El control genera una señal de salida de START cuando se encuentra en el estado T_1. Mientras se encuentra en el estado T_1, el control verifica el estado de la entrada *E*. Sí *E = 1*, entonces *R* se despeja a 0; en otra forma, *R* permanece sin cambio. En cualquier caso, el estado siguiente es T_2.

4.3.1 BLOQUES ASM

Un bloque ASM es una estructura que engloba un cuadro de estado y todas las casillas de decisión y condición conectadas a sus trayectorias de salida, o un cuadro de decisión con sus respectivos cuadros de condición. Un bloque ASM tiene una entrada y cualquier número de trayectorias de salida y consiste en uno más cuadros interconectados. Un cuadro de estado sin ningún cuadro de decisión o condición constituye un bloque ASM simple. El diagrama distingue el bloque ASM con líneas punteadas alrededor de la estructura entera. Las operaciones de las casillas de estado, decisión y condición que se encuentran dentro de un bloque se ejecutan en una sola transición del pulso de reloj. En la Figura 4.5 se muestra un ejemplo de un diagrama ASM con un bloque.

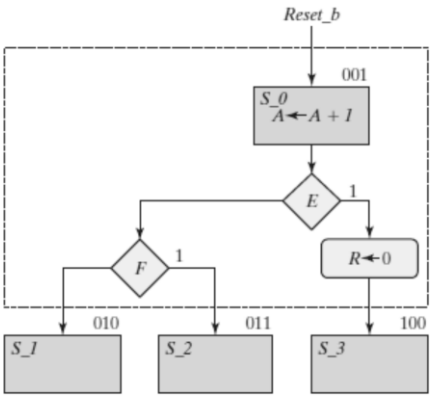


Figura 4.5._ Ejemplo de un bloque ASM.

4.4 VHDL

VHDL es un lenguaje de programación estándar industrial para describir circuitos digitales. La estructura general de un programa en VHDL está formada por módulos o unidades de diseño,

cada uno de ellos compuesto por un conjunto de declaraciones e instrucciones que definen, describen, estructuran, analizan y evalúan el comportamiento de un sistema digital. Existen dos tipos de unidades de diseño en VHDL:

1) Declaración de entidad (entity declaration).
2) Arquitectura (architecture).

A continuación se describirán las estas unidades.

Entidad. Es el bloque elemental de diseño en VHDL. Las *entidades* son todos los elementos electrónicos (sumadores, contadores, compuertas, flip-flops, memorias, multiplexores, etc.) que forman de manera individual o en conjunto un sistema digital. La entidad puede representarse de muy diversas maneras. Por ejemplo, la Figura 4.6a muestra la arquitectura de un sumador completo a nivel compuertas; ahora bien, esta entidad se puede representar a nivel de sistema indicando tan solo las entradas (Cin, A y B) y salidas (SUMA y Cout) del circuito: Figura 4.6b. De igual forma, la integración de varios subsistemas (medio sumador) puede representarse mediante una entidad (Figura 4.6c). Los subsistemas pueden conectarse internamente entre sí; pero la entidad sigue identificando con claridad sus entradas y salidas generales. Una entidad la podemos ver como el encapsulado de un circuito integrado, en la cual únicamente vemos sus entradas y salidas.

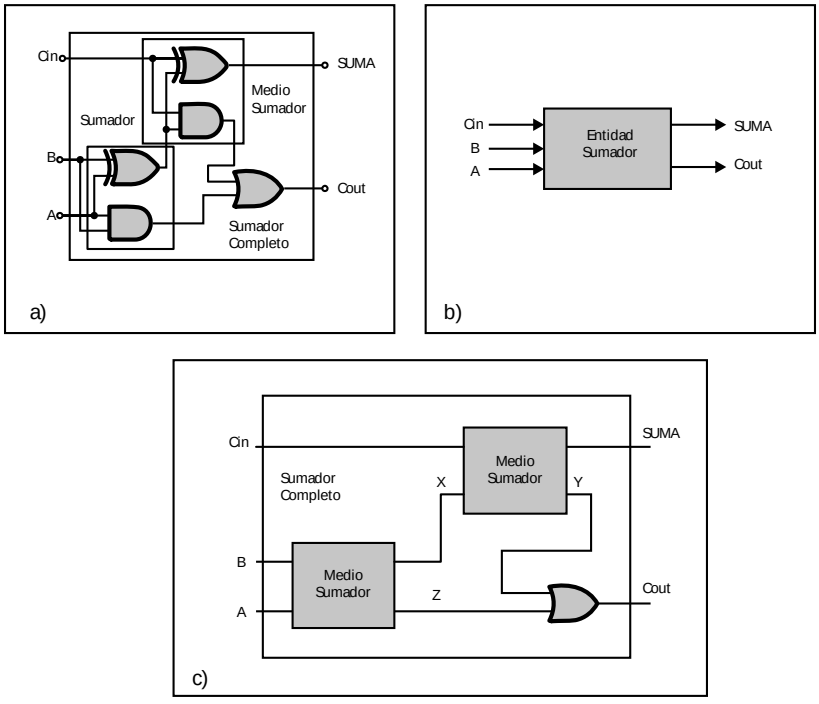


Figura 4.6._ Representación de una entidad para ser descrita en código VHDL.

Arquitectura. Una arquitectura (*architecture*) se define como la estructura que describe el funcionamiento de una entidad, de tal forma que permita el desarrollo de los procedimientos que se llevaran a cabo con el fin de que la entidad cumpla las condiciones de funcionamiento deseadas. La gran ventaja que presenta VHDL para definir una arquitectura radica en la manera en que puede describirse los diseños; es decir, mediante el algoritmo de programación empleado se puede describir desde el nivel de compuertas hasta sistemas complejos.

De manera general, los estilos de diseño (niveles de abstracción en VHDL) en la programación de arquitecturas se clasifican como:

- Estilo Funcional.
- Estilo por flujo de datos.
- Estilo estructural.

4.4.1 VENTAJAS DEL USO DE VHDL

Algunas ventajas del uso de VHDL son las siguientes:

- Permite diseñar, modelar y comprobar (simular) cualquier sistema digital desde un nivel de abstracción alto hasta un nivel de abstracción bajo o de definición estructural de compuertas y biestables.
- Reutilización de código, los módulos creados en VHDL pueden utilizarse en diferentes diseños. Esa misma descripción se puede utilizar en distintas tecnologías sin la necesidad de rediseñar todo el circuito.
- Al no ser un lenguaje propietario, cualquier usuario puede desarrollar un sistema en VHDL y comercializarlo.
- Con la modularidad que permite VHDL, se puede dividir o descomponer los diseños hardware y su descripción en unidades más pequeñas.

4.4.2 NIVELES DE ABSTRACCIÓN EN VHDL

VHDL soporta principalmente tres tipos de estilos de diseño, los cuales dependen del nivel de abstracción que se esté manejando, en general estos son:

1) El estilo comportamental o funcional.

2) El de flujo de datos.
3) El estructural.

El estilo comportamental o funcional es el nivel de abstracción más elevado que soporta VHDL. Cuando se describe un circuito usando este nivel de abstracción, el circuito es descrito en términos de su operación con el tiempo, el diseñador sólo describe el comportamiento del sistema sin preocuparse de los componentes internos del mismo. Usando un nivel de abstracción comportamental se puede describir las operaciones de un circuito secuencial a nivel de registros. Escribir diseños de circuitos a un nivel comportamental es como programar en cualquier lenguaje de alto nivel, se escriben pequeños programas que operan de manera secuencial y se comunican entre ellos mediante sus interfaces.

El estilo de flujo de datos describe los circuitos en términos de cómo los datos se mueven a través del sistema; describe cómo es que la información fluye a través de los registros del circuito. El diseñador toma en cuenta las distintas señales que interactúan en un circuito, así como el comportamiento por medio de ecuaciones lógicas y sentencias de asignación. Es comúnmente llamado Transferencia Lógica de Registros (RTL, del inglés *Register Transfer Logic*). Es un nivel intermedio que permite que la lógica combinacional sea simplificada, mientras las partes más importantes del circuito (los registros) sean especificadas de acuerdo a la función a modelar.

El estilo estructural, o nivel lógico, es el tercer nivel de abstracción, es usado para describir circuitos en términos de sus componentes. Puede ser usado para crear una descripción a bajo nivel de un circuito, como la descripción a nivel transistor, o una descripción a nivel muy elevada como un diagrama de bloques. El diseñador emplea los recursos que el lenguaje proporciona para describir las interconexiones entre los distintos componentes del circuito.

4.5 HERRAMIENTAS CAD PARA EL DISEÑO DE SISTEMAS EN FPGAS

Para implementar un sistema en un FPGA un diseñador se apoya en un sistema CAD. Para diseñar un circuito lógico se requieren varias herramientas CAD (las cuales casi siempre están empaquetadas en el "*sistema CAD"*) destinadas a realizar las siguientes tareas: ingreso del diseño, síntesis y optimización, simulación y diseño físico. En este contexto, la palabra *herramienta* significa un programa de software que permite al usuario realizar una tarea específica. El flujo de diseño para la implementación de un circuito lógico en un sistema CAD se ilustra en la Figura 4.7.

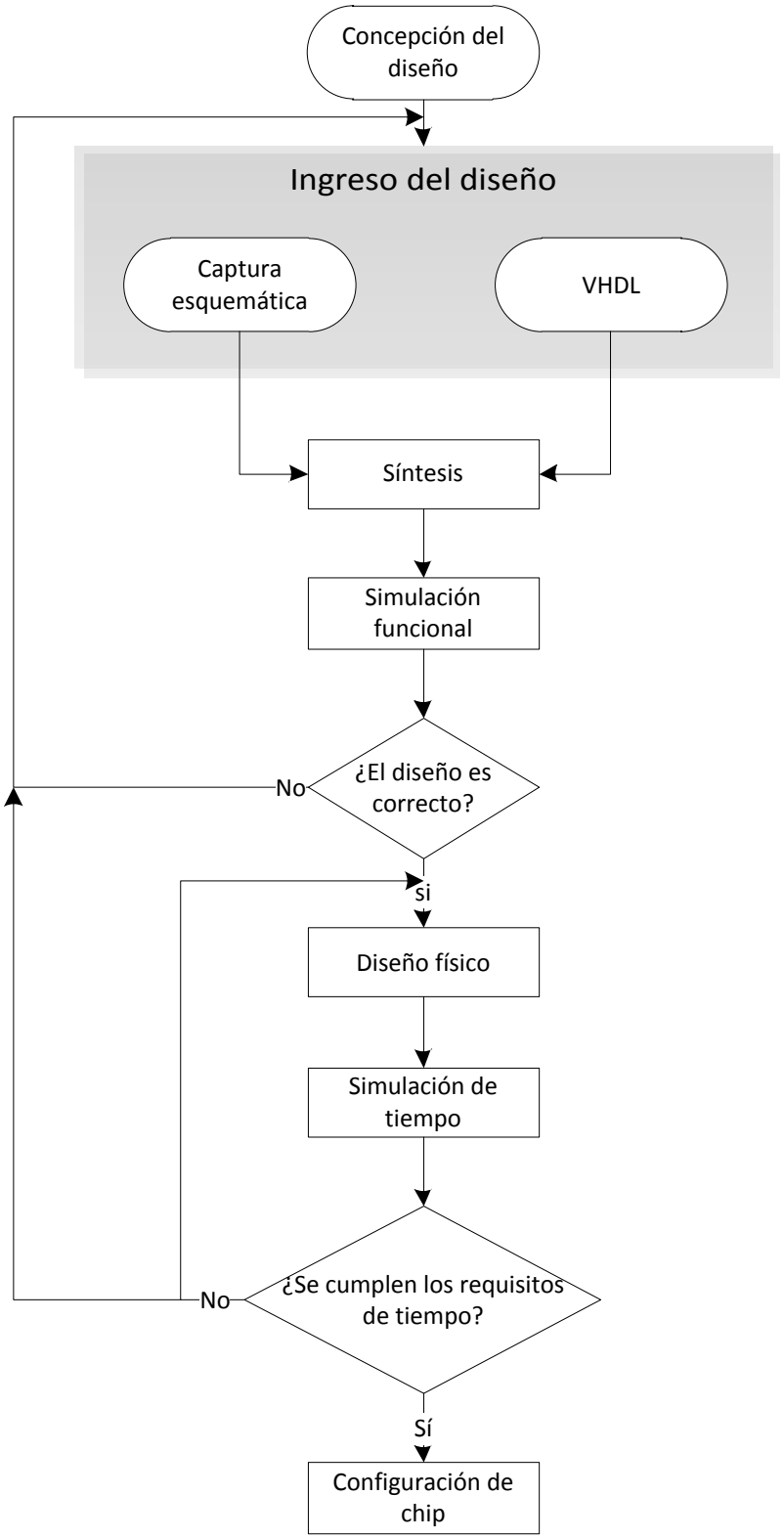


Figura 4.7._ Flujo de diseño en un sistema CAD.

El proceso inicia con la generación de un diseño inicial, en el cual se ocupan las herramientas mencionadas en los Temas 4.2 y 4.3. Una vez que se tiene el diseño, se prepara una descripción del circuito deseado en forma de un lenguaje de descripción de hardware como VHDL (explicado en el Tema 4.4). El código VHDL se procesa después en la etapa de síntesis del sistema CAD, para posteriormente ser implementado físicamente. La totalidad del sistema desarrollado se implementó siguiendo el flujo de diseño antes mencionado.

4.5.1 SÍNTESIS

La síntesis es el proceso por el que se genera un circuito lógico a partir de las especificaciones del usuario. En la Figura 4.8 se muestran tres fases comunes que comprende el proceso de síntesis.

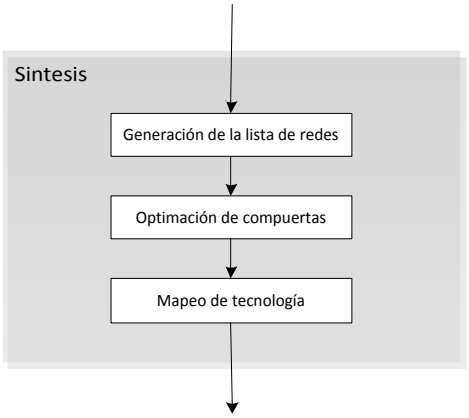


Figura 4.8._ Etapas incluidas en la herramienta de síntesis.

Generación de lista de redes: En la fase de generación de la lista de redes (*netlist*) se revisa la sintaxis del código y se informa respecto a cualquier error hallado, por ejemplo, señales indefinidas, paréntesis faltantes y palabras reservadas erróneas. Una vez corregidos todos los errores se genera una lista de redes del circuito según lo determina la semántica del código de VHDL.

Optimización de compuertas: En esta fase se realiza la optimización del diseño generado en la etapa anterior. Estas optimizaciones manipulan la lista de redes para obtener un circuito equivalente, pero mejor, de acuerdo con las metas de optimización, estas metas son configuradas por parte del programador.

Mapeo de la tecnología: Es la fase final de la síntesis. En esta fase se determina cómo puede producirse cada uno de los componentes de la lista de redes con los recursos disponibles en el chip objetivo.

4.5.2 SIMULACIÓN FUNCIONAL

Un circuito representado en forma de expresiones lógicas se simula, entre otras cosas, para verificar que funcione como se espera. La herramienta que cumple esta tarea recibe el nombre de simulador funcional. Utiliza expresiones lógicas (conocidas como ecuaciones) generadas durante la síntesis y supone que se implementarán con compuertas perfectas por las que pasarán instantáneamente las señales. El simulador requiere que el usuario especifique las valoraciones de las entradas del circuito que han de aplicarse durante la simulación. Los resultados de la simulación suelen entregarse en forma de diagrama de tiempo que el usuario examina para verificar que el circuito opera como se requiere. La simulación supone que no hay retrasos de propagación en el circuito, pues el objetivo consiste en evaluar la funcionalidad básica y no el tiempo. El retardo existente en la familia de FPGAs Spartan®-3E se menciona en el Tema 3.2.1.1.2. Una vez que la lista de redes producida por la síntesis es funcionalmente correcta, podemos pasar a la etapa de diseño físico.

4.5.3 DISEÑO FÍSICO

En esta etapa se determina exactamente cómo se implementará la lista de redes sintetizada en el chip objetivo. Esta etapa comprende tres fases (Figura 4.9): colocación, enrutamiento y análisis de tiempo estático.

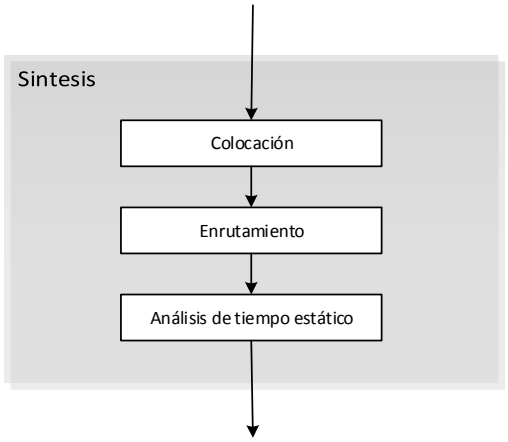


Figura 4.9._ Fases de diseño físico.

Colocación: En la fase de colocación se elige una ubicación en el dispositivo para cada bloque lógico de la lista de redes con mapeo de tecnología. Para hallar una buena solución de colocación ha de considerarse una serie de ubicaciones diferentes para cada bloque lógico. Para un circuito

grande, que pueda contener decenas de miles de bloques, esto representa un problema arduo. Por ejemplo, supongamos que un circuito tiene N bloques lógicos y que deben colocarse en el FPGA que también contiene exactamente N bloques. Una herramienta de colocación tiene N opciones para la ubicación del primer bloque que selecciona. Quedan *N-1* opciones para el segundo bloque, *N-2* opciones para el tercero y así sucesivamente. La multiplicación de estas opciones arroja un total de $(N)(N-1)\dots(1) = N!$ soluciones de colocación posibles. Para valores aún más moderados, *N!* es un número enorme, lo que significa que deben usarse técnicas heurísticas para hallar una buena solución mientras se considera sólo una pequeña fracción del número total de opciones.

Enrutamiento: Una vez que se elige la ubicación en el chip para cada bloque lógico de un circuito, en la fase de enrutamiento se conectan los bloques entre si usando los cables que hay en el chip. La herramienta CAD de enrutamiento intenta hacer el mejor uso de varios tipos de cables, por ejemplo, conexiones eficientes para cadenas de acarreo (en el caso de que se implementara un sumador).

Análisis de tiempo estático: Después de que el enrutamiento se completa se conocen los retrasos de tiempo para el circuito implementado, ya que el sistema CAD calcula éstos para todos los bloques y cables del chip. Una herramienta de análisis de tiempo estático examina esa información de retraso y produce un conjunto de tablas que cuantifican el rendimiento del circuito. Un ejemplo de estas tablas se puede observar en la Sección 5.7.1, las cuales se obtuvieron a partir del análisis de tiempo para el sistema que se implementó en la tesis.

4.5.4 SIMULACIÓN DE TIEMPO

Tanto las compuertas como otros elementos lógicos se implementan con circuitos electrónicos. Un circuito electrónico no puede cumplir su función de manera instantánea. Cuando cambian los valores de las entradas al circuito se precisa cierto tiempo antes de que ocurra el cambio correspondiente en la salida. Esto se llama retardo de propagación del circuito. Lo que se busca al realizar la simulación de tiempo es obtener el dato del mayor retardo en el circuito implementado, este retardo se obtiene con las tablas producidas en el análisis de tiempo estático.

Capítulo 5

Diseño e implementación del sistema

5.1 INTRODUCCIÓN

Los sistemas embebidos son aquellos que están diseñados para realizar una o pocas tareas con funciones dedicadas (diseñados para cubrir necesidades específicas), por lo regular en un sistema en tiempo real. El proyecto a desarrollar consiste en la implementación de un sistema embebido que permita efectuar la transformada de Fourier de una imagen, con el fin de validar su utilidad en aplicaciones para la reconstrucción de un frente de onda en sistemas de óptica adaptativa. En el mercado existen varias alternativas que permiten implementar sistemas para efectuar el procesamiento digital de señales, tales como los DSPs y FPGAs. Los procesadores DSP tienden a ser más adecuados cuando la aplicación es muy compleja o se requieren muchas configuraciones, y las tasas de procesamiento son lo suficientemente bajas para permitir su implementación. En ciertas aplicaciones donde se tengan varios algoritmos de procesamiento puede llegar a ser necesario ocupar varios DSPs. Los FPGAs son utilizados cuando se cuenta con una alta tasa de flujo de datos y es necesario procesarlos en un tiempo relativamente corto, por ejemplo en óptica adaptativa, donde se trabaja con frecuencias superiores a los 2 Hz. Por ejemplo, en términos de GMACs (del inglés *Giga Multiply-Accumulate*[2]), un FPGA puede remplazar una gran cantidad de DSPs, como consecuencia de esto el desempeño de una FPGA medido en GMAC/s (*Giga Multiply-Accumulate operations per Second*) es superior al de un DSP. En promedio los DSP ofrecen un rendimiento de 10 GMAC/s y las FPGAs un rendimiento de 700 GMAC/s. En muchas aplicaciones se utilizan ambas tecnologías, siendo la FPGA para el preprocesamiento y el DSP para el procesamiento de datos. Sin embargo, la ventaja que ofrece el FPGA es que en el además de la etapa del DSP, se pueden implementar otras etapas, como las de comunicación con diversos

[2] En procesamiento digital de señales es común comparar los DSP por el número de bloques MAC (del inglés, Multiplier–ACcumulator) que contienen y el número de operaciones por segundo que estos pueden realizar (GMAC/s).

dispositivos, pudiendo implementar una gran cantidad de protocolos de comunicación, y no sólo a las que está limitado un DSP. Además el FPGA nos permite implementar varios DSP que funcionen en paralelo, aquí es donde radica su fortaleza (procesamiento en paralelo). Los módulos que se pueden implementar en un FPGA vienen dados por las características de cada una de las tarjetas existentes en el mercado.

5.2 MODELADO DEL SISTEMA

Antes de iniciar con la descripción del sistema es necesario mencionar algunos aspectos relevantes en el modelado del mismo, tales como, las herramientas EDA elegidas y la metodología de diseño a utilizar. Como se mencionó anteriormente (Capítulo 4) las herramientas EDA tienen como objetivo facilitar el proceso de diseño de un sistema electrónico y están divididas en herramientas EDA-software (EDA-CAD) y herramientas EDA-hardware. Para el diseño y modelado del sistema (los diferentes módulos a implementar) se ha elegido como herramienta de hardware a la tarjeta de desarrollo Nexys™2 de la compañía *Digilent*® y como herramienta EDA-CAD al programa Xilinx ISE® en su versión 13.4.

La herramienta ISE® (de la empresa Xilinx®), es un entorno de desarrollo integrado (IDE), cuyo objetivo es facilitar el proceso de diseño de un sistema digital mediante la automatización del tránsito por las diferentes fases que integran dicho proceso (lo que es el modelado, simulación y la síntesis lógica hasta llegar a la implementación del sistema en la arquitectura elegida). En la Figura 5.1 se muestra el panel principal del entorno de desarrollo de ISE®.

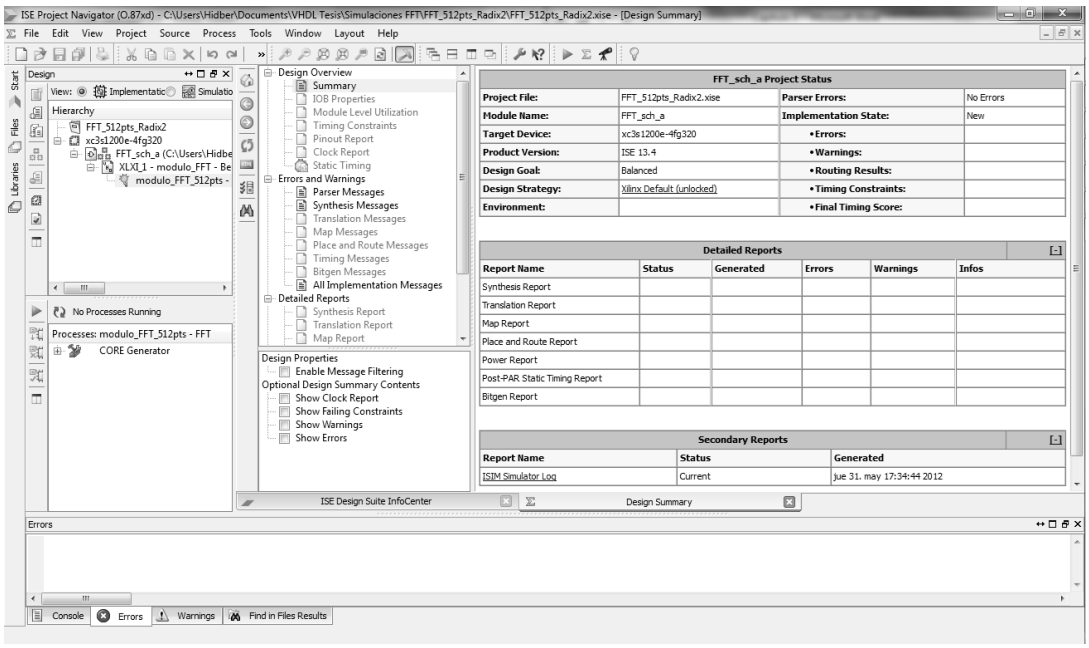


Figura 5.1._ Herramienta Xilinx ISE®.

En el Capítulo 4 se mencionaron dos tipos de metodologías utilizadas para realizar el diseño de un sistema digital, estas son: metodología ascendente y metodología descendente. La metodología elegida para desarrollar el sistema propuesto en la tesis es la descendente. Para el modelado de los diversos módulos que conformaran al sistema se ha elegido el estilo comportamental y el estilo estructural para unirlos.

La herramienta elegida para la implementación del sistema es el FGPA Spartan®-3E 1200, el cual se encuentra dentro de la tarjeta de desarrollo Nexys™2 (Figura 5.2). Se eligió esta FPGA porque se encontraba disponible en el Laboratorio de Electrónica y Detectores (LED) del Instituto de Astronomía de la Universidad Nacional Autónoma de México, cuando inició este proyecto de tesis.

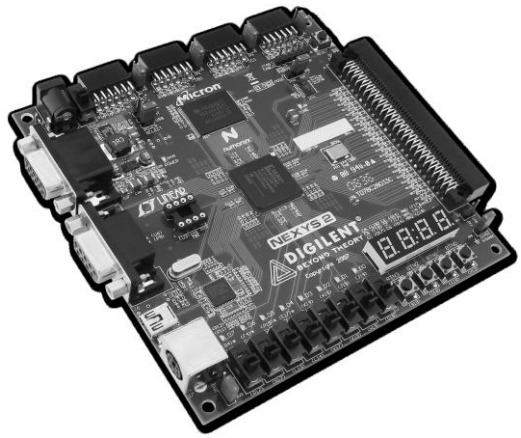


Figura 5.2. Tarjeta de desarrollo Nexys™2

La tarjeta Nexys™2 es un dispositivo de desarrollo para la implementación de sistemas digitales. Las características más relevantes de este componente se listan a continuación:

- FGPA Spartan®-3E 1200.
- Utiliza el puerto USB para configurar el FPGA.
- Compatible con la herramienta Xilinx ISE®.
- Incluye 16 MBytes de PSRAM (del inglés, Pseudo-Static DRAM) de la compañía Micron y 16 Mbytes StrataFlash.
- Plataforma flash no volátil para la configuración del FPGA.
- Oscilador integrado de 50 MHZ y socket para un oscilador extra.
- Conectores de expansión con 60 entradas/salidas al FPGA.
- 8 LEDs (del inglés, Light-Emitting Diode), 4 push buttons, 8 interruptores y 4 displays de 8 segmentos.
- Interfaz RS-232.
- Interfaz VGA.

El *core* de la tarjeta de desarrollo, como se mencionó antes, es el Spartan®-3E 1200, en el Anexo A se encuentra la hoja de datos de este FPGA, el diseño del sistema a implementar se realizó de acuerdo a las características de este dispositivo. Se desarrolló bajo un FPGA por las ventajas, antes mencionadas, que este nos ofrece sobre otros dispositivos.

En la Figura 5.3 se muestra el diagrama de bloques del sistema propuesto. El objetivo principal es validar la viabilidad de implementar un sistema embebido que efectué la Transformada de Fourier sobre una imagen, con el fin de que este pueda ser implementado (en un futuro) en un sistema de mayor complejidad (que realice el algoritmo completo para la reconstrucción del frente de onda), para aplicaciones en óptica adaptativa.

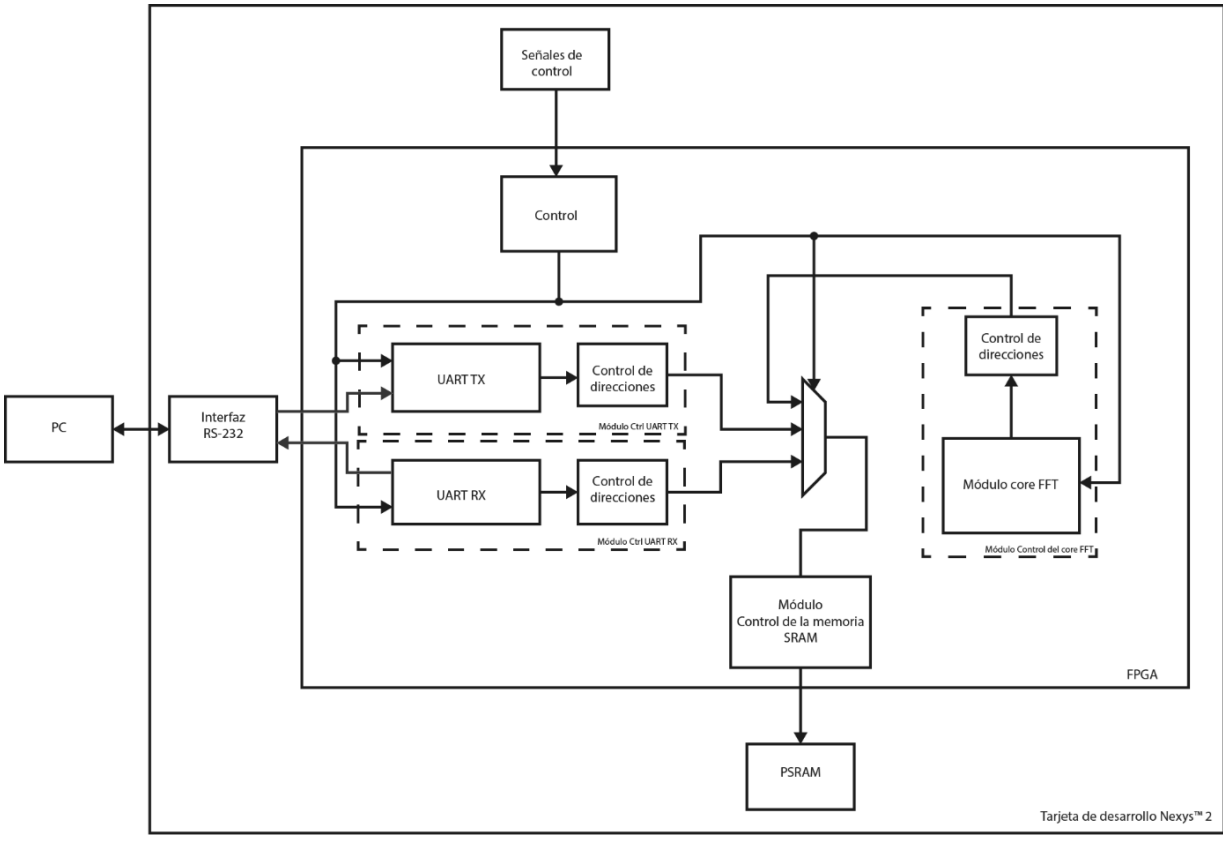


Figura 5.3._ Diagrama de bloques del sistema desarrollado.

El funcionamiento del sistema consiste, primero, en recibir los datos de una imagen a través de una interfaz de comunicación, debido a la gran cantidad de datos a recibir, estos deberán ser almacenados en un dispositivo externo. Una vez que los datos se hayan terminado de recibir y estén correctamente almacenados, deberán ser procesados, se efectuará sobre ellos la Transformada de

Fourier, para después volver a ser enviados a la PC a través de la interfaz de comunicación seleccionada.

El protocolo de comunicación elegido fue el RS-232, ya que la tarjeta de desarrollo cuenta con un puerto para implementarlo. El sistema de almacenamiento es a través de la memoria PSRAM que también se encuentra dentro de la tarjeta de desarrollo. El control general del sistema se llevará a cabo a través de los periféricos elementales: LEDs, interruptores y botones.

El sistema se compone de cinco módulos principales, los cuales están controlados por el *módulo de control principal del sistema*, el cual es el encargado de mantenerlos funcionando de manera ordenada.

1. **Control de la SRAM.** Módulo encargado de realizar el control de la memoria PSRAM de la tarjeta Nexys™2, para la escritura y lectura de datos.

2. **Ctrl UART RX**. Implementación de la UART del protocolo RS-232 para la recepción de datos. Este módulo es el encargado de llevar a cabo el correcto reensamble de los datos recibidos desde la PC. Tiene comunicación con el módulo de control de la memoria SRAM.

3. **Ctrl UART TX.** Implementación de la UART del protocolo RS-232 para la transmisión de datos. Se encarga de transmitir correctamente todos los datos hacia la Computadora. Tiene comunicación con el módulo de control de la memoria SRAM.

4. ***Core* FFT.** Este módulo efectúa el procesado de datos, a los cuales se les aplica la FFT.

5. **Control del *core* FFT.** Módulo encargado de controlar el core FFT. Además se encarga de la implementación del algoritmo para realizar la FFT sobre los datos de una imagen, ocupando para ello el core FFT.

6. **Control Principal del sistema**. Es el módulo que agrupa a los módulos anteriores y hace la conexión con los periféricos elementales. Su principal función es el de activar los módulos cuando sea necesario (cuando el usuario lo solicite).

Los módulos 2 y 3 forman parte del protocolo RS-232. Además, existen otros submódulos encargados de realizar el control entre dos o más módulos principales, los cuales se explicaran en las páginas siguientes. La descripción de cada módulo se realizó por medio de cartas ASM, las cuales se pueden consultar en el Anexo D.

Cabe mencionar que los datos con los que se va a trabajar dentro del FPGA son del tipo punto flotante de acuerdo a las especificaciones estipuladas en el estándar IEEE 754.

5.2.1 CONDICIÓN PARA QUE EL SISTEMA PUEDA SER EMPLEADO EN *SISTEMAS DE ÓPTICA ADAPTATIVA*

Los FPGAs, como ya se ha comentado, son útiles cuando se requiere procesar datos en un tiempo relativamente corto, es decir, aplicaciones que requieren altas velocidades de reloj. Los FPGAs trabajan con relojes superiores a los 50 MHz, lo que los hace útiles para ser empleados en óptica adaptativa, ya que aquí se trabaja con frecuencias superiores a los 2 Hz (Capitulo 1.2), es decir, se requiere que los datos sean procesados en un tiempo menor a los 500 ms. Por lo que en un principio el sistema a diseñar no tiene que superar el tiempo antes mencionado.

5.3 MÓDULO DE CONTROL DE LA SRAM

Las imágenes utilizadas en el procesamiento de la FFT tienen, en un principio, un tamaño de 512x512 pixeles con una profundidad de color de 8 bits. Al utilizar estas imágenes se hace necesario tener un espacio de almacenamiento externo al FPGA, debido a que la memoria con la que dispone el FPGA es insuficiente (136 Kbits de memoria distribuida y 504 Kbits de memoria SRAM de bloque). Esta memoria externa será utilizada tanto para guardar la imagen original recibida a través del puerto de comunicación serie, así como la imagen ya procesada. La tarjeta de desarrollo Nexys™2 cuenta con una memoria PSRAM[3] CMOS Micron CellularRAM MT45W8MW16BGX. Este dispositivo cuenta con las siguientes características principales.

- Modos de operación síncrona y asíncrona.
- Tiempo de acceso en modo asíncrono de 70 ns.
- Bus de 80 MHz en modo síncrono.

La RAM tiene un núcleo de 128 Mbits de DRAM, la cual está organizada en un arreglo de memoria de: 8 MBytes x 16 bits, lo que da un total de 16 MBytes de memoria[4], cada palabra de la memoria es de 16 bits, con un total de 2^{23} direcciones de memoria.

En la Figura 5.4 se muestra un diagrama de bloques funcional de este dispositivo. Dos registros de control accesibles al usuario definen la operación de la memoria: el registro de

[3] Una memoria PSRAM es una RAM hibrida, que incluye una función de refresco y circuitos de control de direcciones para que se comporte de manera similar una memoria RAM estática (SRAM) o a una RAM dinámica (DRAM).

[4] 16 MB = 16 x 1024x1024x8 = 134,217,728 bits, 128 Mb = 128x1024x1024 = 134,217,728 bits.

configuración del bus (BCR, del inglés *Bus Configuration Register*) define como la RAM interactúa con el bus de memoria del sistema, el registro de configuración de refresco (RCR, del inglés *Refresh Configuration Register*) se emplea para controlar como se lleva a cabo la actualización del arreglo de la DRAM. Estos registros son automáticamente cargados con valores predefinidos durante la inicialización de la memoria y pueden ser actualizados durante la operación normal de ésta. El control de estos registros se hace por medio del bloque CL (del inglés *Control Logic*), en el cual se conectan las señales de control del dispositivo. El bloque para la decodificación de las direcciones lógicas (ADL, del inglés *Address Decode Logic*) se encarga de llevar el control de acceso a la memoria para una determinada dirección. El registro DIDR (del inglés, *Device ID Register*) provee información sobre el fabricante del dispositivo, generación de la CellularRAM y la configuración especifica del dispositivo. Este dispositivo cuenta con un bus de direcciones de 23 bits, un bus bidireccional de 16 bits y 8 señales de control. El bus de datos está dividido en dos, para la salida/entrada de los datos más significativos y la salida/entrada de los bits menos significativos.

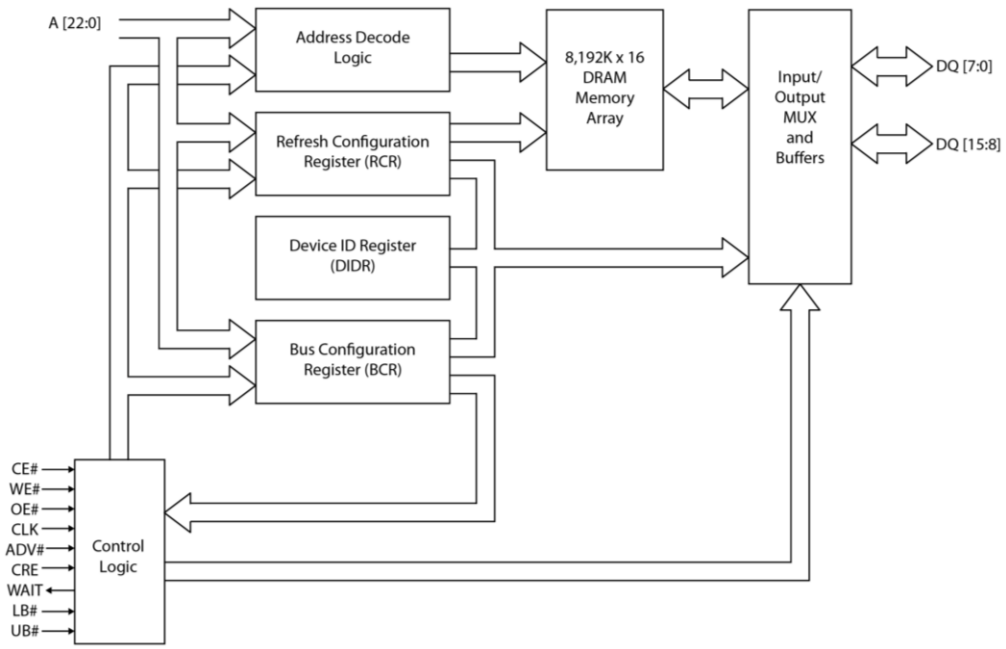


Figura 5.4._ Diagrama funcional de la memoria PSRAM.

La descripción de pines de la memoria CellularRAM MT45W8MW16BGX pueden ser consultados en la Tabla 5.1.

Tabla 5.1._ Descripción de los pines E/S para el dispositivo MT45W8MW16BGX.

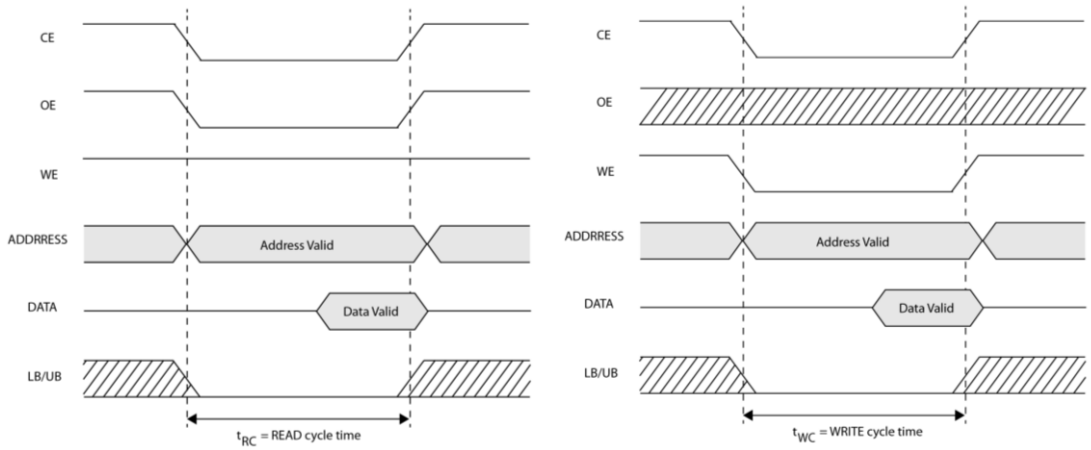
Señal	**Descripción**
A (23:0)	Bus de direcciones de 24 bits para la SRAM .
DQ (15:0)	Bus de datos de 16 bits de la SRAM.
OE	Señal de habilitación para la salida de datos.
WE	Señal de habilitación para la escritura de datos.
ADV	Dirección valida.
CLK	Señal de reloj para el modo síncrono.
UB	Señal de activación del byte superior para direccionamiento por byte.
LB	Señal de activación del byte inferior para direccionamiento por byte.
CE	Habilitación del chip.
WAIT	Señal de espera en modo asíncrono.
CRE	Habilitación de registro de control.

Se decidió utilizar el modo asíncrono, debido a su facilidad de implementación. En este modo únicamente se emplean 6 señales de control (CE, WE, OE, LB, UB, ADV), todas activas en bajo. La operación de estas señales se detalla en la Tabla 5.2.

Tabla 5.2._ Tabla de funcionamiento de la memoria PSRAM.

Operación	CE	WE	OE	LB	UB	DQ(byte inferior)	DQ(byte superior)
Desactivado	1	-	-	-	-	Z	Z
	0	1	1	-	-	Z	Z
	0	-	-	1	1	Z	Z
Lectura	0	1	0	0	1	Data out	Z
	0	1	0	1	0	Z	Data out
	0	1	0	0	0	Data out	Data out
Escritura	0	0	-	0	1	Data in	Z
	0	0	-	1	0	Z	Data in
	0	0	-	0	0	Data in	Data in

En la Figura 5.5 se muestra el diagrama de tiempo básico para la escritura y lectura de la RAM. La operación de lectura es inicializada poniendo las señales CE, OE y LB/UB en un nivel bajo, mientras mantenemos WE en un nivel alto. El dato valido saldrá en el bus E/S después de un determinado tiempo (^{t}RC, del inglés *Read Cycle time*). Para la operación de escritura ponemos las señales CE, WE, LB y UB en un estado lógico bajo, durante este proceso el estado en el que se encuentre la señal OE no importa. Para que el dato se escriba correctamente en la memoria RAM debe transcurrir un determinado tiempo (^{t}WC, del inglés Write Cycle time).



a) b)

Figura 5.5._ Diagramas de tiempo simplificados de la memoria PSRAM. a) Operación de lectura. b) Operación de escritura.

En ambas operaciones, una dirección de memoria valida se carga al poner en un cero lógico la señal ADV. Para el modo asíncrono, tanto de lectura como de escritura hacia la memoria SRAM se tiene un tiempo de acceso de 70 ns (^{t}RC, ^{t}WC). Para asegurar el correcto funcionamiento del módulo, debemos verificar que se cumplan correctamente los requerimientos de tiempo.

El reloj que integra la tarjeta de desarrollo Nexys™2 tiene una frecuencia de 50 MHz, por lo que tenemos un periodo de 20 ns. Tomando en cuenta este tiempo se desarrolló el algoritmo de escritura y lectura de la SRAM. Es posible aumentar la frecuencia de trabajo (20 ns) por medio de los bloques DCM con los que cuenta el FPGA, lo cual se demuestra en el anexo C, sin embargo, debido a que el acceso a la SRAM es fijo, no tendría ningún beneficio aumentar la frecuencia.

En la Figura 5.6 se aprecia el símbolo del módulo que se implementó para el control de la RAM y en la Tabla 5.3 se describen los pines correspondientes al módulo.

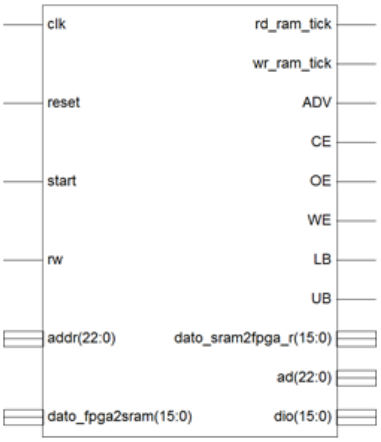


Figura 5.6._ Símbolo del módulo de control de la SRAM.

Tabla 5.3._ Descripción de los pines de entrada y salida del módulo de control de la SRAM

Señal	Ent.	Sal.	Descripción
clk	✔		Señal de reloj para el control del proceso.
reset	✔		Señal de inicialización del módulo de control SRAM.
start	✔		Señal de inicialización del proceso de lectura o escritura.
rw	✔		Señal de configuración para lectura o escritura de la RAM. 1 – Lectura, 0 – Escritura.
addr(22:0)	✔		Arreglo de señales de 23 bits para indicar la dirección donde se va a escribir el dato.
dato_fpga2sram(15:0)	✔		Dato a escribir en la memoria RAM.
dato_sram2fpga_r(15:0)		✔	Dato leído de la memoria RAM.
ADV, OE, CE, WE, LB, UB		✔	Señales de control de la RAM.
ad (22:0)		✔	Arreglo de 23 bits de la dirección de lectura o escritura de la RAM.
dio (15:0)	✔	✔	Bus de datos que se han leído o se van a escribir en la RAM.
wr_ram_tick		✔	Señal que indica cuando se terminó de escribir un dato.
rd_ram_tick		✔	Señal que indica cuando se terminó de leer un dato.

Cuando los demás módulos quieren tener acceso a la SRAM, tienen que comunicarse con el “módulo de control de la SRAM”, pero solo se permite el acceso uno a la vez.

En las Tablas 5.4 y 5.5 se muestran la descripción de los tiempos de procesamiento obtenidos a partir de los Diagramas de tiempo del módulo de control de la SRAM.

Tabla 5.4._ Descripción de los tiempos para el proceso de lectura de la SRAM.

Acción	Tiempo [ns]
Tiempo total del proceso.	140
Tiempo de duración del proceso hasta que la señal *rd_ram_tick* se pone en alto.	120
Tiempo que duran las señales de control en un nivel bajo.	100
Tiempo que transcurre desde que se activa la señal *start* hasta que las señales de control se ponen en un nivel bajo.	40
Tiempo que transcurre desde que se activa la señal *start* hasta que el dato leído se coloca en el bus *dato_sram2fpga_r(15:0)*.	120

Tabla 5.5._ Descripción de los tiempos para el proceso de escritura de la SRAM.

Acción	Tiempo [ns]
Tiempo total del proceso.	140
Tiempo de duración del proceso hasta que la señal *wr_ram_tick* se pone en alto	120
Tiempo que duran las señales de control en un nivel bajo.	100
Tiempo que transcurre desde que las señales de control se ponen en un nivel bajo hasta que el dato a guardar se coloca en el bus *dio(15:0)*.	60
Tiempo que transcurre desde que se activa la señal *start* hasta que las señales de control se ponen en un nivel bajo.	40
Tiempo que permanece el dato a guardar en el bus *dio(15:0)*.	40
Tiempo que transcurre desde que se activa la señal *start* hasta que el dato a guardar se coloca en el bus *dio(15:0)*.	100

5.4 UART RS-232

El protocolo escogido para realizar la transmisión de datos (a través de la PC y la tarjeta FPGA) es el RS-232, ya que es fácil de implementar, además de que la tarjeta de desarrollo Nexys™2 cuenta con un puerto RS-232. La UART (del inglés *Universal Asynchronous Receiver-Transmitter*) es la encargada de decidir cómo se van a transmitir los datos, hace la conversión de datos en paralelo a serie y viceversa. LA UART incluye un transmisor (UART TX) y un receptor (UART RX). El transmisor es esencialmente un registro de corrimiento, el cual carga datos en

paralelo y después va haciendo un corrimiento de bit a bit a una determinada velocidad. El receptor, por otro lado, recibe los datos bit a bit, para después convertirlo a un dato en paralelo. En la Figura 5.7 se muestra la estructura de un Byte transmitido.

Figura 5.7._ Transmisión de un Byte.

Mientras no hay transmisión, la señal de salida a través del puerto serie siempre mantiene un nivel lógico alto, “1”, este estado se conoce comúnmente como *idle*. La transmisión empieza con el bit de inicio, el cual es un “0” lógico, seguido por los bits de datos y un bit de paridad opcional, terminando con el bit de parada, el cual es un “1” lógico. Dado que la transmisión es asíncrona no se necesita que una señal de reloj sea transmitida a través del puerto serie.
La UART diseñada en el FPGA tiene las siguientes características:

- Baud rate del transmisor: 57600 bps.
- Baud rate del receptor: 57600 bps.
- Número de bits de datos: 8 bits
- Bit de paridad: no.
- Número de bits de parada: 1

5.4.1 MODULO CTRL UART RX

El módulo ctrl UART RX tiene la función de recibir los datos que provienen de la interfaz RS-232, estos datos llegan en tramas de 8 bits (punto fijo). Como se mencionó en la Sección 5.1 se va a trabajar con datos en punto flotante, por lo tanto los datos recibidos deben ser convertidos de acuerdo al estándar IEEE-754, una vez convertidos, los direccionamos a su correspondiente ubicación en la memoria SRAM. Este proceso se ilustra en el diagrama de bloques de la Figura 5.8.

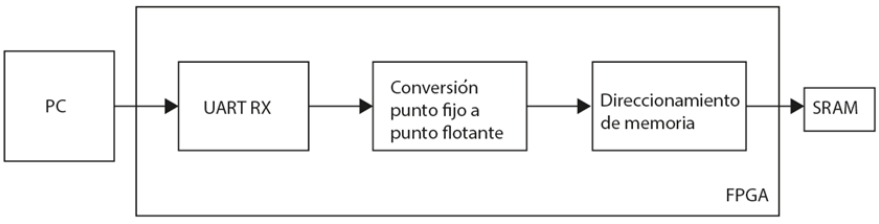
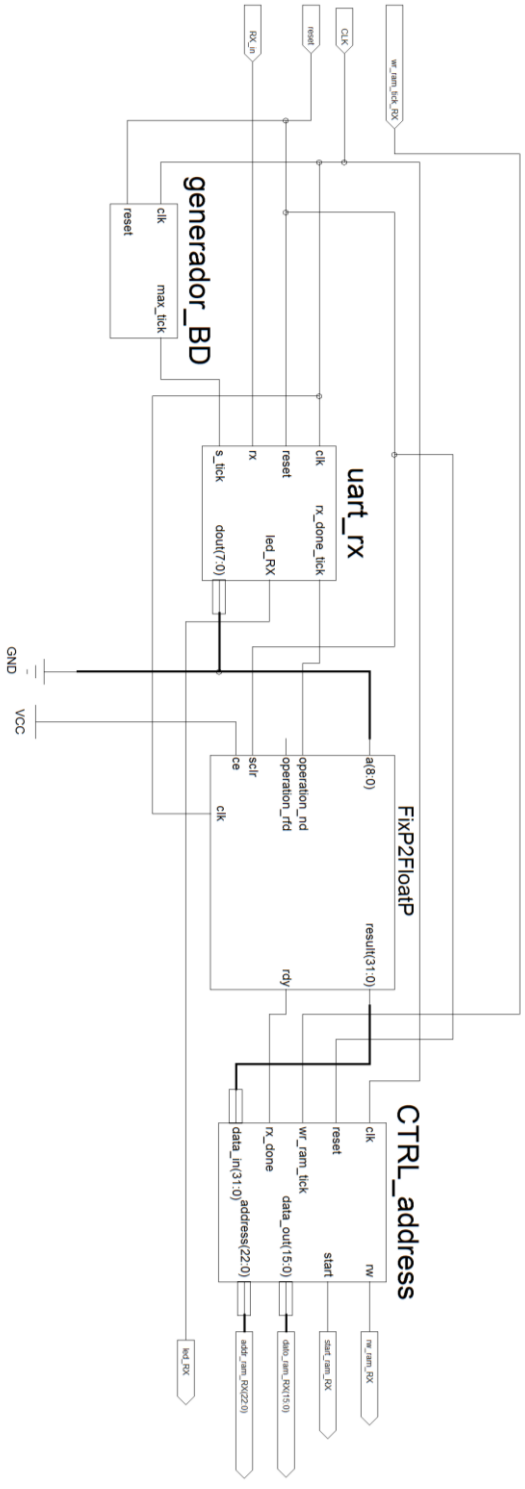
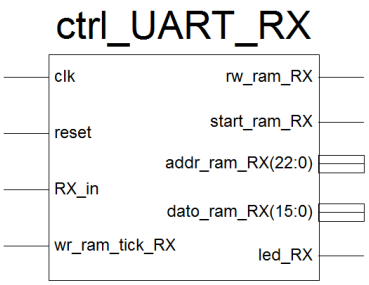


Figura 5.8._ Diagrama de bloques del proceso de comunicación del módulo UART RX – SRAM.

Por lo tanto el módulo ctrl UART RX se compone de 3 submódulos: UART RX, conversión de datos y direccionamiento de memoria. El diagrama esquemático del módulo completo se ilustra en la Figura 5.9.



a)



b)

Figura 5.9._ a) Integración del módulo UART RX, conversión de punto fijo a punto flotante y control de direccionamiento de la SRAM. b) Representación esquemática de la integración del módulo (ctrl_UART_RX)

En la Tabla 5.6 se describen las asignaciones de cada pin del módulo mostrado en la Figura 5.9b.

Tabla 5.6._ Asignación de pines para el módulo UART RX - SRAM (ctrl_UART_RX).

Señal	Ent.	Sal.	Descripción
clk	✔		Reloj.
reset	✔		Reset.
RX_in	✔		Entrada serie de los datos.
wr_ram_tick_RX	✔		Señal de entrada que indica cuando se terminó de realizar la escritura en la SRAM
rw_ram_RX		✔	Señal para indicarle a la SRAM si se va a a iniciar una operación de escritura o lectura.
start_ram_RX		✔	Señal para iniciar el proceso de escritura de la SRAM.
addr_ram_RX		✔	Dirección de la memoria SRAM.
dato_ram_RX		✔	Dato que se va a almacenar en la SRAM.
led_RX		✔	Señal que se mantiene en alto mientras se realiza la operación de almacenamiento de la SRAM.

Los datos, en su totalidad, se tardan teóricamente 45.51 s. en ser transmitidos, $\frac{1}{57600} * 10 * (512 * 512) = 45.1\ s.$ Un dato de 8 bits se tarda $\frac{1}{57600} * 10 = 176\ \mu s$. La imagen que se almacena en la SRAM ocupa 1 MB.

Para un solo dato transmitido de la PC a la FPGA (8 bits), el tiempo que le toma al sistema llevar a cabo el proceso, desde que se inicia la operación de conversión de datos hasta que se terminan de guardar los datos, es de 460 ns (tiempo obtenido a partir de simulaciones de diagramas de tiempo proporcionados por la herramienta ISE®).

5.4.1.1 SUBMÓDULO UART RX

El módulo está desarrollado utilizando el esquema de sobremuestreo, el cual nos ayuda a estimar el punto medio de los bits transmitidos y luego en ese punto recuperar el bit transmitido. La frecuencia de muestreo más utilizada para realizar el sobremuestreo es 16 veces el baud rate, lo que significa que cada bit de datos es muestreado 16 veces. El sobremuestreo trabaja de la siguiente forma, asumiendo que la comunicación usa N bits de datos y M bits de parada:

1. Se espera hasta que se reciba un “0” lógico, el cual es el bit de inicio, este bit indica que se van a empezar a recibir los datos, en este momento se inicia el contador que lleva el control del muestreo.
2. Cuando el contador llega hasta 7 se alcanza la mitad del bit de inicio, en este momento se pone de nuevo el contador en cero y se reinicia la cuenta.
3. Cuando el contador alcanza el 15, se alcanza la mitad del primer bit de datos, en este momento se recupera el primer bit de los datos y se va poniendo este bit en un registro de corrimiento. Se reinicia el contador.
4. Se repite el paso anterior N-1 veces para recuperar todos los bits de datos.
5. Si hay bit de paridad se repite nuevamente el paso 3 para recuperar el bit de paridad.
6. Nuevamente se repite el paso 3 M veces más para obtener los demás bits de parada.

El módulo receptor para la UART está compuesto por dos componentes:

1) Generador de baud rate
2) Receptor

El generador de baud rate produce la señal de muestreo, la cual es 16 veces el baud rate deseado de la UART. Para el caso de un baud rate de 57600 bps la tasa de muestreo es de $57600x16 = 921600$ pulsos por segundo, ya que la FPGA tiene un reloj de 50 MHz necesitamos un contador que ponga un nivel lógico alto cada $\frac{50\ MHz}{921600} = 54$ ciclos de reloj. En la Figura 5.10 se aprecia el símbolo del módulo del generador de baud rate.

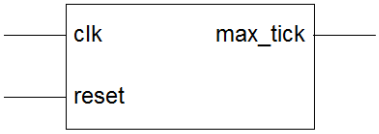


Figura 5.10._ Módulo del generador de baud rate para el receptor de la UART.

El módulo de la Figura 5.10 tiene dos entradas: *clk*, el cual es el reloj de 50 MHz, y *reset*, el cual reinicializa el contador de los pulsos de reloj. Cuenta con una única salida, *max_tick*, la cual se pone en un nivel lógico alto (durante un ciclo de reloj) cada vez que el contador alcanza el valor de 54.

El componente receptor se programó de acurdo al algoritmo descrito anteriormente. En la Figura 5.11 se muestra el diagrama de bloques de este componente y en la tabla 5.7 la descripción de sus pines.

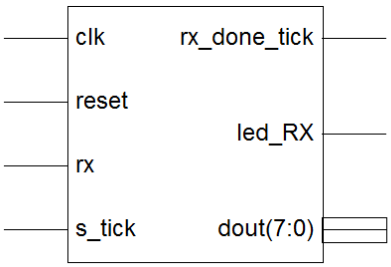


Figura 5.11._ componente de recepción para el módulo RX de la UART.

Tabla 5.7._ Asignación de pines para el módulo RX .

Señal	Ent.	Sal.	Descripción
clk	✔		Reloj.
reset	✔		Reset.
rx	✔		Entrada serie de los datos.
s_tick	✔		Pulso proveniente del generador del baud rate.
rx_done		✔	Señal que se pone en un alto lógico durante un periodo del ciclo de reloj indicando que el dato esta ya ubicado en el bus *dout.*
led_ RX		✔	Señal que se mantiene en un alto lógico mientras se efectúa la operación de recepción
dout(7:0)		✔	Salida de datos en paralelo.

La integración de los dos componentes antes descritos constituye el módulo de la UART RX. Dicha integración se ilustra en la Figura 5.12.

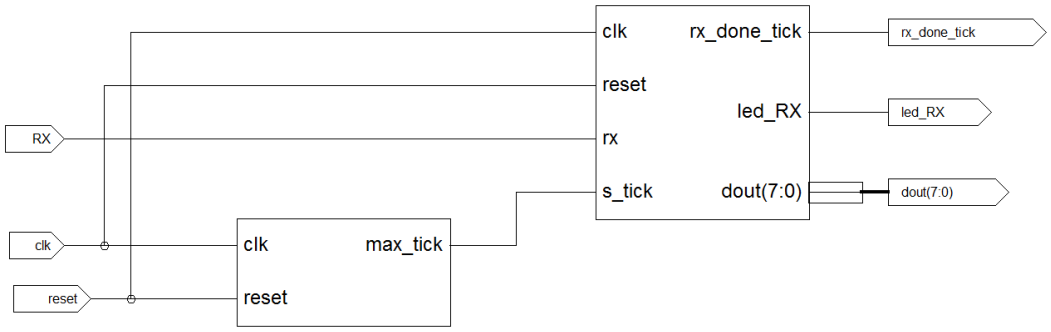


Figura 5.12._ Módulo UART RX.

5.4.1.2 SUBMÓDULO DE CONVERSIÓN A PUNTO FLOTANTE

Para implementar el módulo de conversión de punto fijo a punto flotante se empleó el ipcore de Xilinx®, LogiCORE IP FLOATING-POINT. Este ipcore provee a los diseñadores los medios necesarios para realizar aritmética en punto flotante en un dispositivo FPGA. Este núcleo se puede personalizar de acuerdo al tipo de operación que se necesite y a los datos con los que se va a trabajar. Las operaciones que soporta son:

- Multiplicación.
- Adición/Substracción.
- División.
- Raíz cuadrada
- Comparación.
- Conversión de punto flotante a punto fijo.
- Conversión de punto fijo a punto flotante.
- Conversión entre los diferentes estándares de punto flotante.

Este módulo cumple con el estándar IEEE-754 y es compatible con las siguientes tarjetas FPGAs de Xilinx®: Kintex™-7, Virtex®-7, Virtex®-6, Virtex®-5, Virtex®-4, Spartan®-6, Spartan®-3/XA, Spartan®-3E/XA, Spartan®-3A/AN/3A DSP/XA. Además, este *core* nos permite trabajar con un amplio número de operaciones, las cuales se especifican en el momento de crear el *core*, y cada una de estas operaciones tiene una interfaz común, la cual se muestra en la Figura 5.13.

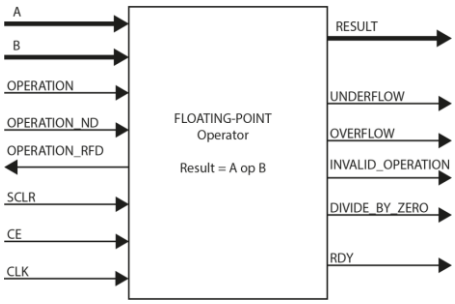


Figura 5.13._ Diagrama de bloques genérico del core de aritmética en punto flotante.

Los puertos empleados por el *core*, ilustrados en la Figura 5.13, se muestran más a detalle en la Tabla 5.8.

Tabla 5.8._ Descripción de los pines del core: LogiCORE IP FLOATING-POINT.

Señal	Ent.	Sal.	Descripción
A	✔		Operando A
B	✔		Operando B. Sólo presente en una operación binaria.
Operation	✔		Especifica la operación a ser realizada. Únicamente está presente cuando se selecciona la operación Adición/Substracción o alguna operación de comparación.
Operation_ND	✔		Debe estar en alto para indicar que el operando A, operando B y la operación son datos válidos.
Operation_RFD		✔	Se pone en alto cundo el *core* está listo para aceptar nuevos operandos.
SCLR	✔		Reset síncrono.
CE	✔		Clock enable.
CLK	✔		Reloj.
Result		✔	Resultado de la operación.
Underflow		✔	Se pone en alto cuando ocurre un subdesbordamiento.
Overflow		✔	Se ponen en un nivel alto cuando ocurre un desbordamiento de datos.
Invalid_OP		✔	Operación invalida, se mantiene en un alto lógico mientras haya una operación inválida.
Divide_by_zero		✔	Se pone en una lato lógico cuando ocurre una división por cero.
RDY		✔	Señal que se pone en alto cuando se tiene un resultado valido.

La operación a realizar por el módulo "conversión a punto flotante" es el de convertir los datos en 8 bits que se reciben a punto flotante (precisión de 32 bits). El diagrama esquemático de este módulo se presenta en la Figura 5.14, cuyos pines son los mismos que se explicaron en la Tabla 5.8.

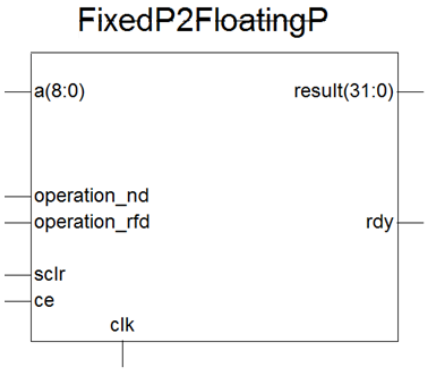


Figura 5.14._ Módulo de conversión de punto fijo a punto flotante (precisión de 32 bits).

De acuerdo a la Figura 5.19, el número de bits de entrada es 9 (a [8:0]), esto es debido a que la representación en punto fijo requiere que el bit más significativo sea el signo (0 – positivo, 1 - negativo) del dato, ya que todos los valores recibidos siempre son positivos, este bit siempre va tener un valor de cero y los 8 bits restantes serán los bits recibidos a través del protocolo de comunicación RS-232. La máxima latencia presentada en el *core* es de 6 ciclos de reloj (120 ns).

5.4.1.3 SUBMÓDULO DE DIRECCIONAMIENTO DE MEMORIA

Una vez que se convierten los datos a punto flotante, estos pasan al módulo de direccionamiento de memoria, el cual se encarga de manejar las direcciones de la SRAM para que los datos sean guardados de forma correcta.

La forma en que se envían los datos a través del protocolo y se guardan los datos en la SRAM es la siguiente:

- Se cuenta, por ejemplo, con una imagen de 4x4 elementos (representando un pixel). Cada pixel con un nivel de intensidad en particular, según se muestra en la matriz "M" siguiente:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

- Se envían primero los datos del primer renglón, de izquierda a derecha, luego se envía el segundo renglón y así sucesivamente.
- Cada dato está formado por 8 bits, estos son transformados a una notación de 32 bits (punto flotante), debido a que cada palabra de la memoria SRAM es de 16 bits, los datos deben ser descompuestos en dos partes para así ser almacenados correctamente.
- Los 16 bits más significativos de los datos, en punto flotante, se almacenan primero en una localidad de la SRAM, después, en la siguiente localidad de memoria se almacenan los bits menos significativos.
- Por lo tanto, para el primer dato M(0,0), los primeros 16 bits se almacenan en la localidad 0 y los 16 bits restantes se almacenan en la siguiente localidad de memoria (ver Tabla 5.9).

Tabla 5.9._ Almacenamiento de los datos en la SRAM

localidad de memoria	dato	Localidad de memoria	dato
0	1	16	9
1		17	
2	2	18	10
3		19	
4	3	20	11
5		21	
6	4	22	12
7		23	
8	5	24	13
9		25	
10	6	26	14
11		27	
12	7	28	15
13		29	
14	8	30	16
15		31	

- Una imagen de 512x512 ocupa 524,288 palabras de 16 bits para almacenarse correctamente, lo que da un total de 8,388,608 bits de memoria, a lo que es equivalente 1 MB de memoria SRAM. Los datos de la imagen se almacenan a partir de la localidad 0 hasta la 524,287.

5.4.2 MODULO CTRL UART TX

Este módulo tiene como fin llevar el control para la lectura de los datos de la SRAM, para posteriormente ser enviados a través del bus de comunicación RS-232. El proceso de funcionamiento se ilustra en la Figura 5.15.

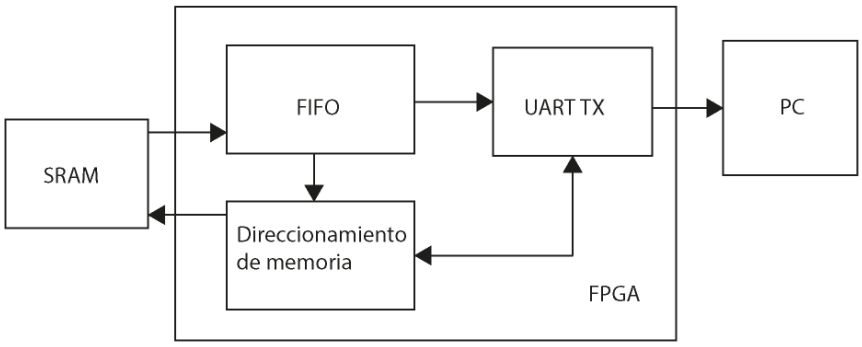
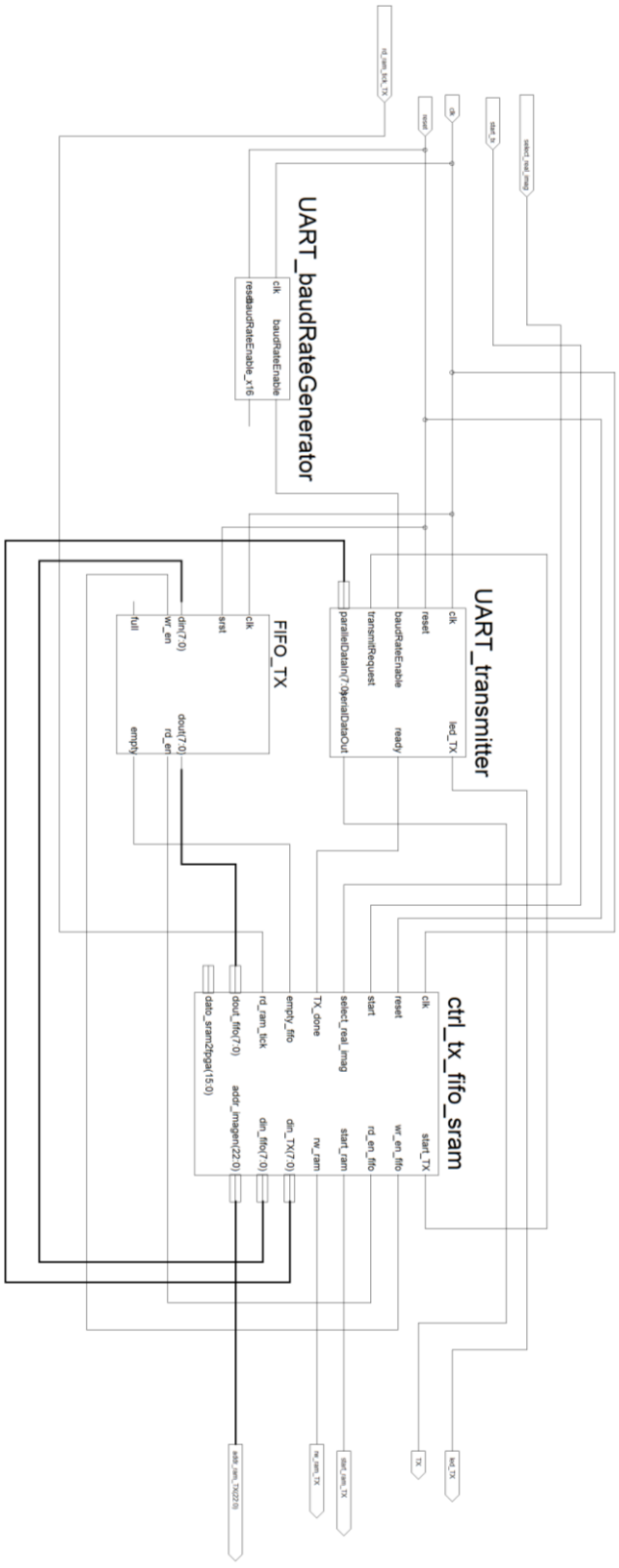


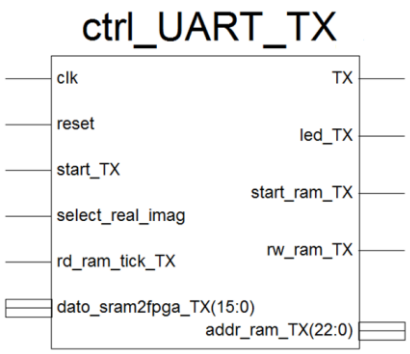
Figura 5.15._ Diagrama de bloques del funcionamiento del módulo de comunicación SRAM – UART TX.

De acuerdo a la Figura 5.15, tenemos un submódulo de direccionamiento de la SRAM, el cual lleva el control de las direcciones de la SRAM para una correcta lectura de datos. Una vez que se ha leído un dato, este pasa a guardarse en una pila del tipo FIFO (del inglés, First In First Out). Un dato completo de 32 bits está compuesto de 4 Bytes, por lo tanto, la FIFO tiene una capacidad de 4 Bytes. La FIFO envía una señal (cuando está vacía) al módulo de control de direccionamiento para que este active la lectura de datos de la SRAM. Una vez que la FIFO se ha llenado, empezamos a transmitir estos Bytes (por medio del protocolo RS232). Cuando terminan de transmitirse los datos, el módulo la FIFO, envía una señal al módulo de control de direccionamiento, indicándole que ya es posible continuar con la nueva lectura de datos de la SRAM. Este proceso se realiza hasta que se terminan de enviar todos los datos de la FFT de la imagen que se procesó.

El diagrama esquemático de la integración de los 3 submódulos ocupados (UART TX, FIFO y direccionamiento de memoria) para crear el módulo ctrl UART TX se muestra en la Figura 5.16.


UART_baudRateGenerator
clk
reset
baudRateEnable
baudRateEnable_x16
UART_transmitter
clk
reset
baudRateEnable
transmitRequest
parallelDataIn(7:0)
serialDataOut
ready
led_TX
FIFO_TX
clk
srst
din(7:0)
wr_en
full
dout(7:0)
rd_en
empty
ctrl_tx_fifo_sram
clk
reset
start
select_real_imag
TX_done
empty_fifo
rd_ram_tick
dout_fifo(7:0)
data_sram2fpga(15:0)
start_TX
wr_en_fifo
rd_en_fifo
start_ram
rw_ram
din_TX(7:0)
din_fifo(7:0)
addr_imagen(22:0)

a)



b)

Figura 5.16._ a) Integración del módulo UART TX, pila y control de direccionamiento de la SRAM. b) Representación esquemática de la integración del módulo (ctrl_UART_TX_ram).

En la Tabla 5.10 se describen las asignaciones de cada pin del módulo mostrado en la Figura 5.16b.

Tabla 5.10._ Asignación de pines para el módulo UART RX - SRAM (ctrl_UART_TX_ram).

Señal	Ent.	Sal.	Descripción
clk	✔		Reloj.
reset	✔		Reset.
start_TX	✔		Inicia la transmisión de datos.
select_real_imag	✔		Selecciona que datos se van a transmitir, si los reales (0) o los imaginarios (1).
rd_ram_tick_TX	✔		Señal que se pone en alto durante un ciclo de reloj cuando el proceso de lectura de la SRAM finaliza.
Dato_sram2fgpa_TX(15:0)	✔		Dato leído de la SRAM.
addr_ram_TX(22:0)		✔	Dirección de la memoria SRAM.
rw_ram_RX		✔	Si es 1 realizamos un lectura a la SRAM, si es 0 realizamos un proceso de escritura.
led_TX		✔	Señal que se mantiene en alto mientras se realiza la operación de almacenamiento de la SRAM
start_ram_TX		✔	Preparamos a la SRAM para realizar un proceso de lectura o escritura.
TX		✔	Dato de salida de la interfaz RS-232.

El tiempo total de transmisión de datos a una velocidad de 57600 Baudios es de aproximadamente 13 minutos, considerando los retardos agregados en el funcionamiento del módulo de direccionamiento de memoria, este valor se obtuvo experimentalmente, como se puede observar en el capítulo 6.

5.4.2.1 SUBMÓDULO UART TX

La estructura del submódulo transmisor de la UART es muy similar a la estructura del submódulo receptor (UART RX), ya que de igual manera consiste en un generador de baud rate y un componente transmisor. El generador de baud rate es esencialmente el mismo que se ocupa en el módulo receptor, por lo tanto el símbolo del componente también se ilustra en la Figura 5.10.

El componente transmisor es esencialmente un registro de corrimiento que va desplazando los bits a un determinado baud rate.

La integración de los dos componentes antes descritos constituyen el módulo de la UART TX, esta integración se ilustra en la Figura 5.17. Los puertos con los que cuenta el componente generador del baud rate del transmisor son los mismos que el del receptor. El componente transmisor cuenta con 5 puertos de entrada y 3 puertos de salida. En el puerto *clk* se tiene la entrada del reloj de la FPGA, el puerto *reset* nos sirve para reiniciar el módulo, la entrada de los pulsos del generador de baud rate se da en el puerto *baudRateEnable*, la solicitud para iniciar una nueva transmisión se ingresa por el puerto *transmitRequest* y los datos a transmitir se ingresan por medio del bus *parallelDataIn*. El puerto *ready* manda un pulso con una duración de un periodo del reloj de la FPGA una vez que se ha completado el proceso de transmisión, los datos en serie son transmitidos a través del bus *SerialDataOut* y la señal que obtenemos del puerto *led_TX* se mantiene en un alto lógico durante el proceso de transmisión de datos.

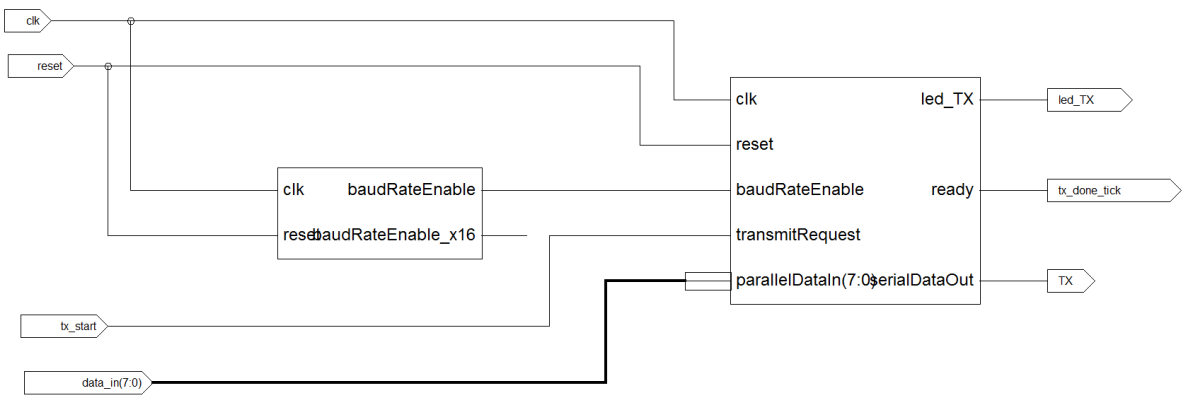


Figura 5.17._ Módulo UART TX.

El tiempo teórico que el protocolo de comunicación RS-232 tarda en enviar un Byte es de $\frac{1}{57600} x10 = 0.17361\ [ms]$.

5.4.2.2 FIFO

Para implementar la pila se utilizó el ipcore LogicCORE™ IP FIFO de XIlinx®. Este *core* proporciona una optimizada solución para la implementación de pilas del tipo FIFO en las FPGAs de Xilinx® utilizando un mínimo de recursos y funcionando de manera óptima. El *core* nos permite configurar aspectos tales como, el ancho de palabra (de 1 hasta 1024), profundidad (hasta 4,194,304 palabras), banderas de estado y tipo de memoria con la que se va a implementar. El *core* puede ser configurado para utilizar un reloj común o relojes independientes para las operaciones de lectura y escritura. Al utilizar relojes independientes, el *core* nos permite trabajar con diferentes dominios de tiempo para el proceso de escritura y lectura, si se utiliza un único reloj ambos procesos funcionan en el mismo dominio de tiempo. Este *core* nos permite crear la pila con los dos tipos de memoria con los que se cuenta en la FPGA, la memoria distribuida y la memoria de bloques, dependiendo de nuestras necesidades se ocupa una u otra. Este *core* además implementa una característica útil, llamada *First-Word Fall-Through (FWFT)*, la cual nos permite verificar el último dato que se encuentre en la pila sin activar la operación de lectura. En una configuración normal sin la característica FWFT es necesario activar la operación de lectura para acceder a los datos, lo que nos conlleva a perder un ciclo de reloj para tener los datos listos en el bus de salida de la pila (en el caso de querer acceder solamente al último dato guardado). Al configurar una pila como FWFT podemos ahorrarnos este ciclo de reloj de activación de la pila y acceder de inmediato al último dato almacenado, lo que genera una latencia de 0 s para el modo FWFT. El módulo generado para la pila se muestra en la Figura 5.18 y su descripción de pines se muestra en la Tabla 5.11.

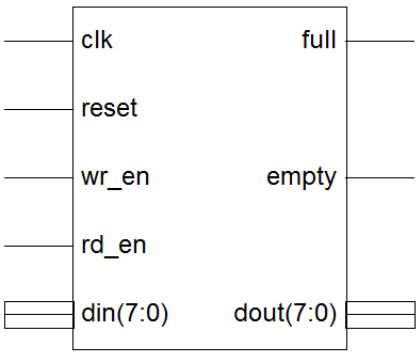


Figura 5.18._ Diagrama esquemático de la pila tipo FIFO, utilizando el core LogicCORETM IP FIFO.

Tabla 5.11._ Asignación de los pines de la pila tipo FIFO.

Señal	Ent.	Sal.	Descripción
clk	✔		Reloj.
reset	✔		Reset.
wr_en	✔		Activa la escritura de la pila.
rd_en	✔		Activa la lectura de la pila.
din (7:0)	✔		Datos de entrada de la pila, son los datos que se van a almacenar en la pila.
full		✔	Señal que indica que la pila se encuentra llena.
empty		✔	Señal que indica que la pila se encuentra vacía.
dout (7:0)		✔	Datos de salida de la pila, se ubican en el bus después de haber activado el proceso de lectura.

5.4.2.3 SUBMÓDULO DE DIRECCIONAMIENTO DE MEMORIA

Tanto el módulo de la UART TX, como la pila están controlados por el módulo de direccionamiento de memoria. La función básica de este módulo es realizar la conexión con el módulo de memoria de la SRAM.

5.5 MÓDULO FFT (CORE FFT)

El módulo FFT es el encargado de efectuar el algoritmo de la FFT (Fast Fourier Transform). Para implementar este componente se tenían dos opciones principales: diseñarlo desde cero o adaptar un módulo FFT ya desarrollado. La principal ventaja que se tiene al desarrollarlo desde cero, es que se cuenta con una mayor libertad en su implementación, es decir se implementa el mejor algoritmo que para nosotros es el más óptimo para un determinado número de datos y durante su desarrollo se va optimizando hasta tener la implementación deseada, programada a gusto del diseñador. Sin embargo, también se presentan desventajas, como por ejemplo, el tiempo que conlleva tener una implementación óptima, ya que se tiene que ir validando paso a paso con el uso de las simulaciones de tiempo. Y Al ser un nuevo módulo, este debe ser completamente validado

para un correcto funcionamiento, además de que se tiene que hacer un óptimo empleo de los recursos de la FPGA.

Las diferentes empresas que diseñan FPGAs tienen ya sus propias implementaciones de diferentes módulos para procesamiento de señales, es decir implementan sus propios DSP para el uso de terceros, estas implementaciones son conocidas como *IP cores*. La ventaja de utilizar estos *cores* es que al ser diseñadas para una línea de tarjetas en específico se tiene contemplado el uso óptimo de recursos de la FPGA, además de que nos aseguran un correcto funcionamiento de los mismos.

Se trabajó con una tarjeta de Xilinx®, por lo tanto el *core* ocupado únicamente servirá para su línea de FPGAs. Este *core* es el FFT LogicCore™. Las principales características que nos ofrece este *core* son las siguientes:

- Implementación de la transformada directa y transformada inversa.
- Longitud de la transformada de $N = 2^m, m = 3 - 16$.
- Precisión de los datos de entrada de 8 – 34 bits.
- Precisión del factor de fase de 8 – 34 bits.
- Tipo de aritmética usada
    - Punto Fijo sin escalamiento
    - Punto fijo escalado.
    - Punto flotante.
- Interfaz para utilizar datos en punto fijo o en punto flotante.
- Posibilidad de elegir entre truncar o redondear los datos después de una operación.
- Posibilidad entre elegir utilizar la memoria RAM en bloques o la distribuida para el almacenamiento de datos y el factor de fase.
- Los datos a la salida se pueden mostrar tanto en orden natural o inverso.
- 4 tipos de arquitectura: Pipelined, Radix-4, Radix-2 y Radix-2 Lite.

El *core* de la FFT calcula la DFT directa o DFT inversa de una serie de N puntos, donde N puede ser de longitud $2^m, m = 3 - 16$. Para entradas en punto fijo, los datos son un vector de N valores complejos representados en b_x bits en complemento a dos, es decir, b_x bits para cada valor real e imaginario, donde b_x puede ser de longitud de 8 a 34 bits. Para el factor de fase b_w también puede ser de 8 a 34 bits.

Para entradas en punto flotante, los datos son un vector N de valores complejos representados por 32 bits de acuerdo al estándar IEEE 754 para precisión simple. El factor de fase es representado con valores en punto fijo de 24 o 25 bits.

La totalidad de la memoria ocupada para la implementación del *core* se encuentra dentro de la FPGA, y ésta puede ser la memoria RAM de bloques o la memoria RAM distribuida. La memoria (cualquiera de los dos tipos que se haya elegido) es utilizada para guardar los datos de entrada, los datos resultantes de las operaciones y los datos de los factores de fase. Los N elementos de los datos de salida están representados utilizando b_y bits para cada componente real e imaginario. Los datos de entrada deben ser introducidos en un orden natural y los datos de salida pueden ser representados en orden natural o con los bits invertidos.

El *IP core* de la FFT emplea los algoritmos Radix-4 o Radix-2 para calcular la transformada rápida de Fourier. Para la arquitectura de Pipeline se emplea el método de decimacion en frecuencia, mientras que para las arquitecturas Radix-2 Lite, Radix-2 y Radix-4 se emplea el método de decimacion en el tiempo. Cuando se utiliza el algoritmo Radix-4, la FFT de N puntos consiste en $log_4 N$ etapas, con cada etapa conteniendo N/4 Radix-4 mariposas. Longitudes de datos que nos son potencia de 4 necesitan una etapa adicional que implementa el algoritmo de Radix-2. LA FFT de N puntos para el algoritmo Radix-2 realiza una descomposición de $log_2 N$ etapas, con cada etapa conteniendo N/2 Radix-2 mariposas.

5.5.1 IMPLEMENTACIÓN DEL CORE FFT

De las cuatro arquitecturas que es posible implementar en el ip core FFT LogicCore™ se debe seleccionar la mejor opción para ser implementada en la tarjeta con la que se cuenta, Spartan®-3E 1200, es decir se debe escoger aquella arquitectura que se pueda implementar en la FPGA con los recursos con los que esta cuenta, sin sobrepasar la cantidad de los mismos. Para escoger la mejor opción, se implementaron diversos módulos del *core,* simulándolos en el entorno ISE®, con el fin de hacer una comparación de recursos empleados. Las arquitecturas implementadas fueron:

- Radix-4.
- Radix-2.
- Radix-2 Lite.

Las características de los módulos de las arquitecturas a implementar son similares:

- Formato de datos en punto flotante.
- Orden de los datos de salida: Orden natural.
- Los datos se almacenan en la RAM de bloques.
- Los datos de los factores de fase se almacenan en la memoria RAM distribuida, y tiene un ancho de palabra de 25 bits.

Se dejó fuera la arquitectura Pipelined, ya que la principal bondad de ésta es introducirle datos continuamente en cada ciclo de reloj, frame tras frame, y debido a la implementación del módulo que se hizo de la memoria RAM esto no sería posible, ya que entre cada lectura hay un retraso de tiempo.

La comparación se realizó implementando las tres arquitecturas, cada una capaz de realizar la FFT de 4 diferentes secuencias (64, 128, 256 y 512 puntos). Cada una de las implementaciones se simuló (simulación de tiempo) en el programa ISim de Xilinx®. Se realizaron dos tipos de comparación: los recursos ocupados para efectuar la transformada de Fourier, para cada secuencia implementada en cada una de las arquitecturas y el tiempo que les toma realizar la FFT. Esta comparación nos ayudó a determinar cómo se comporta el uso de recursos en la FPGA según se va aumentando el número de puntos por secuencia. Las Tablas de comparación completas se pueden observar en el Anexo B. El número de puntos se realizó hasta 512, ya que se va a implementar el módulo con este número de puntos. De acuerdo a estas graficas se observa que los recursos ocupados por cada arquitectura se mantienen casi constantes, no importando el número de puntos en cada secuencia. En la Figura 5.19 se puede apreciar la gráfica con los tiempos que le toma al *core* efectuar la FFT para diversas secuencias en las tres arquitecturas implementadas.

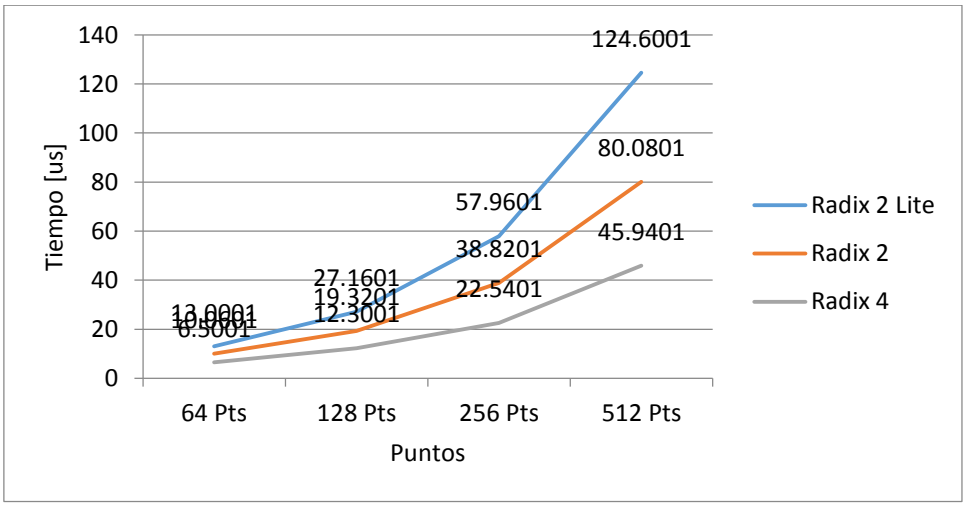


Figura 5.19._ Grafica comparativa con los tiempos para procesar la FFT en las diversas arquitecturas.

El tiempo que le toma al *core* efectuar la FFT va aumentando casi al doble con respecto a la anterior secuencia de datos. Para la implementación del *core*, de acuerdo a las gráficas, se decidió descartar la arquitectura Radix-4, ya que esta hace un uso excesivo de los multiplicadores (ver Anexo B), y la tarjeta Spartan®-3E (con la que se está trabajando) no tiene los suficientes multiplicadores, se requieren 48 y únicamente se cuentan con 28. Entre las dos arquitecturas restantes, se decidió

implementar la de Radix-2 ya que esta ofrece un mejor desempeño con respecto a la arquitectura Radix-2 Lite y no hace un uso excesivo de recursos (ver Anexo B). El módulo implementado se muestra en la Figura 5.20. El diseño final se implementó para efectuar la FFT de una secuencia de 512 puntos, con los datos de entrada y salida en punto flotante.

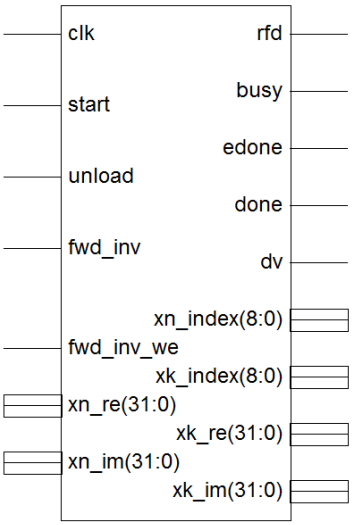


Figura 5.20._ Diagrama esquemático del módulo FFT.

Las descripciones de los pines del módulo se muestran en la Tabla 5.12.

Tabla 5.12._ Descripción de los pines del módulo FFT.

Señal	Ent.	Sal.	Descripción
clk	✔		Señal de reloj del sistema.
start	✔		Señal de inicio del *core*. Cuando se pone en un alto lógico, se empiezan a cargar los datos de entrada para iniciar el proceso de la FFT.
unload	✔		Inicia el proceso de descarga de datos en el bus de salida. Se activa en un nivel alto.
fwd_inv	✔		Señal que indica el tipo de transformada a efectuar, cuando es '1' se efectúa la transformada directa y cuando es '0' se efectúa la transformada inversa.
fwd_inv_we	✔		Activa la señal FWD_INV cuando es '1'.
xn_re(31:0)	✔		Bus de entrada para los datos reales.
xn_im(31:0)	✔		Bus de entrada para los datos imaginarios.
rfd		✔	Señal que se mantiene en un alto lógico durante la carga de datos.

busy	✔	Señal que se mantiene en un alto lógico durante el cálculo de la FFT.
edone	✔	Señal que se pone en un alto lógico un ciclo después que la señal done
done	✔	Una vez que el cálculo de la FFT es completado esta señal se pone en un alto lógico durante un ciclo de reloj.
dv	✔	Señal que se mantiene en un alto lógico mientras los datos son descargados en los buses de salida.
xn_index(8:0)	✔	Índice de los datos de entrada.
xk_index(8:0)	✔	Índice de los datos de salida.
xk_re	✔	Bus de salida de los datos reales.
xk_im	✔	Bus de salida de los datos imaginarios.

5.6 MODULO DE CONTROL DEL CORE FFT

Este módulo se encarga de llevar el control para efectuar el algoritmo de la FFT a una imagen. En la Figura 5.21 se muestra el diagrama de bloques del funcionamiento del módulo completo.

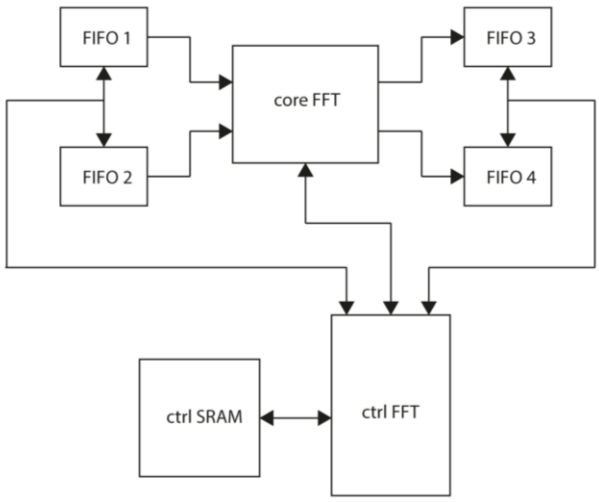


Figura 5.21._ Diagrama de bloques del módulo para la implementación del algoritmo de la FFT para una imagen.

Para efectuar el algoritmo de la FFT en 2D, primero obtenemos la Transformada de cada columna (de la imagen), obteniendo de esta forma una nueva matriz. Posteriormente obtenemos la FFT de cada renglón de la matriz obtenida anteriormente. Por lo tanto, se pretende primero leer los datos originales de la imagen que están almacenados en la SRAM, aplicarles la FFT (por columnas) y los datos resultantes almacenarlos nuevamente en la SRAM, volver a leer estos nuevos datos, pero ahora separándolos por renglones, aplicarles la FFT y nuevamente los datos resultantes almacenarlos en la SRAM.

5.6.1 DESCRIPCIÓN DE LOS SUBMÓDULOS QUE INTEGRAN AL MÓDULO DE CONTROL DEL CORE FFT

El módulo completo consta de tres submódulos:

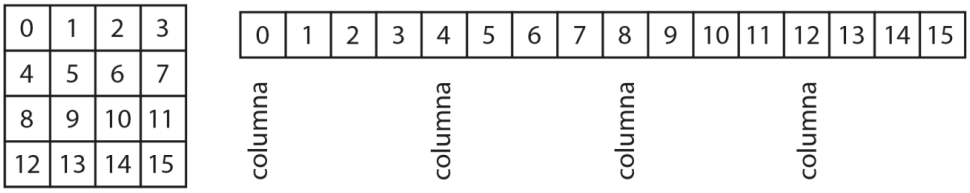
1) Pilas.
2) *Core* FFT.
3) Módulo de control del *core* FFT, en este último se implementa el algoritmo para el cálculo de la FFT de una imagen.

Pilas: Debido a que el *core* FFT carga los datos imaginarios y reales en cada ciclo de reloj (20 ns) y la lectura de la SRAM tarda 140 ns, es necesario ocupar pilas que nos permitan almacenar una secuencia completa de datos leídos desde la SRAM, para posteriormente descargar estos datos en cada ciclo de reloj y al mismo tiempo cargarlos en el *core* FFT. Lo mismo sucede en la descarga de datos, el *core* FFT descarga los datos en cada ciclo de reloj (20 ns), por lo tanto también es necesario utilizar pilas para descargar primero ahí los datos y, posteriormente, ir sacando los datos de la pila conforme se van guardando en la SRAM. Las pilas ocupadas son de tipos FIFO. Básicamente, de acuerdo a la Figura 5.21, lo primero que se hace es leer una secuencia completa (una columna) de la SRAM, esta se va almacenando en las pilas 1 y 2, cuando se termina de almacenar la secuencia completa, el módulo de control del *core* le "dice" al *core* FFT que inicie la operación, en este momento los datos se descargan de las pilas y se cargan en el *core* FFT, para dar inicio al cálculo de la FFT. Cuando el *core* termina la transformada, empieza a descargar los datos en otras dos pilas, FIFO 3 y FIFO 4, para los datos reales e imaginarios respectivamente, cuando estas dos pilas se llenan el módulo de control del *core* empieza a activar la operación de escritura de la SRAM. El proceso anterior continúa hasta que las pilas 3 y 4 quedan completamente vacías. Se repite todo el proceso (desde la lectura de la SRAM hasta el almacenamiento de los datos procesados en la SRAM) hasta completar todas las columnas. Una vez que se completa el procesamiento anterior, se inicia el proceso del cálculo de la FFT de los renglones, se procede de la misma forma descrita anteriormente.

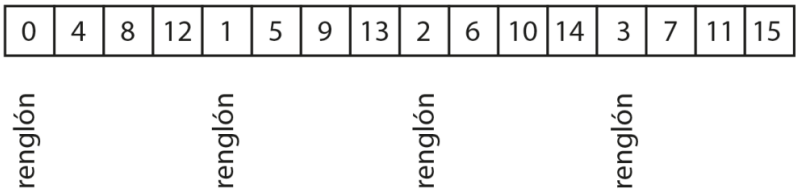
***Core* FFT.** Módulo encargado de realizar la operación de cálculo de la FFT, ya se de una columna (512 datos) o un renglón (512 datos).

Módulo de control del *core* FFT. Modulo donde se implementa el algoritmo para el cálculo de la FFT de una imagen.

La explicación del algoritmo para el cálculo de la FFT en 2D se ejemplifica en la Figura 5.22. La imagen primero se almacena en la memoria RAM en forma de un vector (Figura 5.22a). Después se accede a todas las localidades donde se encuentran los datos de las columnas (por ejemplo, la primer columna está formada por los datos 0, 4, 8 y 12), a los cuales se les aplica la FFT, el resultado se almacena en forma de vector, el cual estará ordenado en forma de columnas (los datos de las columnas estarán uno tras otro), se aplica la FFT a todos las columnas. Al finalizar la FFT de las columnas se obtienen dos matrices, una imaginaria y otra real. Se vuelve a aplicar la FFT, pero ahora a los renglones (Figura 5.22b, el primer reglón está formado por los datos 0, 1, 2, 3), tanto a las matrices reales e imaginarias, obteniéndose otras dos matrices una real y otra imaginaria. En total se necesitan 4 MB para almacenar las 4 matrices anteriores.



a)



b)

Figura 5.22._ Los datos de la imagen se almacenan en forma de vector en la memoria SRAM (a), primero se obtiene la FFT de todas las columnas, el resultado se almacena nuevamente en forma de vector (b), a estos datos se les vuelve a calcular la FFT, pero ahora de los renglones.

En la Figura 5.23 se muestra el diagrama esquemático del módulo implementado de control del *core* FFT.

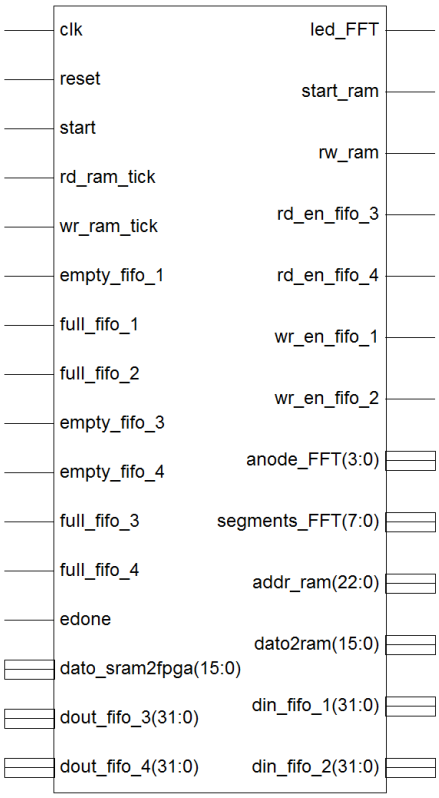


Figura 5.23._ Módulo de control del core *FFT*

En la Tabla 5.13 se describe el funcionamiento de cada pin del módulo de control del *core* FFT.

Tabla 5.13._ Descripción de los pines del módulo de control del core *FFT.*

Señal	Ent.	Sal.	Descripción
Clk	✔		Señal de reloj del sistema.
Start	✔		Señal de inicio del módulo. Cuando se pone en un alto lógico se inicia el proceso del cálculo de la FFT de la imagen.
Reset	✔		Ponemos todas las señales a un estado conocido.

rd_ram_tick	✔		Entrada de la señal del módulo de control de la SRAM que nos indica cuando se ha terminado de realizar el proceso de lectura.
rd_ram_tick	✔		Entrada de la señal del módulo de control de la SRAM que nos indica cuando se ha terminado de realizar el proceso de escritura.
empty_fifo_1	✔		Bus de entrada para la señal *empty* de la pila 1.
full_fifo_1	✔		Bus de entrada para la señal *full* de la pila 1.
full_fifo_2	✔		Bus de entrada para la señal *full* de la pila 2.
empty_fifo_3	✔		Bus de entrada para la señal *empty* de la pila 3.
empty_fifo_4	✔		Bus de entrada para la señal *empty* de la pila 4.
full_fifo_3	✔		Bus de entrada para la señal *full* de la pila 3.
full_fifo_4	✔		Bus de entrada para la señal *full* de la pila 4.
edone	✔		Bus de entrada de la señal *edone* proveniente del *core* FFT, el cual nos indica cuando el cálculo de la FFT termina.
dato_sram2fpga (15:0)	✔		Bus de entrada de los datos que se han leído desde la SRAM.
dout_fifo_3	✔		Bus de salida de los datos que se van a almacenar en la pila 3.
dout_fifo_4	✔		Bus de salida de los datos que se van a almacenar en la pila 4.
Led_FFT		✔	Señal que se mantiene en un alto lógico durante el proceso del cálculo de la FFTt de la imagen.
start_ram		✔	Señal de salida para iniciar el proceso de acceso a la SRAM.
rw_ram		✔	Señal que prepara el proceso de escritura o lectura de la SRAM
rd_en_fifo_3		✔	Señal que activa el proceso de lectura de la pila 3.
rd_en_fifo_4		✔	Señal que activa el proceso de lectura de la pila 4.
wr_en_fifo_3		✔	Señal que activa el proceso de escritura de la pila 3.
wr_en_fifo_4		✔	Señal que activa el proceso de escritura de la pila 4.
addr_ram(22:0)		✔	Bus de salida de las direcciones de la memoria SRAM.
dato2ram(15:0)		✔	Bus de salida de los datos a guardar en la memoria SRAM.

din_fifo_1	✔	Bus de salida que se conecta al bus de entrada de la pila 1.
din_fifo_2	✔	Bus de salida que se conecta al bus de entrada de la pila 2.

5.6.2 INTEGRACIÓN DE LOS SUBMÓDULOS

Como ya se explicó en los párrafos anteriores, los submódulos que conforman la parte del sistema que efectúan la FFT de la imagen son:

- 4 Pilas del tipo FIFO.
- Core FFT
- Módulo de control del *core* FFT.

Además de estos, se integró un submódulo adicional, el cual es un contador que sirve para medir el tiempo de procesado de la FFT de la imagen. En la Figura 5.24 se ilustra el diagrama esquemático del módulo general, llamado ctrl_FFT y en la Figura 5.25 se muestra la interconexión de los distintos módulos empleados.

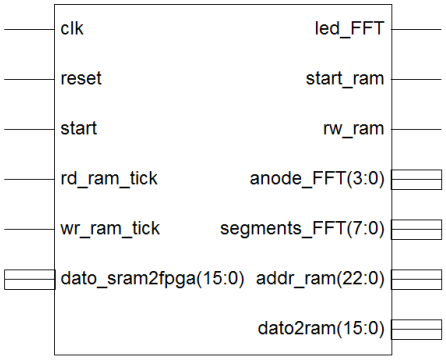


Figura 5.43._ Diagrama esquemático del módulo de control de la FFT (ctrl_FFT).

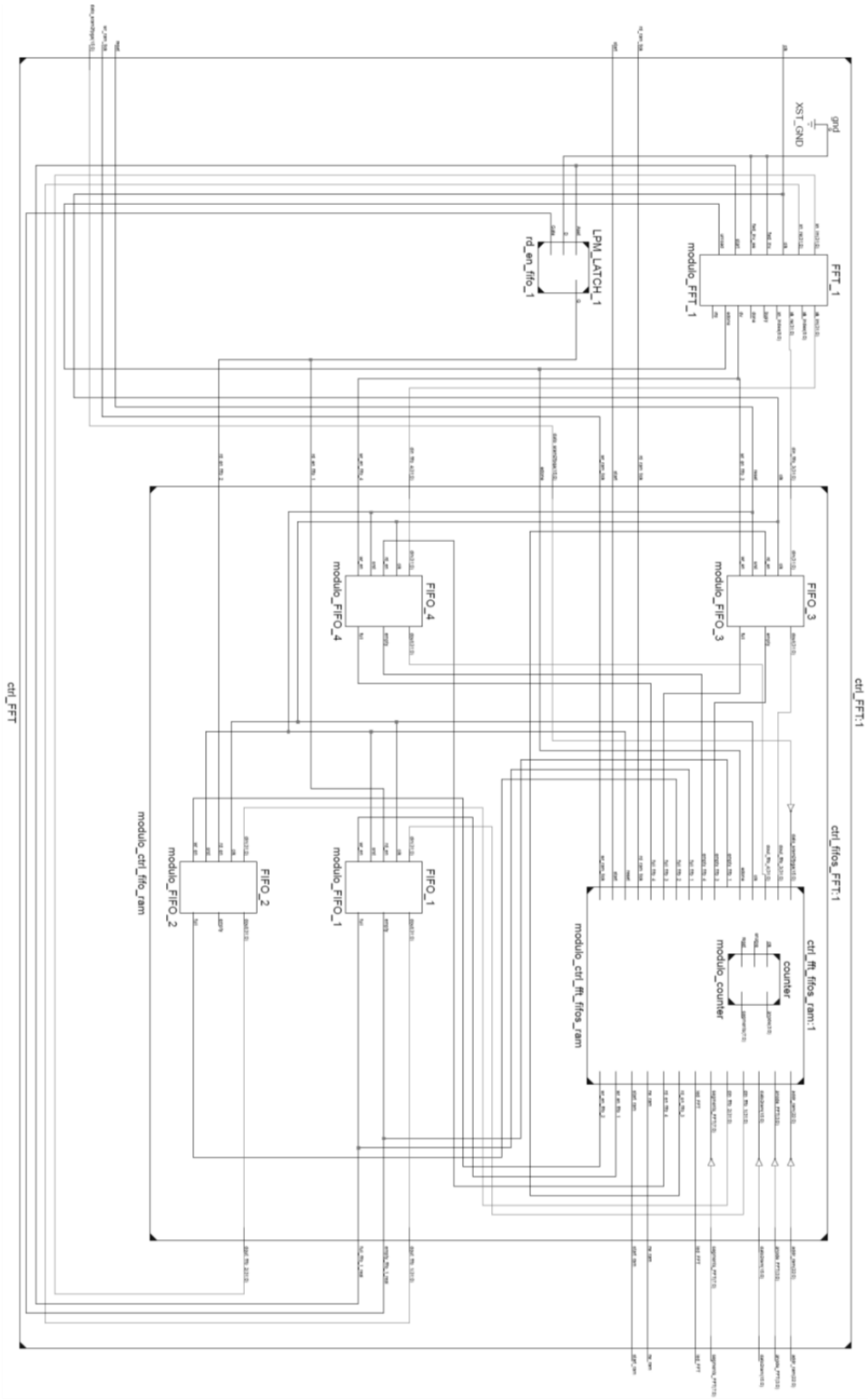


Figura 5.25._ Integración de las pilas, el módulo de control del core FFT y el core FFT para formar el módulo ctrl_FFT.

5.6.3 ANÁLISIS DE TIEMPO DEL MÓDULO DE CONTROL DEL CORE FFT

Como se vio en la sección anterior, el proceso de la obtención de la FFT se puede desglosarse en los siguientes puntos:

- Cálculo de la FFT por columnas:
    1) Lectura de los datos de la imagen original desde la SRAM y guardado de estos en la FIFO 1.
    2) Proceso del cálculo de la FFT, la cual incluye la carga de los datos (desde la FIFO 1 y 2) al core FFT, cálculo de la FFT y descarga del resultado del core (hacia las FIFO 3 y 4).
    3) Lectura de datos de la FIFO 3 y 4 y almacenamiento de los datos (Resultado de la FFT por columnas) leídos en la SRAM.
- Cálculo de la FFT por renglones:
    1) Lectura de los datos (FFT por columnas) de la SRAM y almacenamiento en las FIFO 1 y 2.
    2) Descarga de los datos de las FIFO 1 y 2 en el core FFT, cálculo de la FFT y descarga de los resultados (FFT por renglones) en las FIFO 3 y 4.
    3) Lectura de datos de la FIFO 3 y 4 y almacenamiento de los datos (resultado de la FFT por renglones) leídos en la SRAM.

Con base a las simulación de tiempo realizadas en el entorno ISE® se hizo un análisis de los datos para determinar los tiempos que le toma al módulo completar ciertos procesos, además de obtener el tiempo total que lleva completar el algoritmo para procesar la FFT de una imagen. Estos tiempos se describen en la Tabla 5.14.

Tabla 5.14._ Tiempo de los procesos que son efectuados en el módulo de control del core FFT

Descripción	Tiempo (us)	Total parcial (us)	Total (us)
Calculo de la FFT por columnas			
Tiempo de lectura de un único dato de una columna desde la SRAM (se efectúan dos accesos a memoria) y guardado en la FIFO 1.	0.42		
Tiempo de lectura de todos los datos una columnas (dato anterior x 511, se descarta el ultimo valor, ya que después de la última lectura se inicia inmediatamente el siguiente proceso, ya no hay retardo de lectura.)		214.62	
Delay después de que la señal start del módulo del core FFT se pone en un nivel alto hasta que se inicia la carga de datos.	0.02		
Carga de datos	10.24		
Calculo de la FFT	59.26		
Delay antes de la descarga	0.32		

Descarga de datos	10.24		
Tiempo que se tarda en realizar la carga, transformada de Fourier y la descarga de datos en la pila		80.08	
Delay antes del siguiente proceso	0.06		
Tiempo de lectura de un dato de la FIFO 3 y almacenamiento en la SRAM	0.36		
Tiempo de lectura de todos los datos de la FIFO 3 (dato anterior x 511, se descarta el ultimo valor, ya que después de la última lectura se inicia inmediatamente el siguiente proceso, ya no hay retardo de lectura.)		184.02	
Delay antes del siguiente proceso	0.38		
Tiempo de lectura de un dato de la FIFO 3 y almacenamiento en la SRAM	0.36		
Tiempo de lectura de todos los datos de la FIFO 3 (dato anterior x 511, se descarta el ultimo valor, ya que después de la última lectura se inicia inmediatamente el siguiente proceso, ya no hay retardo de lectura.)		184.34	
Tiempo extra para realizar comparaciones y delays antes de empezar determinados procesos		0.78	
Total de tiempo por 1 columna		663.84	
Tiempo total de las 512 columnas			339886.08
Calculo de la FFT por renglones			
Tiempo de lectura de un renglón completo desde la SRAM (se efectúan dos accesos a memoria) y guardado en la FIFO 2.	214.62		
Delay antes de la siguiente lectura a la SRAM	0.44		
Tiempo de lectura de un renglón completo desde la SRAM (se efectúan dos accesos a memoria) y guardado en la FIFO 1.	214.62		
Tiempo total de la lectura de la SRAM y almacenamiento en la FIFO 1 y 2.		429.68	
Delay después de que la señal *start* del módulo del core FFT se pone en un nivel alto hasta que se inicia la carga de datos.	0.02		
Carga de datos	10.24		
Calculo de la FFT	59.26		
Delay antes de la descarga	0.32		
Descarga de datos	10.24		
Tiempo que se tarda en realizar la carga, FFT y la descarga de datos en la pila		80.08	
Delay antes del siguiente proceso	0.06		
Tiempo de lectura de un dato de la FIFO 3 y almacenamiento en la SRAM	0.36		
Tiempo de lectura de todos los datos de la FIFO 3 (dato anterior x 511, se descarta el ultimo valor, ya que después de la última lectura se inicia inmediatamente el siguiente proceso, ya no hay retardo de lectura.)		184.02	
Delay antes del siguiente proceso	0.38		
Tiempo de lectura de un dato de la FIFO 3 y almacenamiento en la SRAM	0.36		
Tiempo de lectura de todos los datos de la FIFO 3 (dato anterior x 511, se descarta el último valor, ya que después de la última lectura se inicia inmediatamente el siguiente proceso, ya no hay retardo de lectura.)		184.34	

Tiempo extra para realizar comparaciones y delays antes de empezar determinados procesos	0.78	
Total de tiempo por 1 renglón.	878.90	
Tiempo total de los 512 renglones.		449996.8
Total en completar la FFT de una imagen de 512 x 512		789882.88

5.7 Integración del sistema (Módulo de control principal del sistema)

Una vez completados los 5 módulos principales, éstos se integraron dentro de un módulo principal (módulo de control principal del sistema) el cual servirá como módulo de control de todo el sistema. Debido a que los módulos ctrl UART TX, ctrl UART RX y control del core FFT hacen uso de la memoria SRAM y sólo se dispone de un módulo de control para ésta, éstos 3 módulos tendrán que compartir el módulo de control de la SRAM, por lo tanto, se implementaron multiplexores a la salida de los 3 módulos, para que no haya conflictos al momento de acceder a la memoria SRAM.

Los multiplexores se utilizan en las señales de salida de los pines de los módulos, para las señales de entrada de los pines se utilizan comparadores. Dependiendo del valor que tenga el multiplexor en su señal de selección, los comparadores hacen una “comparación” con el valor del selector, si es igual a ese valor la salida del comparador es igual a su entrada, por lo tanto hay comunicación entre la salida de un módulo y la entrada hacia otro módulo, en caso de que la comparación sea distinta no existirá comunicación entre módulos. El diagrama de conexión completo que encapsula a los 5 módulos descritos en las secciones anteriores se muestra en la Figura 5.26.

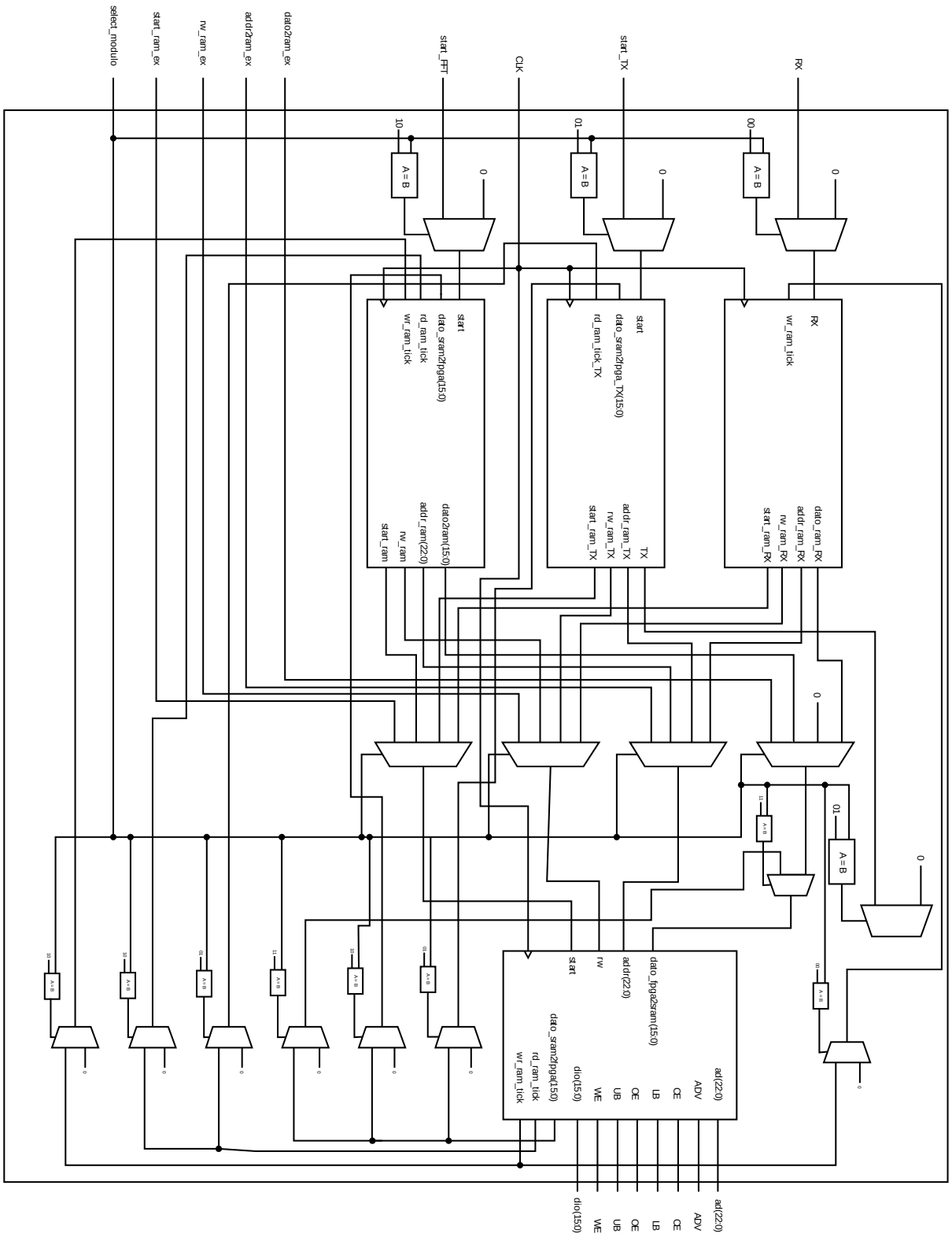


Figura 5.26._ Integración de las pilas, el módulo de control del core FFT y el core FFT para formar el módulo ctrl_FFT.

En la Figura 5.27 se ilustra el diagrama esquemático del sistema implementado y en la Tabla 15.15 se describen las entradas y salidas del módulo.

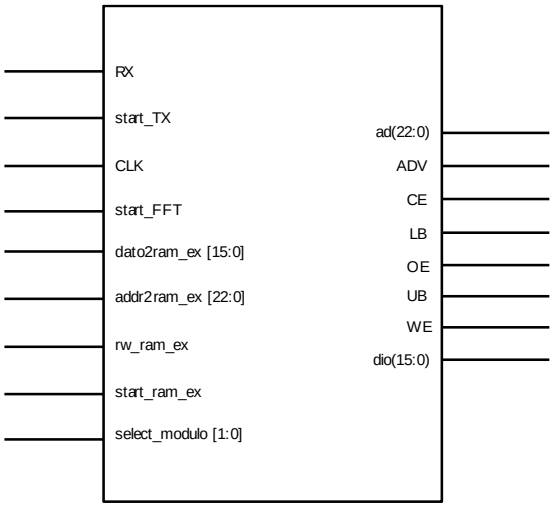


Figura 5.27._ Diagrama esquemático del Sistema desarrollado.

Tabla 5.15._ Descripción de los pines del Sistema implementado.

Señal	Ent.	Sal.	Descripción
clk	✔		Señal de reloj para el control del proceso.
RX	✔		Bus de entrada de datos serie.
start_TX	✔		Inicia la transmisión de datos.
start_FFT	✔		Inicia el cálculo de la FFT en 2D.
select_modulo [1:0]	✔		00 – se activa el módulo de recepción de datos, 10 – se activa el módulo para el cálculo de la FFT en 2D, 01 – se activa el módulo para la transmicion de datos, 11 – Se tiene acceso a la memoria SRAM de forma externa.
dato2ram_ex(15:0)	✔		Dato a escribir en la memoria RAM.
Add2ram_ex(15:0)	✔		Dirección de memoria en la que se quiere almacenar. los datos externos.
rw_ram_ex	✔		Se activa el proceso de lectua o escritura de la SRAM.
start_ram_ex	✔		Se activa el acceso a la memoria SRAM.
ADV, OE, CE, WE, LB, UB		✔	Señales de control de la RAM.

ad (22:0)		✔	Arreglo de 23 bits de la dirección de lectura o escritura de la RAM.
dio (15:0)	✔	✔	Bus de datos que se han leído o se van a escribir en la RAM.

En la tabla 5.16 se muestra la utilización de recursos del FPGA Spartan®-3E con el sistema completamente implementado.

Utilización de recursos			
Registros	Usados	Disponibles	Utilizados
Registros en slices	4,944	17,344	28%
LUTs de 4 entradas	5,361	17,344	30%
Slices	4,243	8,672	48%
Total Number of 4 input LUTs	6,123	17,344	35%
IOBs	83	250	33%
RAM en bloques	11	28	39%
DCMs	1	8	12%
Multiplicadores	16	28	57%

5.7.1 ANÁLISIS DE TIEMPO Y RELACIÓN CON UN SISTEMA DE ÓPTICA ADAPTATIVA

Antes de grabar el FPGA con el sistema diseñado, se realizó el análisis de tiempo estático (ver sección 4.5.3), con el cual se obtuvo el mayor retardo de propagación presente en el sistema. En la tabla 5.17 se muestran los datos que despliega el entorno ISE®.

Tabla 5.17. Tabla con el análisis de tiempo estático.

Slack (setup path):6.579ns **(requerimiento - (retardo - clock path skew + uncertainty))**

Fuente: modulo_master_ctrl/modulo_ctrl_FFT/modulo_ctrl_fifo_ram/modulo_ctrl_fft_fifos_ram/estado_actual_FSM_FFd49 (FF)	**clk:** modulo_master_ctrl/clk_50MHz rising at 0.000ns
Destino: modulo_master_ctrl/modulo_ctrl_FFT/modulo_ctrl_fifo_ram/modulo_ctrl_fft_fifos_ram/dato2ram_actual_8 (FF)	**clk:** modulo_master_ctrl/clk_50MHz rising at 20.000ns

Requerimiento	Retardo	Clock Path Skew:	Clock Uncertainty
20.000ns	13.421ns	0.000ns	0.000ns

El sistema debe funcionar correctamente a 50 MHz, por lo tanto no debe de existir un retardo que supere los 20 ns. De acuerdo a la tabla el retardo presente es de 13.42 ns, por lo tanto se tiene un tolerancia (Slack) de 6.579 ns. Con esto se demuestra que el sistema va a funcionar correctamente con el reloj que se va a ocupar. Sin embargo, no es posible que el sistema funcione de manera adecuada con relojes que trabajan con frecuencias mayores a 74 MHz, ya que en estos casos los pulsos de reloj se ven superados por el retardo de propagación. Este restricción se puede evitar indicándole al entorno ISE® que busque un mejor enrutamiento (ver Sección 4.5.3) de los componentes para disminuir el retardo de propagación, poniendo como condición de que se va a ocupar un reloj con una mayor frecuencia (por ejemplo 100 MHz).

En la sección 5.6.3 se obtuvo el tiempo que el sistema tarda en calcular la FFT de una imagen, el cual es de 789.882 ms. Un sistema de óptica adaptativa trabaja a frecuencias mayores a 2 Hz (500 ms), por lo que el sistema, en este momento, no sería apto para ser implementado en procesos de óptica adaptativa. Sin embargo es posible disminuir el tiempo de procesamiento al ocupar una mayor frecuencia de reloj y cambiar el tipo de memoria RAM por una que nos permita acceder a los datos en cada ciclo de reloj. Todo lo anterior con el fin de que el sistema completo de recuperación de frente de onda no pase los 500 ms.

Capítulo 6

Pruebas y Resultados

6.1 INTRODUCCIÓN

El conjunto formado por los módulos descritos en el capítulo 5 representan una biblioteca inicial que formará parte de la herramienta propuesta, la cual consiste en la obtención de la reconstrucción de un frente de onda a partir de los datos obtenidos por un sensor de curvatura. Para validar el funcionamiento correcto del sistema, éste se tiene que someter a pruebas experimentales. En este capítulo se presentan las pruebas y resultados experimentales en la evaluación del sistema para la obtención de la FFT en 2D.

6.2 PRUEBAS REALIZADAS AL SISTEMA

Las pruebas realizadas tienen como finalidad validar el correcto funcionamiento del sistema. Se realizaron tres pruebas, las cuales fueron:

1) Transmisión de datos PC – FPGA.
2) Transmisión de datos FPGA – PC.
3) Procesamiento de datos (validación del sistema).

La primera prueba válida que el módulo *ctrl UART RX* funcione correctamente. De forma general, la prueba consiste en transmitir los datos de una imagen hacia la FPGA a través del protocolo RS-232, estos datos se guardaran en la memoria SRAM. En la segunda prueba se leen los datos de la memoria SRAM y se transmiten desde el FPGA (pasando en su trayecto por la tarjeta

de evaluación Nexys®2) hacia la PC, validando de esta forma el módulo *ctrl UART TX* . Los datos transmitidos son los mismos datos que se enviaron desde la PC hacia la FPGA.

En las dos pruebas descritas anteriormente, se están validando los siguientes módulos principales:

- Control de la memoria SRAM.
- Ctrl UART RX.
- Ctrl UART TX.

En las dos pruebas descritas anteriormente no se realiza ningún tipo de procesamiento de datos, únicamente se está validando que exista una correcta comunicación PC - FPGA – PC. Como apoyo a estas pruebas se programaron dos aplicaciones en LabVIEW™, las cuales se describirán más adelante. Como dato adicional, en estas pruebas se registró el tiempo de transmisión y recepción de datos de manera experimental.

La tercera prueba consiste en validar los siguientes dos módulos principales:

1) Core FFT.
2) Control del core FFT.

En esta prueba también se valida el correcto funcionamiento del sistema que se desarrolló en la tesis. En la Figura 6.1 se ilustra el diagrama de bloques del funcionamiento de la prueba.

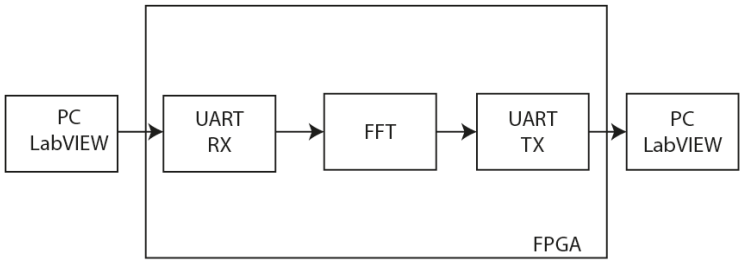


Figura 6.1._ Diagrama de bloque para la validación del sistema.

Una vez que los módulos de comunicación estén funcionando de manera correcta, se procederá a enviar una imagen de prueba a la FPGA, con el fin de que esta sea procesada, aplicándole la FFT. La manera en la que se valida que la imagen fue procesada correctamente, es

por medio de la PC, los datos procesados se transmiten a la PC (via LabVIEW™), que es desde donde se reensamblan y se comparan con datos procesados en Matlab®.

6.2.1 PRUEBA DE TRANSMISIÓN DE DATOS PC - FPGA

Esta prueba verifica el correcto funcionamiento del módulo *ctrl UART RX,* y por lo tanto se validan los siguientes puntos:

1) Transmisión de datos con el protocolo RS-232.
2) Tiempo de transmisión de datos para una imagen de 512 x 512 pixeles.

Para validar esta prueba, se realizó una aplicación en LabVIEW™ con el fin de ayudarnos a transmitir datos de la PC al FPGA, por medio del protocolo RS-232. Se decidió programar la aplicación en LabVIEW™ por la facilidad que tiene este software para implementar diversas interfaces de comunicación. El diagrama de bloques de la figura 6.2 describe el funcionamiento de la prueba.

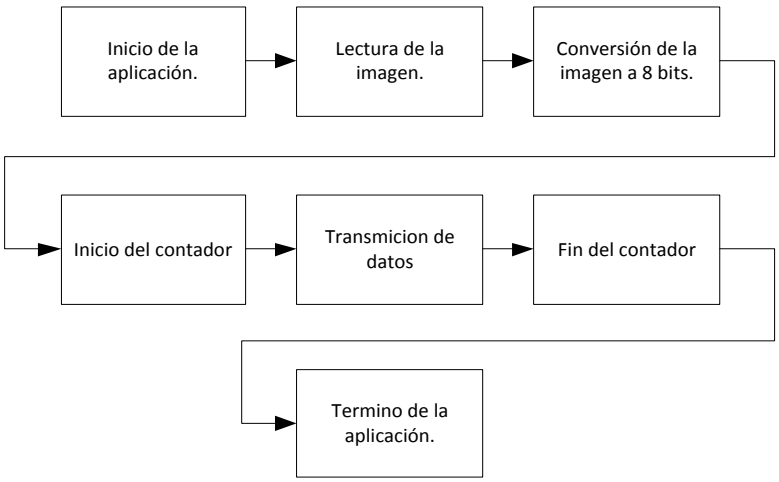


Figura 6.2._ Diagrama de bloques del funcionamiento de la aplicación.

Primero se carga la imagen en la aplicación que se desarrolló en LabVIEW™, en la cual no existe restricción de tamaño (ni profundidad de color), si es mayor a 512x512 pixeles, la imagen transmitida será cortada, si es menor a 512x512 pixeles, los datos faltantes serán llenados con ceros. Posteriormente, la imagen será convertida a una escala de grises con 256 niveles de intensidad (8 bits). Ya que se tenga la imagen procesada en LabVIEW™ la aplicación procederá a transmitir los

datos. Siempre se transmitirán los 512 x 512 Bytes. Justo antes de que se inicie la transmisión de datos, se inicia un contador (programado dentro de la aplicación de LabVIEW™), el cual tiene como objetivo tomar el tiempo total de transmisión de datos. El contador termina cuando la transmisión de datos concluye. La interfaz gráfica de la aplicación se muestra en la Figura 6.3, en la cual se muestra la matriz de los datos en escala de grises de la imagen, la imagen que se cargó y la imagen en escala de grises, además del tiempo que la aplicación tardo en enviar los datos a la FPGA.

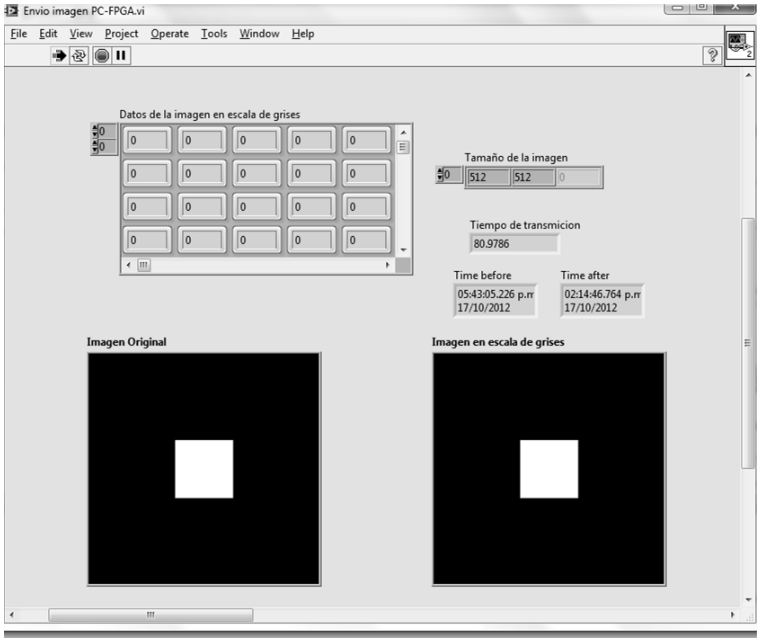
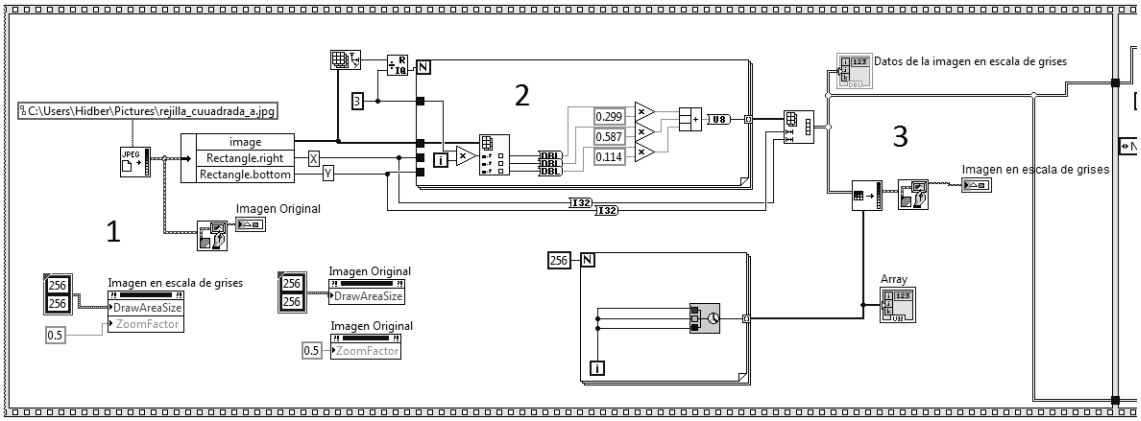
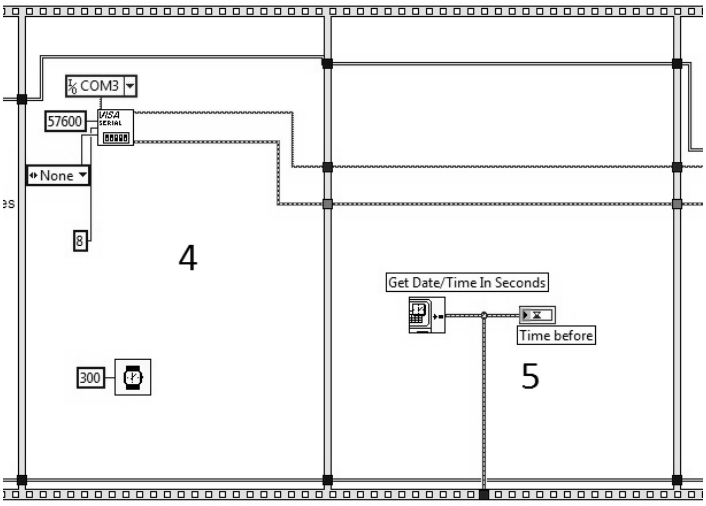


Figura 6.3._ Interfaz gráfica de la aplicación para la transmisión de datos.

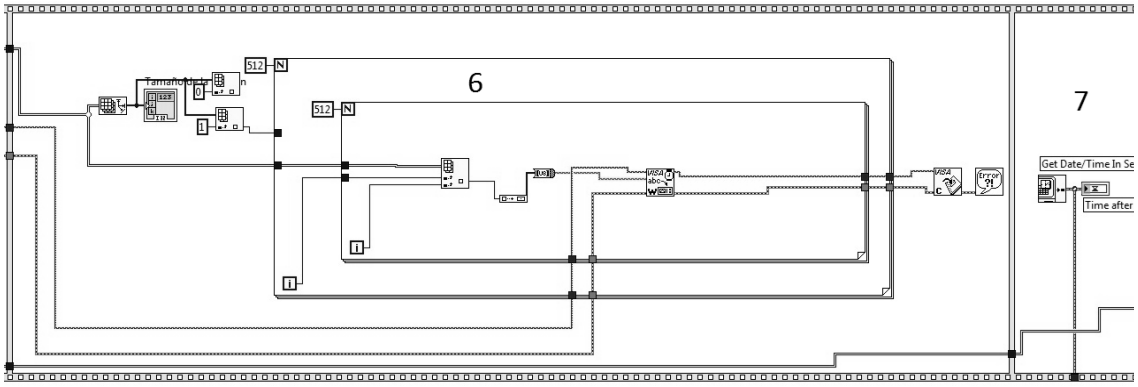
El código fuente de la aplicación programada se muestra en la Figura 6.4 y su descripción se detalla en los párrafos siguientes.



a)



b)



c)

Figura 6.4._Aplicacion para la transmisión de datos PC - FPGA. a) Lectura de los datos y conversión a una imagen de 8 bits, b) Configuración del puerto serie e inicio del contador, c) Envío de datos y termino del contador,

Primero se carga la imagen (Figura 6.4a - 1), la cual puede ser de cualquier tamaño y profundidad de color. Ya que el sistema únicamente procesa datos en escala de grises, es necesario que la imagen sea convertida a 8 bits. Por lo tanto se procede a realizar la conversión (Figura 6.4a - 2) pixel a pixel. Posteriormente se vuelve a rearmar la matriz de la imagen (Figura 6.4a - 3). Para la configuración de la interfaz RS-232 se utilizaron los archivos de LabVIEW™ NI VISA (Figura 6.4b - 4). La configuración del puerto serie se hizo de acuerdo a los siguientes parámetros:

- Velocidad de transmisión: 57600 bps.

- Bits de parada: 1.
- Paridad: Ninguna.
  Bits de datos: 8

Ya que se terminó de realizar la configuración del puerto, se inicia el contador (Figura 6.4b - 4). Inmediatamente después empieza la transmisión de datos (Figura 6.4c - 6), los cuales se envían pixel por pixel. El contador finaliza (Figura 6.4c - 7) inmediatamente después de que se terminaron de transmitir los datos.

6.2.1.1 IMPLEMENTACIÓN FÍSICA

La imagen de la Figura 6.5 muestra el set de pruebas utilizado. Mientras los datos se están transmitiendo un LED se mantiene encendido en la FPGA; cuando terminan de transmitirse este LED se apaga.

La validación del módulo de lectura y escritura de la SRAM se lleva a cabo mediante la prueba de transmisión de datos entre el FPGA y la PC, ya que con la prueba descrita anteriormente, la cual lleva implícita la escritura de la SRAM, no es posible saber si los datos se guardaron correctamente.

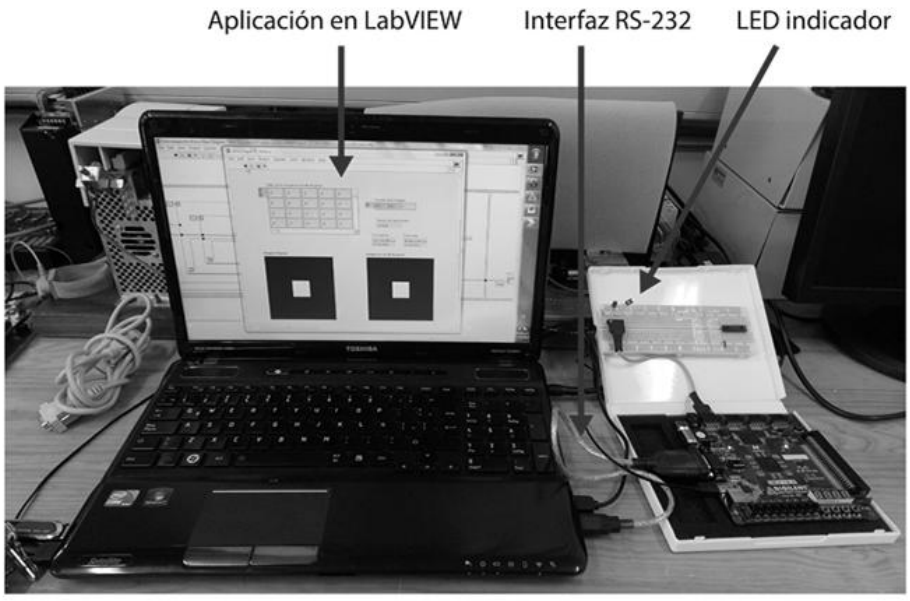


Figura 6.5._ Set de pruebas de la transmisión de datos.

La configuración de los *interruptores* y *push buttons* de la tarjeta de desarrollo Nexys®-2 para realizar esta prueba es la siguiente: SW0 y SW1 deben estar en OFF, BTN0 en OFF, los demás botones e interruptores pueden estar en cualquier estado, pero se recomienda que estén en OFF. En la Figura 6.6 se muestra la asignación de nombres a los *interruptores* y *push buttons*, el nombre se encuentra en la parte superior de cada elemento.

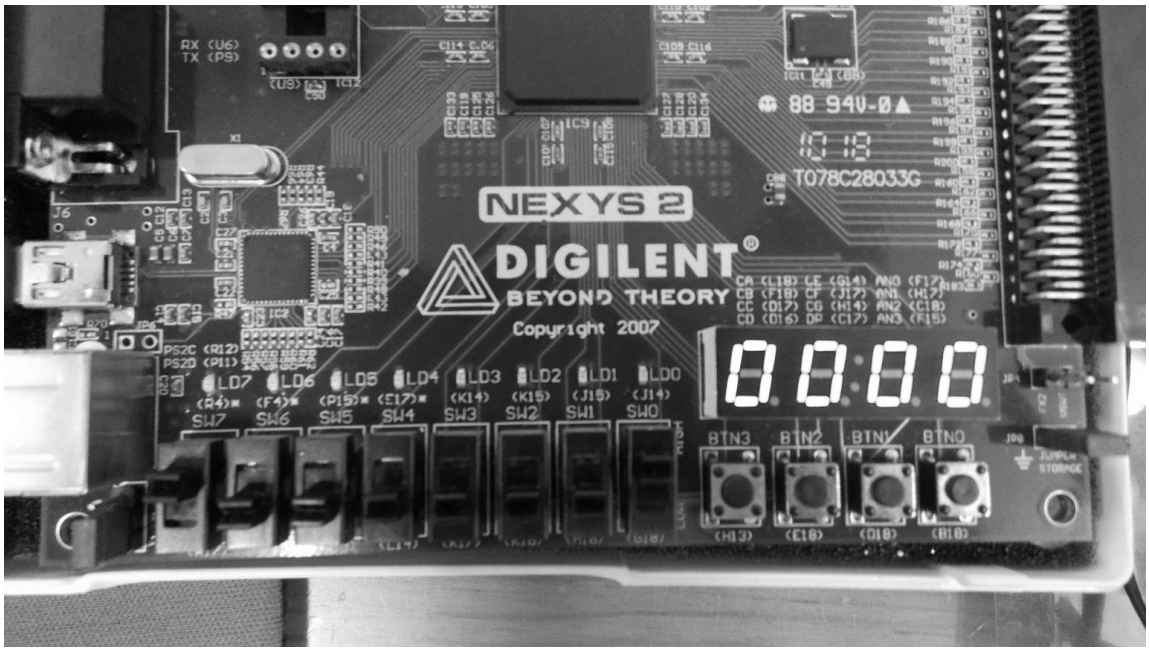


Figura 6.6._ Asignación de los nombres a los interruptores y push buttons. Los nombres se encuentran en la parte superior de cada elemento.

6.2.2 PRUEBA DE RECEPCIÓN DE DATOS PC – FPGA

Esta prueba verifica el correcto funcionamiento del módulo *ctrl UART TX,* y por lo tanto se validan los siguientes puntos:

3) Recepción de datos con el protocolo RS-232.
4) Tiempo de recepción de datos para una imagen de 512 x 512 pixeles.

Al igual que con la prueba anterior, para la validación de esta prueba se programó una aplicación en LabVIEW™, la cual nos permite recibir los datos en la PC, estos datos son enviados desde el FPGA. El diagrama de bloques de la figura 6.7 describe el funcionamiento de la prueba.

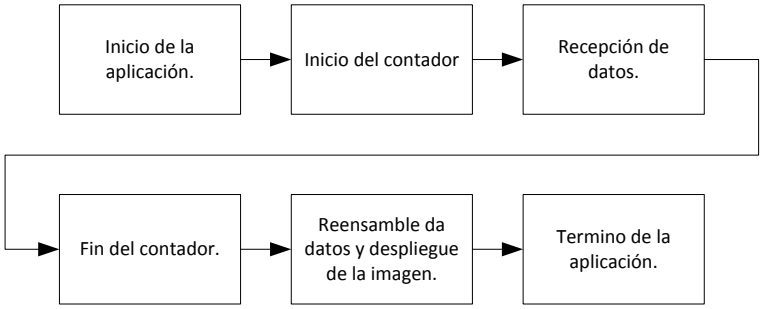


Figura 6.7._ Diagrama de bloques del funcionamiento de la aplicación.

Los datos son enviados a través del puerto RS-232 de la FPGA y son recibidos por la PC en el mismo puerto. Antes de que la computadora reciba el primer Byte, ésta inicia un contador. Cuando el FPGA termina de transmitir todos los datos, el contador programado en la aplicación que se está ejecutando en la PC finaliza e inmediatamente después los datos recibidos son reensamblados y se despliega la imagen resultante. La interfaz gráfica de la aplicación se muestra en la Figura 6.8, en la cual se muestra la matriz de datos recibidos y la imagen reconstruida a partir de esos datos.

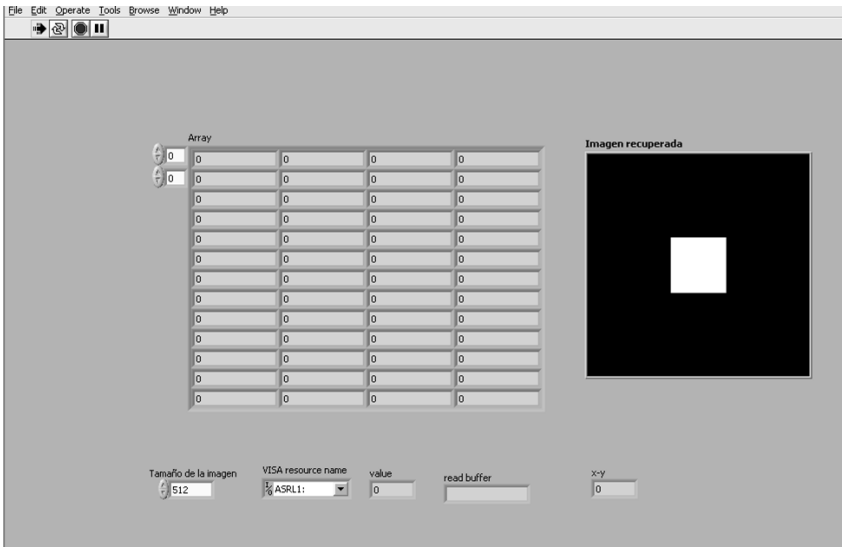
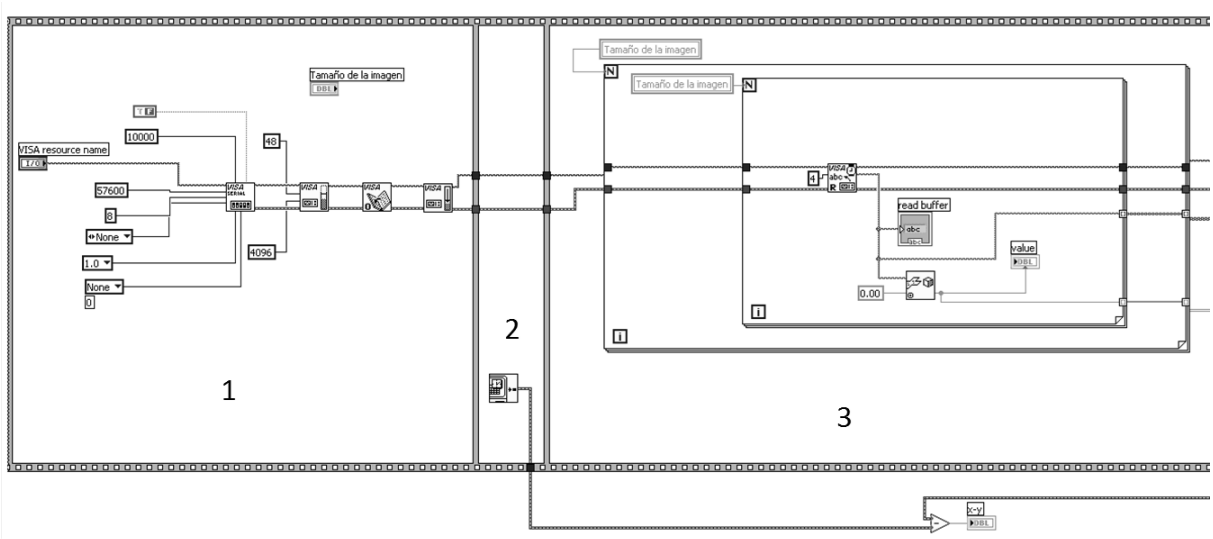
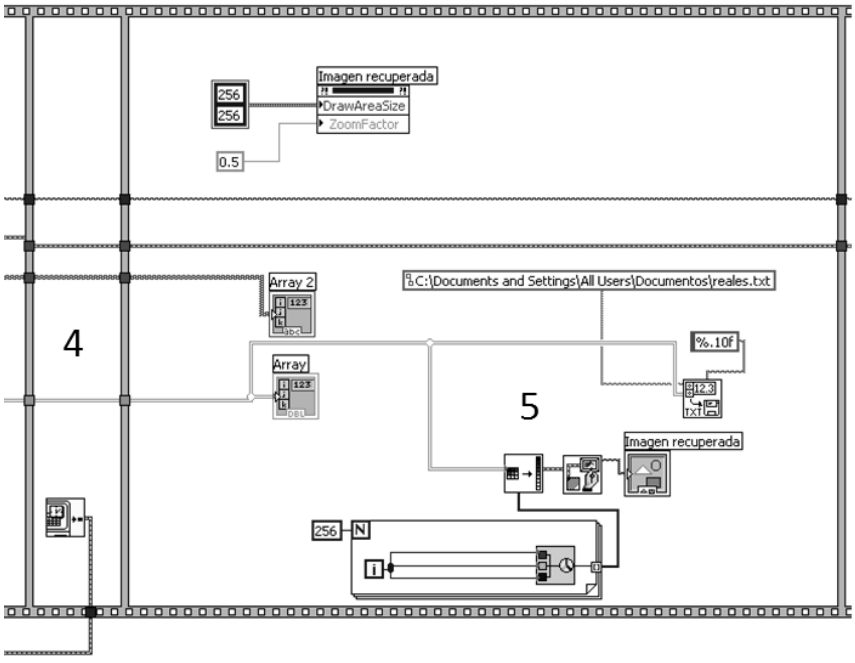


Figura 6.8._ Interfaz gráfica de la aplicación para la transmisión de datos.

El código fuente de la aplicación programada se muestra en la Figura 6.9 y su descripción se detalla en los párrafos siguientes.



a)



b)

Figura 6.9._Aplicacion para la recepción de datos PC - FPGA. a) Configuración del puerto serie y recepción de datos, b) Reensamble de los datos y despliegue de estos como una imagen.

La aplicación fue programada en la versión de LabVIEW™ 7.1 de 32 bits. Al igual que para la transmisión de datos, se utilizaron los archivos de LabVIEW™ NI VISA para la configuración de la interfaz RS-232, de acuerdo a los siguientes parámetros:

- Velocidad de recepción: 57600 bps.
- Bits de parada: 1.
- Paridad: Ninguna.
- Bits de datos: 8.

Una vez que se termina de configurar la interfaz (Figura 6.9a - 1), se toma la hora actual en segundos (Figura 6.9a - 2), la cual nos servirá para saber el tiempo total de recepción de datos. Posteriormente empezamos a recibir los datos, los cuales se van ordenando en forma de matriz, por medio de dos ciclos "*for"* anidados (Figura 6.9a -3). Debido a que el FPGA está mandando los datos en formato de punto flotante de 32 bits, en cada "*loop*" se reciben 4 paquetes de 8 bits, los cuales son agrupados en un paquete de 32 bits y posteriormente convertidos a su valor en decimal (con el VI *Unflatten From String*). Ya que se transmitieron todos los Bytes, se registra de nuevo la hora actual (Figura 6.9a -4) y se realiza una resta con la hora registrada anteriormente, con esto obtenemos el tiempo total de transmisión. El paso siguiente es convertir los datos a una imagen (para que pueda ser mostrada visualmente), para los cual utilizamos el VI *Flatten Pixmap* y *Draw Flattened Pixmap* (Figura 6.9a -5). Los datos además se guardan en un archivo de texto, para realizar esta acción se utiliza el VI *Write To Spreadsheet* (Figura 6.9a -5).

6.2.2.1 IMPLEMENTACIÓN FÍSICA

La imagen de la Figura 6.10 muestra el set de pruebas utilizado. Mientras los datos se están recibiendo un LED se mantiene encendido en la FPGA; cuando terminan de transmitirse este LED se apaga.

Para realizar satisfactoriamente esta prueba, los interruptores SW0 y SW7 (ver Figura 6.6) deben mantenerse en ON, y para iniciar la transmisión apretamos el *push button* BTN3, manteniéndolo por un momento es un estado lógico ON, los demás elementos se deben mantener en un estado lógico 0.

Con esta prueba se valida, como se mencionó en la sección 6.2.1.1, la escritura y lectura de la memoria SRAM, ya que los datos recibidos en la PC deben ser los mismos datos que se envían desde la PC.

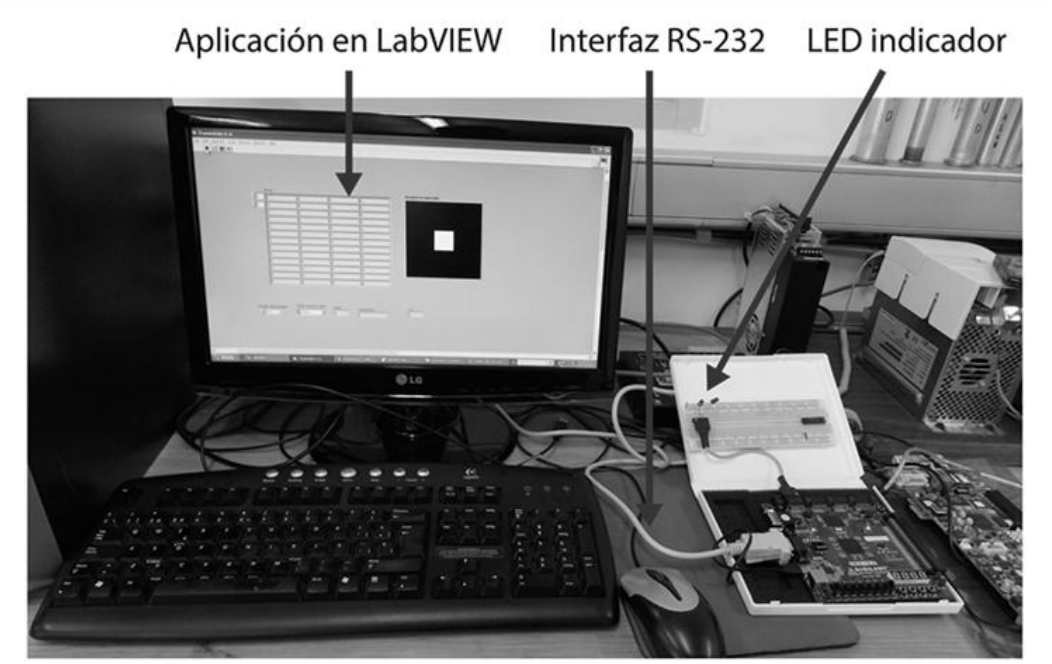


Figura 6.10._ Set de pruebas para la recepción de datos.

6.2.3 PRUEBA DE PROCESAMIENTO DE DATOS

Esta prueba verifica el correcto funcionamiento del módulo *Control Principal del sistema,* y por lo tanto se validan los siguientes dos puntos:

1) Validación del módulo de Control del *core* FFT y los submódulos asociados a él.
2) Tiempo de procesamiento del cálculo de la FFT.

La descripción de la prueba se aprecia en el diagrama de bloques de la Figura 6.11. La prueba de procesamiento de datos consiste en calcular la FFT de una imagen, la cual esta almacenada en la memoria SRAM. La imagen es enviada a la FPGA (la cual tiene control sobre la SRAM) a través de la interfaz RS-232. Una vez que la imagen ha sido procesada, esta es enviada de vuelta a la PC para su correcto análisis. Primero se envían los datos reales de la FFT, cuando finaliza la transmisión son almacenados en una archivo TXT. Posteriormente se envían los datos imaginarios de la FFT y nuevamente son almacenados en un archivo TXT.

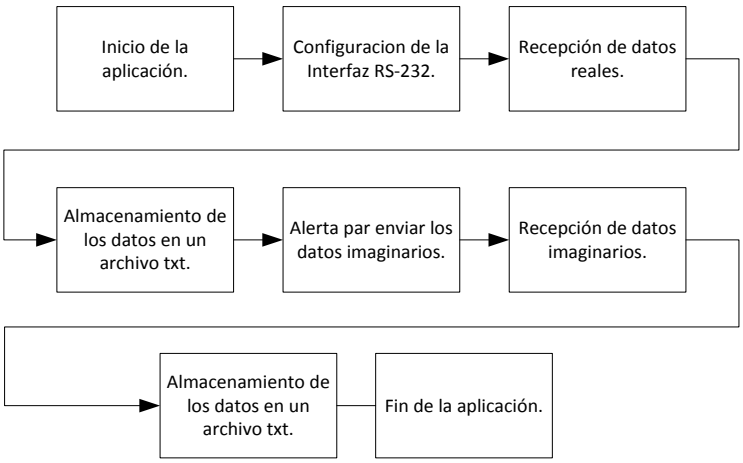


Figura 6.11._ Diagrama de bloque para la validación del sistema y el correcto procesamiento de datos.

Para realizar la transmisión de datos se utiliza la misma aplicación ocupada para la prueba de transmisión de datos desde la PC, explicada anteriormente. Para la recepción de datos se programó otra aplicación en LabVIEW™ 7.1. En la Figura 6.12 se muestra la interfaz gráfica de ésta aplicación, la cual es muy parecida a la ocupada anteriormente para transmisión de datos, pero con código fuente diferente.

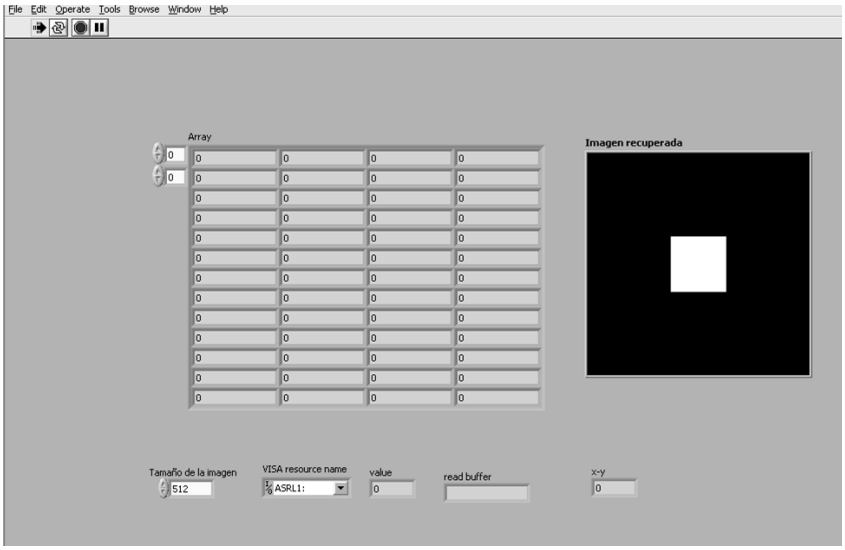
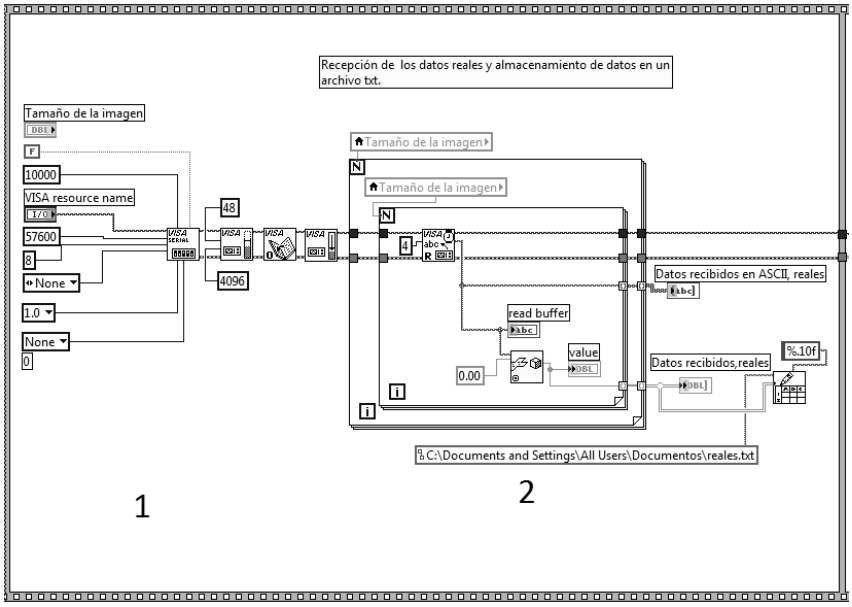
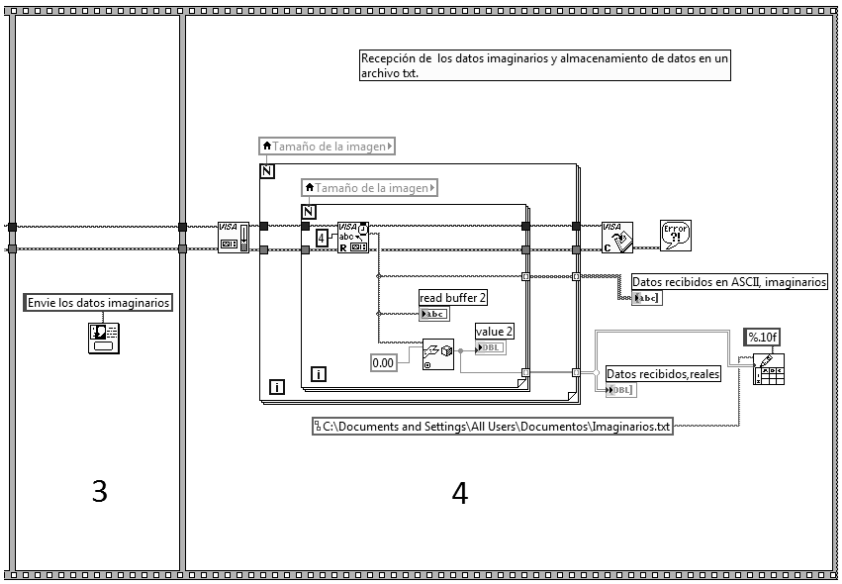


Figura 6.12._ Interfaz gráfica de la aplicación para la transmisión de datos.

El código fuente de la aplicación programada se muestra en la Figura 6.13 y su descripción se detalla en los párrafos siguientes.



a)



b)

Figura 6.13._ Aplicación para la recepción de datos, a) Configuración de la Interfaz RS-232 y recepción de la parte real de los datos de la FFT, b) Recepción de la parte imaginaria de los datos de la FFT

Primero se realiza la configuración de la interfaz RS-232 (Figura 6.13a - 1) con los mismos parámetros de las pruebas anteriores. La aplicación empieza a recibir primero los datos reales (Figura 6.13a - 2), recibe grupos de 4 Bytes, los cuales son convertidos de punto flotante a su representación en decimal, estos son colocados en una matriz de 512x512. Cuando se terminan de recibir todos los Bytes, la matriz resultante es almacenada como texto plano en un archivo txt.

Inmediatamente después de que se terminan de almacenar, salta a la vista un mensaje para que preparemos el envío de los datos imaginarios (Figura 6.13a - 3). Una vez que le demos *click* en el botón aceptar del mensaje, la aplicación estará lista para recibir los datos imaginarios (Figura 6.13a – 4), cuando termine la recepción de los datos, la aplicación los almacenara en un archivo txt, como texto plano. Los archivos donde se almacenaron los datos se emplean para validar el correcto procesamiento en la FPGA (ver Sección 6.3), estos datos se comparan en Matlab®.

6.2.3.1 IMPLEMENTACIÓN FÍSICA

La imagen de la Figura 6.14 muestra el set de pruebas utilizado. La configuración de los *interruptores* y *push buttons* de la tarjeta de desarrollo Nexys®-2 para recibir los datos en la FPGA es la siguiente: SW0 y SW1 deben estar en OFF, BTN0 en OFF, los demás botones e interruptores pueden estar en cualquier estado, pero se recomienda que estén en OFF. Para realizar satisfactoriamente la transmisión de datos desde la FPGA a la PC los interruptores SW0 y SW7 (ver Figura 6.6) deben mantenerse en ON, y para iniciar la transmisión apretamos el *push button* BTN3, manteniéndolo por un momento es un estado lógico ON, los demás elementos se deben mantener en un estado lógico 0.

Figura 6.14._ Set de pruebas para el procesamiento de datos.

Para verificar el tiempo de procesamiento, se programó en VHDL un contador, el cual inicia cuando el módulo de control del *core* FFT empieza a realizar el algoritmo de la FFT en 2D y termina cuando el mismo módulo de control finaliza el procesamiento. Este contador se implementó en la FPGA junto al sistema completo.

6.3 RESULTADOS

La prueba de transmisión de datos se efectúo sin ningún problema. Sin embargo, en la prueba de recepción de datos si se presentaron problemas. En un principio la aplicación para realizar la prueba de recepción de datos se programó en la versión de LabVIEW™ 2011 de 64 bits, sin embargo al momento de que los datos se estaban transmitiendo la aplicación dejaba de responder. Debido a que se ocupa un convertidor USB-RS-232 se llegó a pensar que el problema provenía de este convertidor. Para encontrar la solución se migró la aplicación a distintas versiones de LabVIEW™ (8.6, 2009 y 7.1, todas de 32 bits), para realizar las pruebas se utilizó el mismo convertidor. No surgió ningún problema con la aplicación en las versiones de 32 bits, únicamente con la versión de 64 bits, por este motivo, para la prueba de recepción de datos se ocupó la aplicación programada en LabVIEW™ 7.1 de 32 bits. En conclusión, el convertidor no funcionó en la versión de 64 bits ocupando la versión RC (del inglés, *Relase Candidate*) de NI VISA de agosto de 2011. En la Tabla 6.1 se muestran los tiempos obtenidos para la transmisión y recepción de datos.

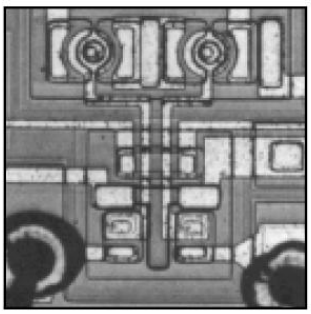
Tabla 6.1._ Tiempos obtenidos para las pruebas de transmisión y recepción de datos.

	Tiempo teórico [s]	Tiempo experimental (promedio) [s]
Transmisión de datos	45.51	81
Recepción de datos	768	788

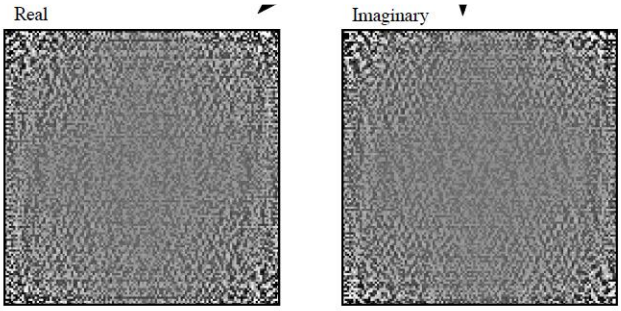
La diferencia de tiempos se debe, principalmente, a que el valor teórico se obtuvo sin tomar en cuenta los retardos que se presentan entre el término de transmisión de un Byte y el inicio de la transmisión del otro Byte (por parte de la PC). Lo mismo sucede con la recepción: no se toma en cuenta el retardo presente entre la transmisión de un Byte y la transmisión del siguiente Byte (por parte de la FPGA). Una vez concluidas las pruebas, se comprobó que las imágenes enviadas son las mismas que las imágenes recibidas, por lo tanto, queda completamente validada la funcionalidad de la transmisión y recepción de datos por parte de la FPGA, además de que también se corroboró que el módulo de control de la SRAM funciona correctamente.

Al realizar la prueba del procesamiento de datos se obtienen dos archivos, uno de datos reales y otro de datos imaginarios.

En la Figura 6.15, obtenida de [16], se muestra un ejemplo de la FFT de una imagen. La Figura 6.15a es la imagen original (una vista microscópica de la etapa de entrada de un amplificador 741). La Figura 6.15b muestra la parte real e imaginaria del espectro de frecuencia de esta imagen, ya que el dominio de la frecuencia puede contener valores negativos en los pixeles, los valores en escala de grises están desplazados de tal forma que los pixeles negativos son negros, cero es gris y los positivos se van acercando al blanco. Los componentes de frecuencia baja en una imagen, por lo regular, son más grandes en amplitud que los de frecuencia más alta. Esto explica los pixeles muy claros y oscuros en las cuatro esquinas de la Figura 6.15b.



a)

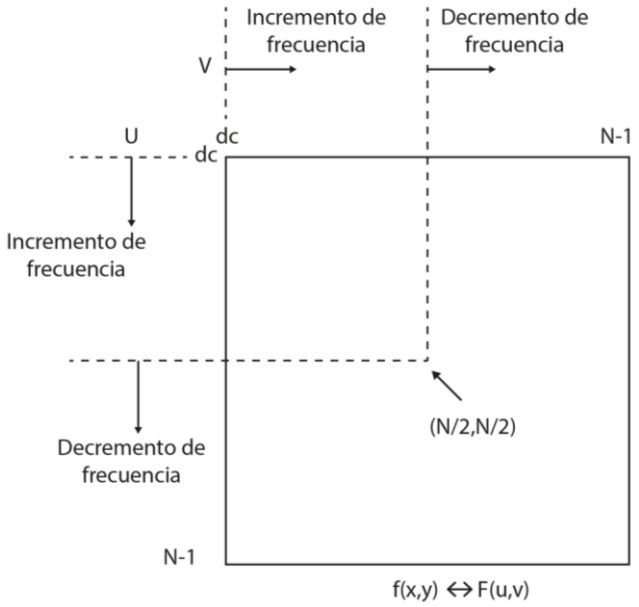


b)

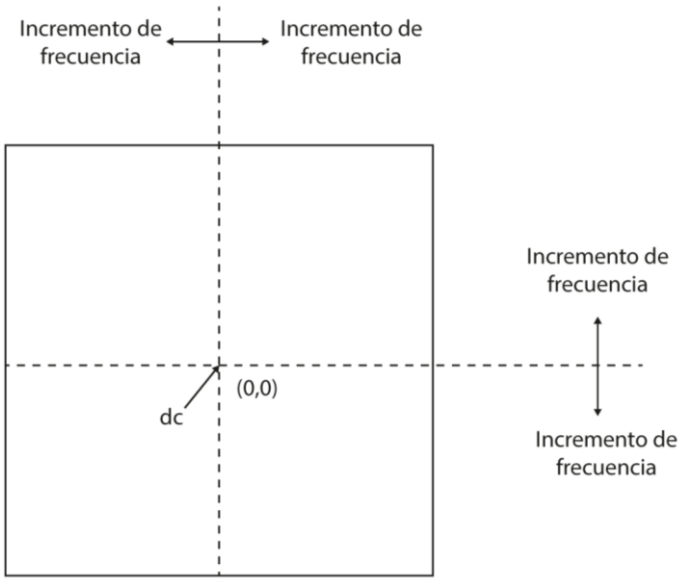
Figura 6.15._ Transformada de Fourier de una imagen, a) Imagen original, b) Parte real e imaginaria de la FFT.

La Figura 6.16a muestra cómo están organizados los datos en el dominio de la frecuencia (con las bajas frecuencias ubicadas en las esquinas). La columna N/2 y el renglón N/2 dividen el espectro en frecuencia en 4 cuadrantes. Para la parte real, el cuadrante superior derecho es una imagen del cuadrante inferior izquierdo, mientras que el cuadrante superior izquierdo es una imagen del cuadrante inferior derecho. Esta simetría también se mantiene para la parte imaginaria, excepto que los valores reflejados de los pixeles tienen el signo contrario. En otras palabras, cada punto en

el espectro de la frecuencia tiene su punto correspondiente ubicado simétricamente al otro lado del centro de la imagen. La Figura 6.16b muestra una forma alternativa para desplegar la FFT.



a)

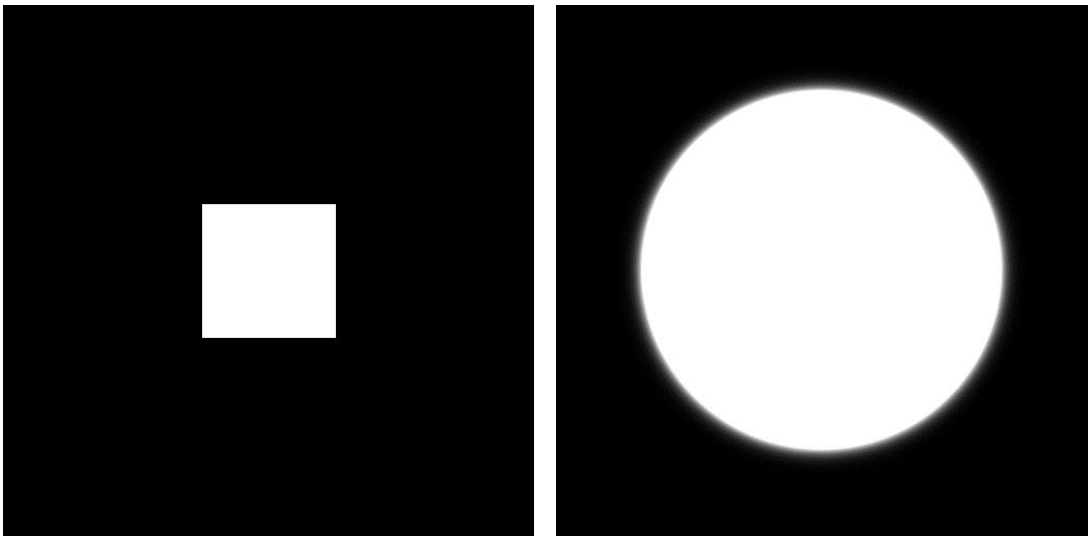


b)

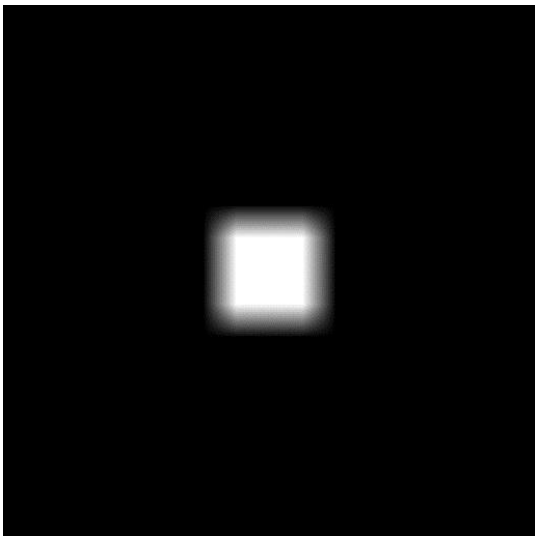
Figura 6.16._ Organización de los datos en el dominio de la frecuencia: a) Organización de los datos al aplicarle el algoritmo de la FFT a una imagen, las frecuencias más bajas se encuentran en las esquinas, b) Otra forma de ordenar los datos, donde las frecuencias más altas se encuentran en las esquinas.

La forma en la que están organizados los datos que se guardan en el archivo con extensión txt (tanto reales como imaginarios) son de la forma que se muestra en la Figura 6.16a.

Las imágenes que se ocuparon para realizar la prueba se despliegan en la Figura 6.17 (a-c). Las tres imágenes tienen una profundidad de intensidad de 8 bits, siendo el 0 para el negro y el 255 para el blanco.



a) b)



c)

Figura 6.17._ Imágenes empleadas para realizar la prueba de procesamiento de datos, a) Abertura cuadrada, b) Abertura gaussiana, d) Abertura con los bordes rampa.

Como se mencionó en la descripción de la prueba de procesamiento, se utilizó un módulo adicional para tomar el tiempo que tarda la FPGA en realizar el procesamiento de datos. En la Tabla 6.2 se muestra la comparación del tiempo experimental con el valor teórico y en la Figura 6.18 se muestra el dato obtenido por el contador, el cual se despliega en 4 *displays* de 7 segmentos. La diferencia de tiempos que existe entre el valor teórico y el experimental se debe a la resolución del contador implementado, la cual es de 1 ms.

Tabla 6.2._ Comparación de tiempo de procesamiento, experimental y teórico.

	Valor teórico [ms]	Valor experimental [ms]
Tiempo de procesamiento	789.88288	790

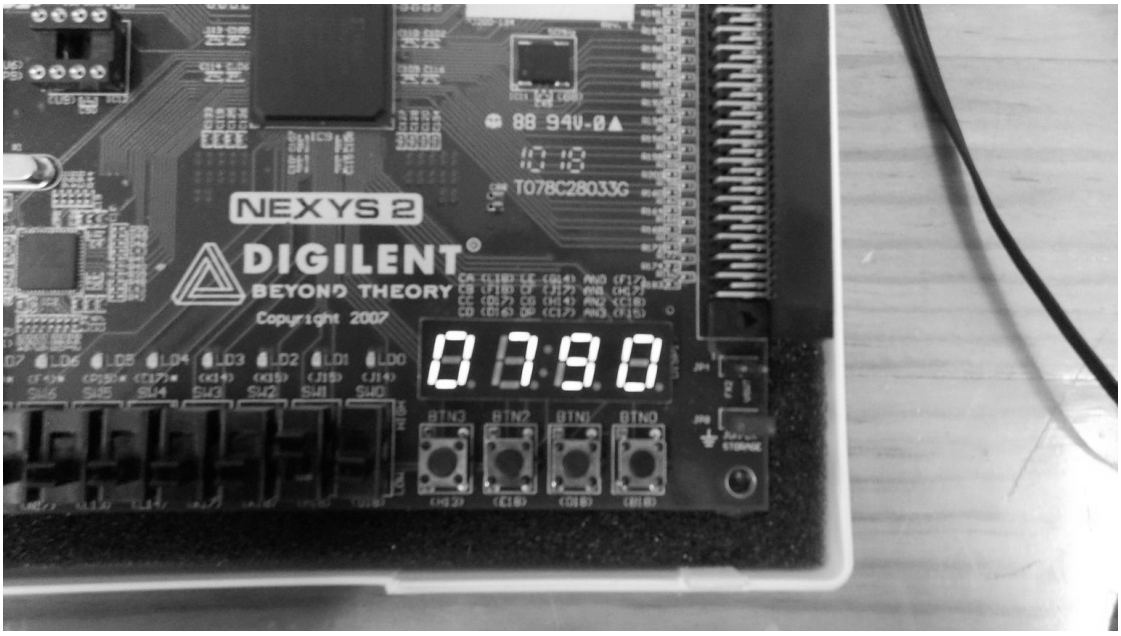
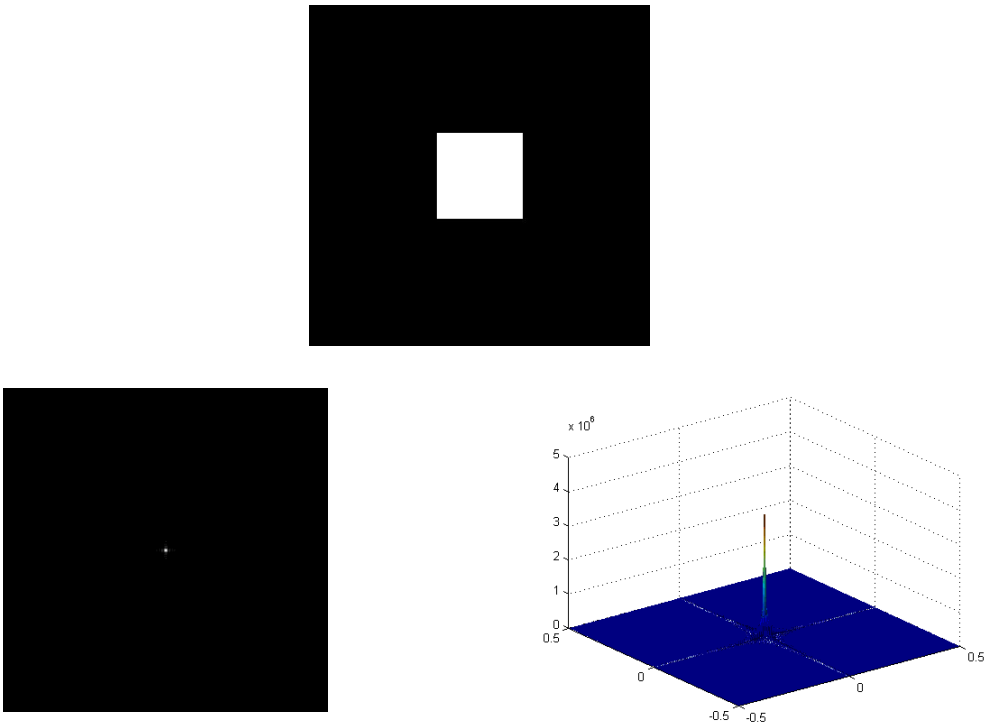
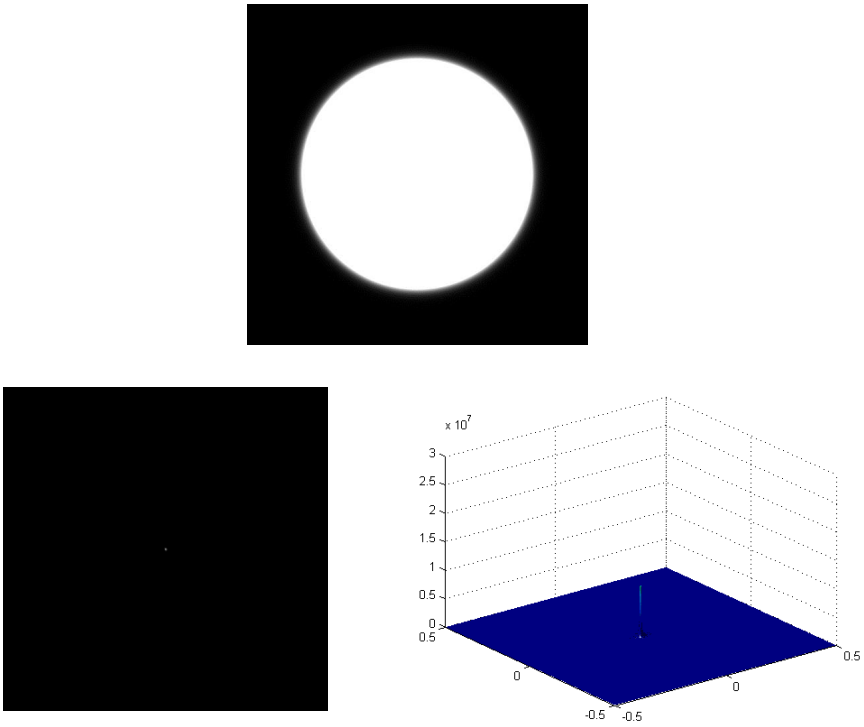


Figura 6.18._ Para realizar la comparación con el tiempo teórico para el procesamiento de datos se implementó un contador, el cual es desplegado en 4 displays de 7 segmentos.

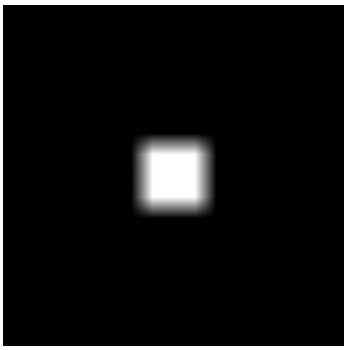
Los datos obtenidos por la prueba de procesamiento (archivos txt de los datos reales e imaginarios) fueron reconstruidos en Matlab®. En la Figura 6.19 se muestra la representación del espectro de la FFT de las imágenes, además de una representación en superficie. Las imágenes mostradas corresponden a la magnitud de los datos reales e imaginarios. Para una mejor apreciación se muestra el espectro centralizado obtenido por el desplazamiento de cuadrantes.

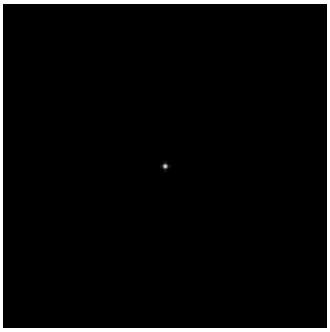


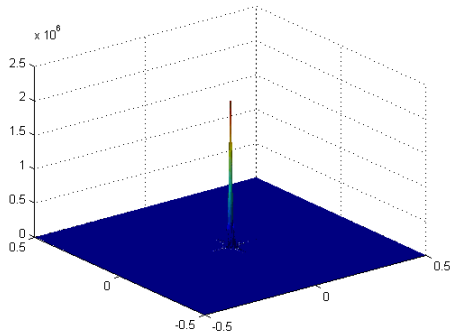
a)



b)







c)

```
FFT_imagen = reales + j*Imaginarios;
S = abs(FFT_imagen);
S1 = mat2gray(S);
figure(1)
imshow(fftshift(S1))
L =512;
M = 512;
dx=L/M;
fx=-1/(2*dx):1/L:1/(2*dx)-(1/L);
fy=fx;
figure(2)
surf(fx,fy,fftshift(S))
camlight left; lighting phong
colormap('jet')
shading interp
```

d)

Figura 6.19._ Representación centralizada de la FFT, a) Abertura cuadrada, b) Abertura gaussiana, c) Abertura con los bordes rampa, d) Código en Matlab® empleado para realizar la representación de la FFT.

Con el fin de realizar una comparación de datos y obtener los errores absoluto y relativo del procesamiento de datos en la FPGA, se tomaron como valores teóricos los datos obtenidos en Matlab® al procesarse las mismas imágenes que en la FPGA. Para tener un valor teórico más exacto en Matlab® se trabajó con datos en punto flotante de doble precisión (64 bits), según lo especificado en el estándar IEEE-754. El error absoluto es la diferencia entre el valor de la medida y el valor tomado como exacto. Puede ser positivo o negativo, según si la medida es superior al valor real o inferior (la resta sale positiva o negativa). El error relativo es el cociente entre el error absoluto y el valor exacto, puede ser positivo o negativo según lo sea el error absoluto. Si se multiplica por 100 se obtiene el error en porcentaje, este error no tiene unidades. Debido a que se está trabajando con 262144 datos no es posible mostrar en la tesis el error de éstos individualmente, por lo tanto se obtuvo un promedio del error. Este promedio es muy representativo para cada dato. El error absoluto es la dispersión de las medidas y el relativo, el porcentaje de esta dispersión respecto al valor total de lo medido. Por lo tanto los errores se obtuvieron para conocer que tanto se alejan los datos obtenidos de los tomados como reales.

Los errores obtenidos para la imagen de la Figura 6.15a son:

Datos reales:

Promedio del error absoluto: 0.0019.

Promedio del error relativo: 1.5680e-04.

Datos imaginarios:

Promedio del error absoluto: 0.0019.

Promedio del error relativo: 4.2999e-09.

Los errores obtenidos para la imagen de la Figura 6.15b son:

Datos reales:

Promedio del error absoluto: 0.0080.

Promedio del error relativo: 0.0082.

Datos imaginarios:

Promedio del error absoluto: 0.0080.

Promedio del error relativo: 0.0062.

Los errores obtenidos para la imagen de la Figura 6.15c son:

Datos reales:

Promedio del error absoluto: 0.0012.

Promedio del error relativo: 0.0269.

Datos imaginarios:

Promedio del error absoluto: 0.0012.

Promedio del error relativo: 0.0126.

El código de Matlab® empleado para obtener los errores se muestra en la Figura 6.20.

```
clear all
close all
clc
%Leemos los datos de la imagen
A = imread('rejilla_rampa-escalon_a.jpg');
B = double (A);
C = fft2(B);
D = real(C);
%Leemos los datos obtenidos de la FPGA
reales_fpga = textread('reales.txt');
%Obtenemos el promedio del error absoluto
error_abs = abs(reales_fpga - D);
disp('Promedio del error absoluto: ')
mean_error_abs = mean(mean(error_abs))
 %obtenemos el promedio del error relativo
[N M] = size(D);
K = N*M;
vector_datos_error = zeros(1,K);
contador_vector = 1;
 for i=1:512
   for j=1:512
      if (D(i,j)) == 0
         vector_datos_error(contador_vector) = 0;
      else
         vector_datos_error(contador_vector) = (reales_fpga(i,j) -
D(i,j))/D(i,j);
      end
      contador_vector = contador_vector + 1;
   end
end
 error_relativo = mean(vector_datos_error);
disp('Promedio del error relativo: ')
error_relativo = abs(error_relativo) * 100
```

Figura 6.20._ Código en Matlab® para la obtención de los errores absoluto y relativo.

El error en los datos se debe a que el *core* de la FFT únicamente implementa el redondeo al valor más cercano, además de que para el factor de fase se utiliza un valor en punto fijo de 25 bits, al contrario de Matlab®, donde para la representación de la fase se utilizan 64 bits (precisión doble en punto flotante). Sin embargo, el error que se presenta es pequeño.

Conclusiones y proyecciones

CONCLUSIONES

Una vez terminadas las pruebas y analizados los resultados que se obtuvieron en éstas, se llegó a las siguientes conclusiones.

1._ La interfaz de comunicación RS-232 para la transmisión de datos no resulta viable para implementarla sobre un sistema embebido en tiempo real, ya que la tasa de transmisión de datos resulta muy lenta (7200 B/s). Se tarda 13 minutos (ver Capitulo 6) para transmitir una imagen procesada de 512x512 pixeles (en total se transmiten desde la FPGA hacia la PC 1,048,576 bytes). Es mejor emplear otros protocolos de comunicación serial, como son el USB o Ethernet, los cuales tienen una tasa de transferencias de datos superior a la ofrecida por RS-232. Para USB 2.0 la velocidad de promedio de transmisión es de 125 MB/s por segundo y para Ethernet es de 12.5 MBs.

2._ El sistema utiliza una memoria SRAM, la cual tiene un acceso de escritura y lectura de 70 ns, por lo que se presenta un retardo considerable en el acceso consecutivo de los datos a la memoria. La mejor opción es ocupar una memoria SDRAM (del inglés *Synchronous Dynamic Random Access Memory*). Con ésta memoria se puede llegar a tener acceso a los datos en cada ciclo de reloj (en el caso ideal de contar con una memoria con un ancho de palabra de 32 bits). Este cambio repercutiría en la disminución del tiempo de procesamiento, el cual se realizaría en aproximadamente 82 ms (con un reloj de 50 MHz), por lo tanto nos ahorraríamos 707.87 ms (789.88 ms – 82 ms).

3._ Es posible reducir el tiempo de procesamiento de la FFT (al aumentar la frecuencia base, por medio de los bloques DCM con los que cuenta el FPGA), ya sea de un renglón o una columna. Es decir, si aumentamos la frecuencia a 100 MHz tendríamos un pulso de reloj de 10 ns, valor que nos permite hacer más eficiente la carta ASM (ver Anexo D) del módulo de control de la SRAM. En el anexo C se muestran las pruebas realizadas al utilizar los bloques DCM para aumentar la frecuencia base, la cual originalmente es de 50 MHz. Sin embargo, por las limitaciones de la memoria SRAM los tiempos de acceso a memoria serían los mismos (los accesos de escritura y lectura a la SRAM son fijos).

4._ El FPGA ocupado para la implementación del sistema, Spartan®-3E, trabaja de forma efectiva para el procesamiento de imágenes menores a 512x512 pixeles. Si se requiere procesar imágenes más grandes se recomienda utilizar un FPGA que mínimo duplique los recursos de la FPGA empleada. Estos recursos son la memoria de bloques (para almacenar los datos parciales en el cálculo de la FFT) y los bloques DSP (para realizar las multiplicaciones y sumas del algoritmo de la FFT).

5._ Se validó completamente que un FPGA cumple con los requisitos para implementar un sistema que nos permite realizar el cálculo de la FFT. Como se mencionó en la Sección 1.2.1.2.2 la FFT puede ser empleada para realizar la recuperación del frente de onda con los datos obtenidos con un sensor de curvatura. Por lo tanto, sí es viable implementar en un FPGA un sistema embebido que esté destinado a realizar la recuperación de un frente de onda, teniendo de esta forma aplicaciones en óptica adaptativa. Sin embargo con el sistema que se implementó existen ciertas restricciones para que en este momento sea útil en los sistemas de óptica adaptativa, tal como se menciona en la Sección 5.7.1.

PROYECCIONES A FUTURO

Antes de continuar con las siguientes etapas de un sistema completo para la recuperación de un frente de onda (ver Figura 1.4 de la sección 1.2.1.2.2, se desarrolló únicamente la etapa del cálculo de la FFT), es necesario realizar un cambio del FPGA, con el fin de superar el ancho de bando que restringe a los sistemas de óptica adaptativa.

Los mayores fabricantes de FPGAs al día de hoy son Xilinx® y Altera®, los cuales hasta el 2008 controlaban cerca del 80% del mercado [17]. Por lo tanto es más fácil conseguir en el mercado un FPGA de esas dos empresas y la elección se centraría en alguno dispositivo que ellos fabrican. Los FPGAs de Xilinx® los podemos separar en 3 grupos. El primer grupo son los FPGAs que están diseñadas para aplicaciones de mayor volumen, donde el rendimiento del sistema no es una prioridad, también conocidos como FPGAs de bajo costo. A este grupo pertenecen los FPGAs Spartan® 3 y 6 y Artix®-7. Los sistemas que funcionan con estos FPGAs, por lo regular trabajan a frecuencias menores a 100 MHz. El segundo grupo abarca las familias de FPGAs de alto rendimiento, y están destinados a aplicaciones que requieren altas velocidades de reloj y operaciones matemáticas extensas, a este grupo pertenecen las familias Virtex®. En el último grupo se encuentra la familia de FPGAs Kintex™, y se puede clasificar como un grupo intermedio entre los FPGAs de alto rendimiento y los de bajo costo. Altera® clasifica sus FPGAs en tres grupos: bajo costo, rango medio y alto rendimiento. Al primer grupo pertenecen la familia de FPGA Cyclone®, al

segundo grupo pertenecen la familia de FPGAs Arria® y al último grupo pertenecen la familia de FPGAs Stratix®.

Los FPGAs de alto rendimiento por lo regular se ocupan en aplicaciones donde los cálculos matemáticos son muy intensos, como por ejemplo en procesamiento de video en tiempo real, además de que se requiere una frecuencia de reloj alta, arriba de los 100 MHz, es decir, aplicaciones que requieren un ancho de banda alto. Los FPGAs de bajo costo son empleados en aplicaciones donde el rendimiento no es una prioridad, y no se requiere de una frecuencia de procesamiento alta, pero sí de aplicaciones de mayor volumen (uso general) sensibles al costo. Los FPGAs de rango medio a bajo son más accesibles que nunca para aplicaciones de uso general.

En la Tabla 1 se muestra una comparación entre 4 familias de FPGAs de Xilinx. Se recomienda utilizar un FPGA de la familia Virtex®-7, ya que éstos FPGAs están diseñados específicamente para aplicaciones de alto rendimiento.

Tabla 1._ Comparación de los FPGAs serie 7 de Xilinx® y Spartan®-6.

	Artix®-7	**Kintex®-7**	**Virtex®-7**	**Spartan®-6**
Celdas lógicas	215K	478K	1955K	150K
BRAM	13 Mb	34 Mb	68 Mb	4.8 Mb
DSP	740	1920	3600	180
DSP performance	929 GMAC/s	2845 GMAC/s	5335 GMAC/s	140 GMAC/s
N° de Transceiver	16	32	96	8
Velocidad de los Transceiver	6.6 Gb/s	12.5 Gb/s	28.05 Gb/s	3.2 Gb/s
Ancho de banda total de los Transceiver (Full dúplex)	211 Gb/s	800 Gb/s	2784 Gb/s	50 Gb/s
Interface de memoria	1066 Mb/s	1866 Mb/s	1866 Mb/s	800 Mb/s
I/O Pins	500	500	1200	576
I/O Voltaje	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V

Al utilizar la FPGA recomendada no estaríamos limitados con el número de DSPs utilizados (ver Sección 5.5.1), por lo tanto se puede implementar cualquier arquitectura del *ip core FFT*. En este caso la limitación del tiempo de procesado está sujeta al reloj que se emplearía. Utilizando un reloj de 200 MHz y aumentando esta frecuencia a 400 MHz tendríamos un tiempo de procesamiento total de la imagen de aproximadamente 10.25 ms (ya que no sería necesario emplear una memoria externa, ya que esta FPGA cuenta con 8.5 MB de memoria RAM de bloque y se necesitan 4.2 MB para el procesamiento de datos). Éste tiempo se produce al implementar la arquitectura Radix-2. En el caso de implementar la arquitectura Radix-4 el tiempo de procesamiento es de 2.9 ms.

En conclusión se recomienda implementar el sistema en un FPGA Virtex®-7 con un reloj de 200 MHz trabajando a una frecuencia de 400 MHz (haciendo uso de los bloques DCM). Y por último implementar la arquitectura Radix-4 para el *ip core FFT*.

Es posible implementar el sistema en FPGAs de rango medio, como es la FPGA Spartan®-6, sin embargo la frecuencia máxima a la que se puede trabajar es de 375 MHz utilizando los bloques DCM. En el caso ideal de tener los datos de las imágenes listos en cada ciclo de reloj, el tiempo de procesamiento sería de 3.13 ms.

Se realizó también una comparación del tiempo de procesamiento de una GPU, CPU y la FPGA, la cual se encuentra en el Anexo E. Con estos datos se puede observar de que el tiempo que le toma a la CPU es de 782.84 ms más rápido y el GPU es 784.65 ms más rápido, ambos con respecto al sistema implementado. Sin embargo si el sistema se implementa en el FPGA Virtex®-7, el tiempo que le lleva a procesar la FFT al CPU y GPU es mayor. La desventaja de utilizar el CPU o el GPU es que si hay procesos externos que ocupen estos procesadores el tiempo de procesamiento aumenta, en la FPGA no ocurre esto, ya que únicamente realiza el procesamiento para el que fue programado.

En general, una vez implementado el sistema para la obtención de la FFT siguiendo las recomendaciones antes mencionadas, en la siguiente etapa del proyecto se tiene como meta implementar las etapas restantes (ver Figura 1.12 de la Sección 1.2.1.2.2) y de esta manera tener un verdadero sistema embebido en tiempo real, el cual ya podría ser implementado en un sistema de óptica adaptativa.

Anexos

Anexo A

Hoja de datos de la FPGA Spartan®-3E

XILINX®

Spartan-3E FPGA Family: Introduction and Ordering Information

DS312-1 (v3.8) August 26, 2009

Product Specification

Introduction

The Spartan®-3E family of Field-Programmable Gate Arrays (FPGAs) is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The five-member family offers densities ranging from 100,000 to 1.6 million system gates, as shown in Table 1.

The Spartan-3E family builds on the success of the earlier Spartan-3 family by increasing the amount of logic per I/O, significantly reducing the cost per logic cell. New features improve system performance and reduce the cost of configuration. These Spartan-3E FPGA enhancements, combined with advanced 90 nm process technology, deliver more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry.

Because of their exceptionally low cost, Spartan-3E FPGAs are ideally suited to a wide range of consumer electronics applications, including broadband access, home networking, display/projection, and digital television equipment.

The Spartan-3E family is a superior alternative to mask programmed ASICs. FPGAs avoid the high initial cost, the lengthy development cycles, and the inherent inflexibility of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary, an impossibility with ASICs.

Features

- Very low cost, high-performance logic solution for high-volume, consumer-oriented applications
- Proven advanced 90-nanometer process technology
- Multi-voltage, multi-standard SelectIO™ interface pins
  - Up to 376 I/O pins or 156 differential signal pairs
  - LVCMOS, LVTTL, HSTL, and SSTL single-ended signal standards
  - 3.3V, 2.5V, 1.8V, 1.5V, and 1.2V signaling
  - 622+ Mb/s data transfer rate per I/O
  - True LVDS, RSDS, mini-LVDS, differential HSTL/SSTL differential I/O
  - Enhanced Double Data Rate (DDR) support
  - DDR SDRAM support up to 333 Mb/s
- Abundant, flexible logic resources
  - Densities up to 33,192 logic cells, including optional shift register or distributed RAM support
  - Efficient wide multiplexers, wide logic
  - Fast look-ahead carry logic
  - Enhanced 18 x 18 multipliers with optional pipeline
  - IEEE 1149.1/1532 JTAG programming/debug port
- Hierarchical SelectRAM™ memory architecture
  - Up to 648 Kbits of fast block RAM
  - Up to 231 Kbits of efficient distributed RAM
- Up to eight Digital Clock Managers (DCMs)
  - Clock skew elimination (delay locked loop)
  - Frequency synthesis, multiplication, division
  - High-resolution phase shifting
  - Wide frequency range (5 MHz to over 300 MHz)
- Eight global clocks plus eight additional clocks per each half of device, plus abundant low-skew routing
- Configuration interface to industry-standard PROMs
  - Low-cost, space-saving SPI serial Flash PROM
  - x8 or x8/x16 parallel NOR Flash PROM
  - Low-cost Xilinx® Platform Flash with JTAG
- Complete Xilinx ISE® and WebPACK™ software
- MicroBlaze™ and PicoBlaze™ embedded processor cores
- Fully compliant 32-/64-bit 33 MHz PCI support (66 MHz in some devices)
- Low-cost QFP and BGA packaging options
  - Common footprints support easy density migration
  - Pb-free packaging options
- XA Automotive version available

Table 1: **Summary of Spartan-3E FPGA Attributes**

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)				Distributed RAM bits(1)	Block RAM bits(1)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs	Total Slices						
XC3S100E	100K	2,160	22	16	240	960	15K	72K	4	2	108	40
XC3S250E	250K	5,508	34	26	612	2,448	38K	216K	12	4	172	68
XC3S500E	500K	10,476	46	34	1,164	4,656	73K	360K	20	4	232	92
XC3S1200E	1200K	19,512	60	46	2,168	8,672	136K	504K	28	8	304	124
XC3S1600E	1600K	33,192	76	58	3,688	14,752	231K	648K	36	8	376	156

Notes:
1. By convention, one Kb is equivalent to 1,024 bits.

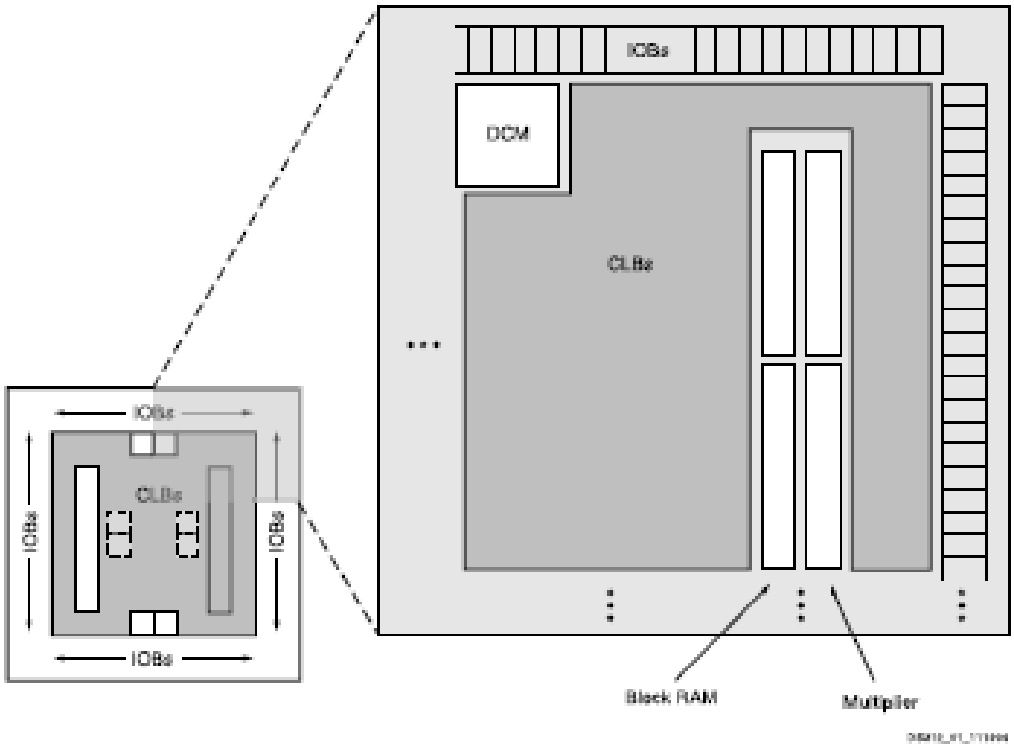
Architectural Overview

The Spartan-3E family architecture consists of five fundamental programmable functional elements:

- **Configurable Logic Blocks (CLBs)** contain flexible Look-Up Tables (LUTs) that implement logic plus storage elements used as flip-flops or latches. CLBs perform a wide variety of logical functions as well as store data.
- **Input/Output Blocks (IOBs)** control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. Supports a variety of signal standards, including four high-performance differential standards. Double Data-Rate (DDR) registers are included.
- **Block RAM** provides data storage in the form of 18-Kbit dual-port blocks.
- **Multiplier Blocks** accept two 18-bit binary numbers as inputs and calculate the product.
- **Digital Clock Manager (DCM) Blocks** provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase-shifting clock signals.

These elements are organized as shown in Figure 1. A ring of IOBs surrounds a regular array of CLBs. Each device has two columns of block RAM except for the XC3S100E, which has one column. Each RAM column consists of several 18-Kbit RAM blocks. Each block RAM is associated with a dedicated multiplier. The DCMs are positioned in the center with two at the top and two at the bottom of the device. The XC3S100E has only one DCM at the top and bottom, while the XC3S1200E and XC3S1600E add two DCMs in the middle of the left and right sides.

The Spartan-3E family features a rich network of traces that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.



Notes:

1. The XC3S1200E and XC3S1600E have two additional DCMs on both the left and right sides as indicated by the dashed lines. The XC3S100E has only one DCM at the top and one at the bottom.

Figure 1: **Spartan-3E Family Architecture**

Configuration

Spartan-3E FPGAs are programmed by loading configuration data into robust, reprogrammable, static CMOS configuration latches (CCLs) that collectively control all functional elements and routing resources. The FPGA's configuration data is stored externally in a PROM or some other non-volatile medium, either on or off the board. After applying power, the configuration data is written to the FPGA using any of seven different modes:

- Master Serial from a Xilinx Platform Flash PROM
- Serial Peripheral Interface (SPI) from an industry-standard SPI serial Flash
- Byte Peripheral Interface (BPI) Up or Down from an industry-standard x8 or x8/x16 parallel NOR Flash
- Slave Serial, typically downloaded from a processor
- Slave Parallel, typically downloaded from a processor
- Boundary Scan (JTAG), typically downloaded from a processor or system tester.

Furthermore, Spartan-3E FPGAs support MultiBoot configuration, allowing two or more FPGA configuration bitstreams to be stored in a single parallel NOR Flash. The FPGA application controls which configuration to load next and when to load it.

I/O Capabilities

The Spartan-3E FPGA SelectIO interface supports many popular single-ended and differential standards. Table 2 shows the number of user I/Os as well as the number of differential I/O pairs available for each device/package combination.

Spartan-3E FPGAs support the following single-ended standards:

- 3.3V low-voltage TTL (LVTTL)
- Low-voltage CMOS (LVCMOS) at 3.3V, 2.5V, 1.8V, 1.5V, or 1.2V
- 3V PCI at 33 MHz, and in some devices, 66 MHz
- HSTL I and III at 1.8V, commonly used in memory applications
- SSTL I at 1.8V and 2.5V, commonly used for memory applications

Spartan-3E FPGAs support the following differential standards:

- LVDS
- Bus LVDS
- mini-LVDS
- RSDS
- Differential HSTL (1.8V, Types I and III)
- Differential SSTL (2.5V and 1.8V, Type I)
- 2.5V LVPECL inputs

Table 2: **Available User I/Os and Differential (Diff) I/O Pairs**

Package	VQ100 VQG100		CP132 CPG132		TQ144 TQG144		PQ208 PQG208		FT256 FTG256		FG320 FGG320		FG400 FGG400		FG484 FGG484	
Size (mm)	16 x 16		8 x 8		22 x 22		28 x 28		17 x 17		19 x 19		21 x 21		23 x 23	
Device	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff
XC3S100E	66 *(7)*	30 *(2)*	83 *(11)*	35 *(2)*	108 *(28)*	40 *(4)*	-	-	-	-	-	-	-	-	-	-
XC3S250E	66 *(7)*	30 *(2)*	92 *(7)*	41 *(2)*	108 *(28)*	40 *(4)*	158 *(32)*	65 *(5)*	172 *(40)*	68 *(8)*	-	-	-	-	-	-
XC3S500E	66[3] *(7)*	30 *(2)*	92 *(7)*	41 *(2)*	-	-	158 *(32)*	65 *(5)*	190 *(41)*	77 *(8)*	232 *(56)*	92 *(12)*	-	-	-	-
XC3S1200E	-	-	-	-	-	-	-	-	190 *(40)*	77 *(8)*	250 *(56)*	99 *(12)*	304 *(72)*	124 *(20)*	-	-
XC3S1600E	-	-	-	-	-	-	-	-	-	-	250 *(56)*	99 *(12)*	304 *(72)*	124 *(20)*	376 *(82)*	156 *(21)*

Notes:

1. All Spartan-3E devices provided in the same package are pin-compatible as further described in Module 4: Pinout Descriptions.
2. The number shown in **bold** indicates the maximum number of I/O and input-only pins. The number shown in (italics) indicates the number of input-only pins.
3. The XC3S500E is available in the VQG100 Pb-free package and not the standard VQ100. The VQG100 and VQ100 pin-outs are identical and general references to the VQ100 will apply to the XC3S500E.

Package Marking

Figure 2 provides a top marking example for Spartan-3E FPGAs in the quad-flat packages. Figure 3 shows the top marking for Spartan-3E FPGAs in BGA packages except the 132-ball chip-scale package (CP132 and CPG132). The markings for the BGA packages are nearly identical to those for the quad-flat packages, except that the marking is rotated with respect to the ball A1 indicator. Figure 4 shows the top marking for Spartan-3E FPGAs in the CP132 and CPG132 packages.

On the QFP and BGA packages, the optional numerical Stepping Code follows the Lot Code.

The "5C" and "4I" part combinations can have a dual mark of "5C/4I". Devices with a single mark are only guaranteed for the marked speed grade and temperature range. All "5C" and "4I" part combinations use the Stepping 1 production silicon.

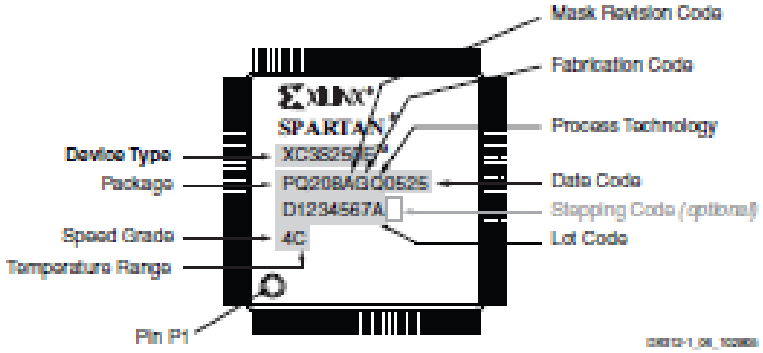


Figure 2: **Spartan-3E QFP Package Marking Example**

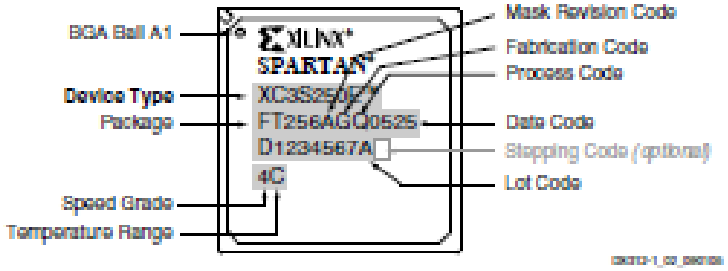


Figure 3: **Spartan-3E BGA Package Marking Example**

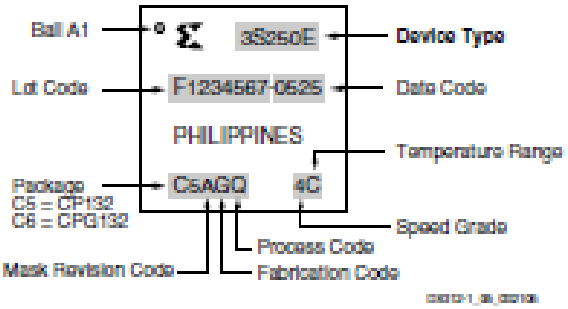


Figure 4: **Spartan-3E CP132 and CPG132 Package Marking Example**

Anexo B

Diagramas de comparación de tiempo en el procesado de la FFT en la FPGA.

ARQUITECTURAS DEL IP CORE

El IP core *FFT LogicCore™* provee cuatro arquitecturas diferentes para la implementación de la FFT:

1) Pipelined, streaming I/O – Ofrece un procesamiento de datos continuo.
2) Radix-4, Burst I/O – Carga y procesa los datos por separado utilizando un método iterativo, basado en el algoritmo Radix-4.
3) Radix-2, Burst I/O – Utiliza el mismo método iterativo que Radix-4, pero la mariposa (implementación del algoritmo) es más pequeña. Por lo que utiliza menos recursos que Radix-4. Basado en el algoritmo Radix-2.
4) Radix-2 Lite, Burst I/O – Basado en la arquitectura Radix-2, utiliza menos recursos que la arquitectura Radix-2, esto lo hace compartiendo recursos los cuales son seleccionados por medio de multiplexores, todo esto a costa de utilizar más tiempo en hacer el procesamiento.

En la Figura B.1, se ilustra la gráfica de rendimiento vs recursos para las diversas arquitecturas. De acuerdo con dicha gráfica, la arquitectura pipelined nos ofrece un mejor rendimiento, pero a costa de ocupar un mayor número de recursos. La arquitectura Radix-2 Lite es la que nos ofrece el peor rendimiento pero es la que ocupa menos recursos de la FPGA. Entre cada arquitectura se presenta aproximadamente una diferencia de un factor de 2 en uso de recursos, esta relación se presenta siempre y cuando se utilicen secuencias de datos que sean potencias de 2.

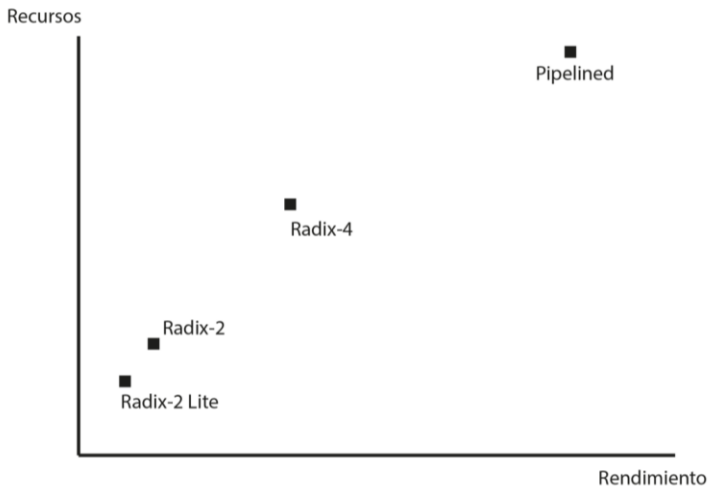


Figura B.1._ Recursos vs Rendimiento para las diferentes arquitecturas de implementación.

ARQUITECTURA PIPELINED

Esta arquitectura implementa varias mariposas del tipo Radix-2, con el fin de poder ofrecer un flujo constante de procesamiento de datos. Cada etapa de procesamiento tiene su propio banco de memoria para almacenar los datos de entrada y los datos intermedios (Figura B.2).

Esta arquitectura tiene la posibilidad de realizar simultáneamente: el cálculo de la trama actual de datos, cargar los datos de entrada para la siguiente trama de datos y descargar el resultado de la trama anterior de datos, por lo que el usuario puede cargar datos continuamente en el *core*, y después de la latencia de la descarga de los primeros datos, descargar continuamente los datos. Para esta arquitectura los datos de entrada necesariamente deben estar en un orden natural (1, 2, 3, ….) y los datos de salida pueden seleccionarse entre ser mostrados en orden natural o en orden inverso. Esta arquitectura acepta secuencias de 8 a 65536 datos. El usuario tiene la posibilidad de seleccionar el número de etapas que desea utilizar.

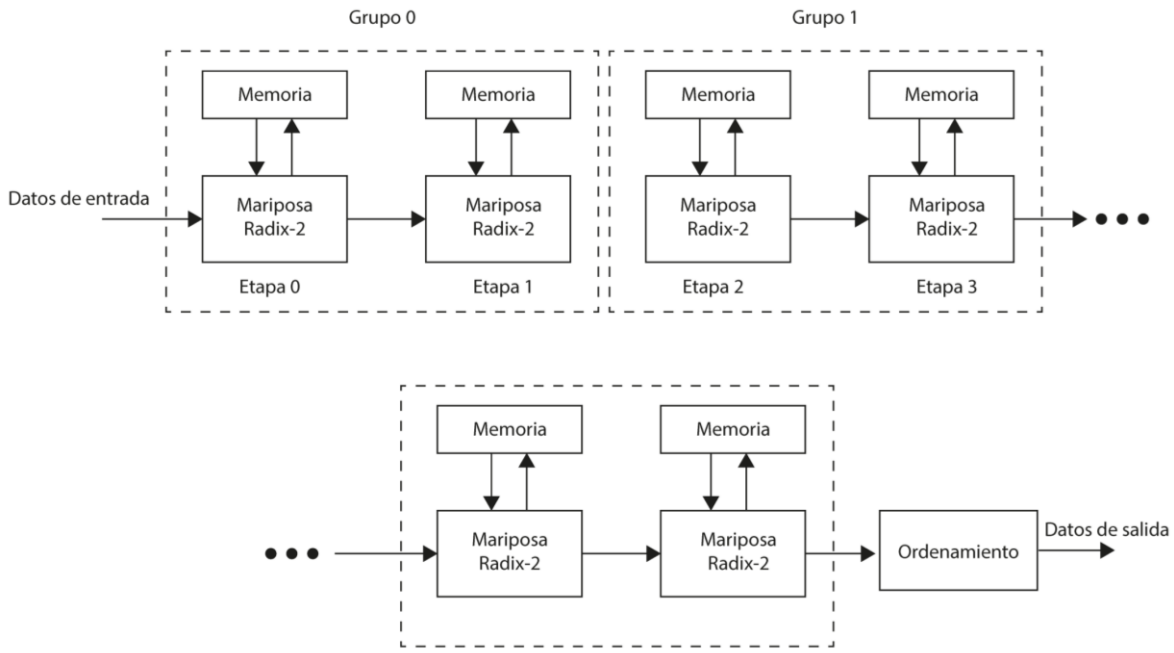


Figura B.2._ Arquitectura Pipelined.

ARQUITECTURA RADIX-4

A diferencia de la arquitectura pipelined, en esta arquitectura sólo se puede efectuar el procesamiento de las secuencias de datos una a la vez, es decir para poder realizar una nueva operación, hay que esperar a que todos los datos de la operación anterior (el resultado de la FFT) sean descargados del *core FFT*. En la Figura B.3 se ilustra el diagrama de bloques de esta arquitectura.

Una vez que el *core* es activado, los datos de entrada son cargados. Una vez que la trama de datos termina de cargarse, el *core* calcula la transformada. Cuando el cálculo ha terminado, los datos ya pueden ser descargados. Esta arquitectura ocupa menos recursos que la arquitectura pipelined, pero tiene un tiempo de procesamiento mucho mayor. Soporta secuencias de 64 a 65536 datos. Tanto los datos como los factores de fase pueden ser almacenados en la memoria RAM distribuida o en la memoria RAM de bloques.

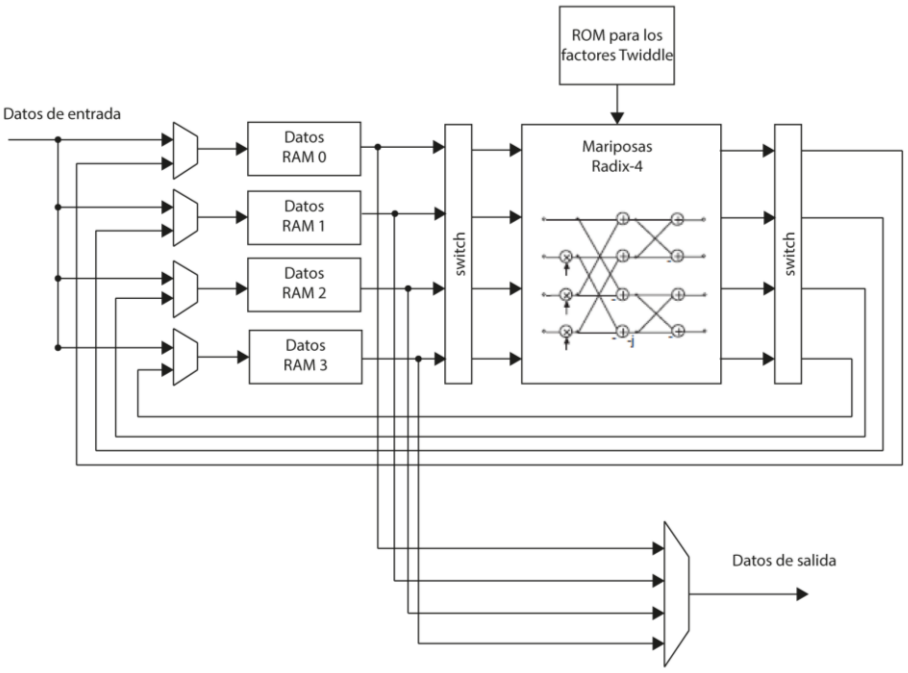


Figura B.3._ Arquitectura Radix-4

ARQUITECTURA RADIX-2

El diagrama de bloques de esta arquitectura (la cual se basa en el algoritmo Radix-2) se ilustra en la Figura B.4.

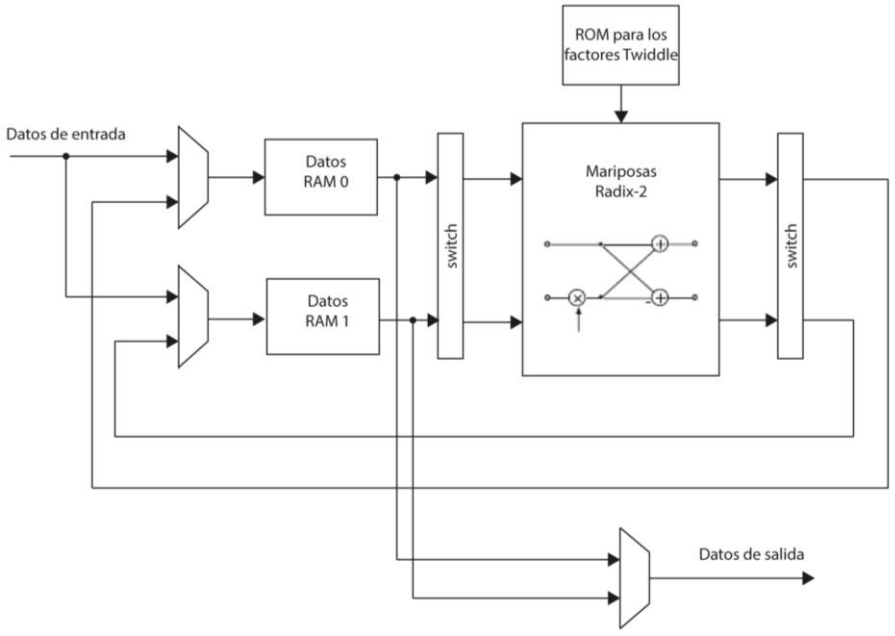


Figura B.4._ Arquitectura Radix-2

Una vez que se cargan los datos de entrada en el *core*, éste procede a efectuar el cálculo de la FFT, cuando termina el cálculo, se procede a descargar los resultados. Se pueden cargar nuevos datos una vez que el proceso anterior esté completado. Esta arquitectura acepta de 8 a 65536 secuencias de datos. Los datos y el factor de fase pueden almacenarse tanto en la memoria RAM distribuida como en la memoria RAM de bloques. Esta arquitectura consume menos recursos que la arquitectura Radix-4, pero el cálculo de la transformada es más lento.

ARQUITECTURA RADIX-2 LITE

Esta arquitectura también emplea el algoritmo Radix-2. La diferencia con la arquitectura anterior es que en la etapa del cálculo de la mariposa todos los datos comparten un único sumador/restador, como consecuencia de ésto, se ocupan menos recursos que con respecto a la arquitectura anterior. En la Figura B.5 se ilustra el diagrama de bloques de la arquitectura.

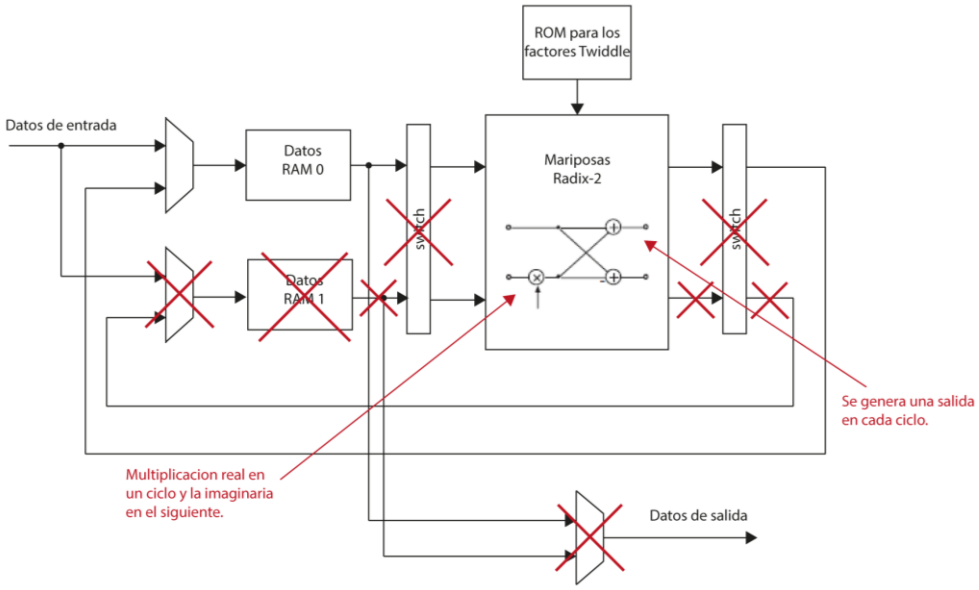
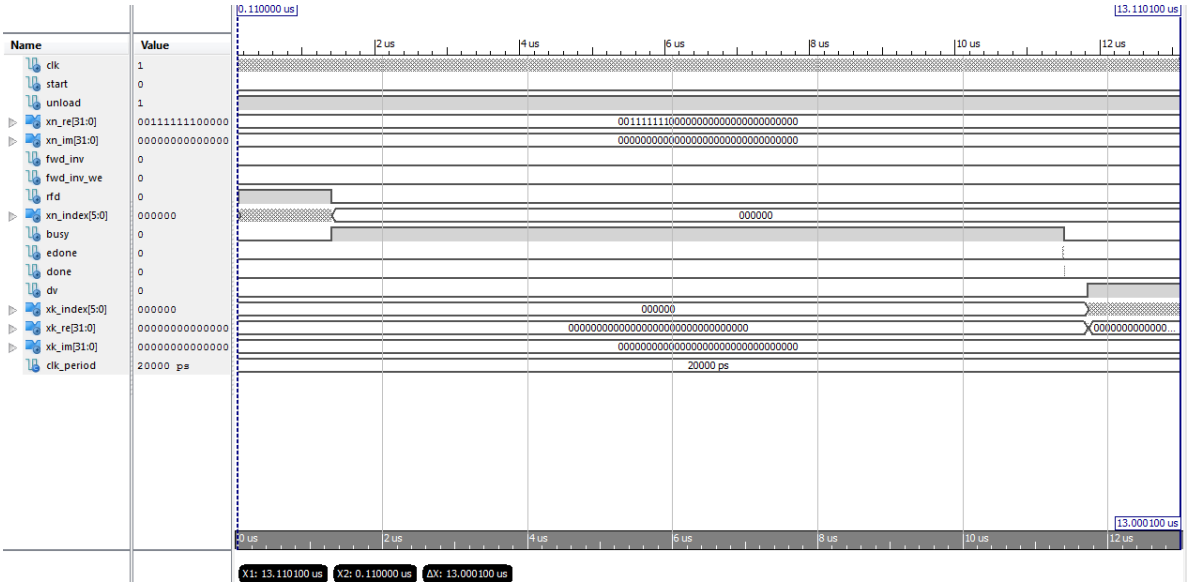


Figura B.5._ Arquitectura Radix-2 Lite. Los componentes tachados se descartan en esta arquitectura, lo que la hace consumir menos recursos lógicos.

Para definir el tipo de arquitectura empleada para el sistema, se procedió a simular el ip core de la FFT empleando las distintas arquitecturas (Radix 2 Lite, Radix 2 y Radix 4), con el fin de obtener un estimado de los recursos empleados y el tiempo que le lleva realizar la FFT para 64, 128, 256 y 512 puntos. A continuación se muestra el diagrama de tiempos de cada arquitectura, seguido por dos tablas, de las cuales la primera contiene los tiempos de procesamiento y la segunda los recursos empleados para la FPGA Spartan®-3E 1200.

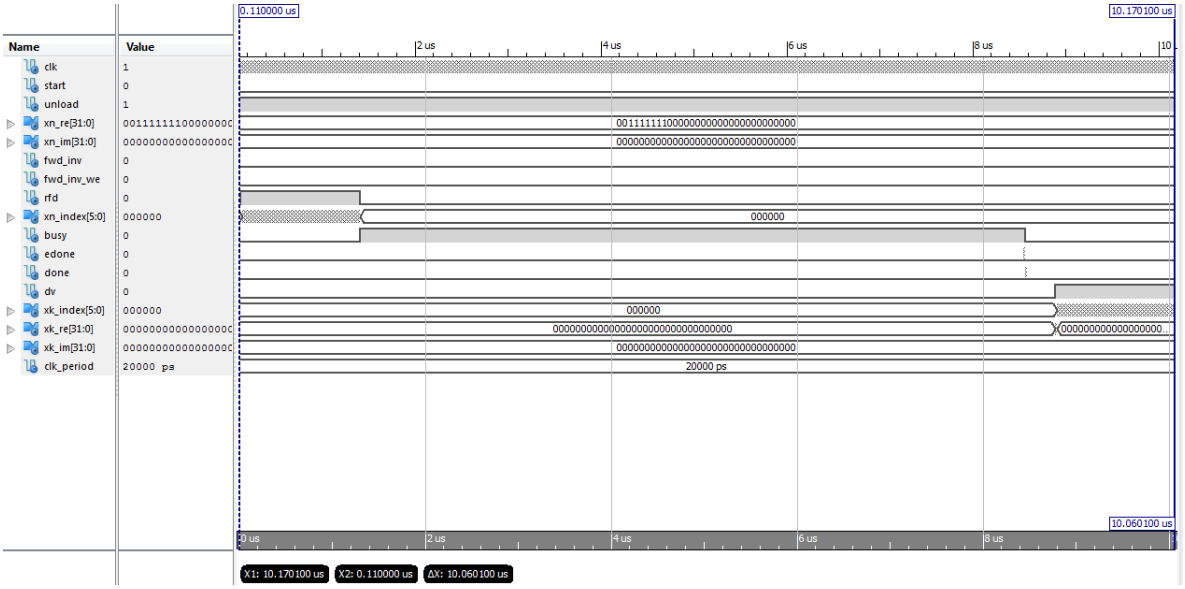
FFT Radix 2 Lite de 64 puntos



Tiempo [us]	Descripción
13.000100	Tiempo transcurrido desde que se pone en 1 la señal *start* hasta que obtenemos el último dato en el bus de salida.
10.100000	Tiempo durante el cual se mantiene en alto la señal *busy*, tiempo que la FPGA tarda en procesar los datos.
1.280000	Tiempo de lectura de los datos reales e imaginarios.
1.280000	Tiempo de despliegue del resultado, datos reales e imaginarios.

Recursos	Usados	Disponibles	Utilización
Numero de Slices	2038	8672	23%
Numero de Slice Flip-Flop	3190	17344	18%
Numero de LUTs de 4 entradas	2615	17344	15%
Numero de IOBs	150	250	60%
Numero de BRAMs	4	28	14%
Numero de Multiplicadores	8	28	28%
Numero de GCLKs	1	24	4%

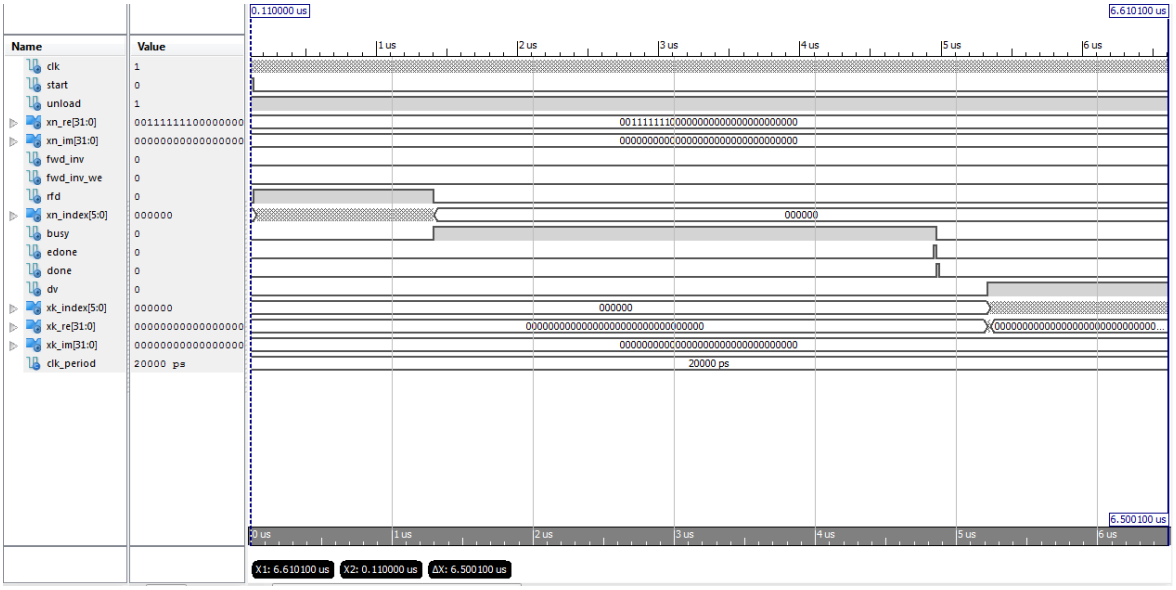
FFT Radix 2 de 64 puntos



Tiempo [us]	Descripción
10.060100	Tiempo transcurrido desde que se pone en 1 la señal *start* hasta que obtenemos el último dato en el bus de salida.
7.160000	Tiempo durante el cual se mantiene en alto la señal *busy*, tiempo que la FPGA tarda en procesar los datos.
1.280000	Tiempo de lectura de los datos reales e imaginarios.
1.280000	Tiempo de despliegue del resultado, datos reales e imaginarios.

Recursos	Usados	Disponibles	Utilización
Numero de Slices	2443	8672	28%
Numero de Slice Flip-Flop	3769	17344	21%
Numero de LUTs de 4 entradas	3331	17344	19%
Numero de IOBs	150	250	60%
Numero de BRAMs	6	28	21%
Numero de Multiplicadores	16	28	57%
Numero de GCLKs	1	24	4%

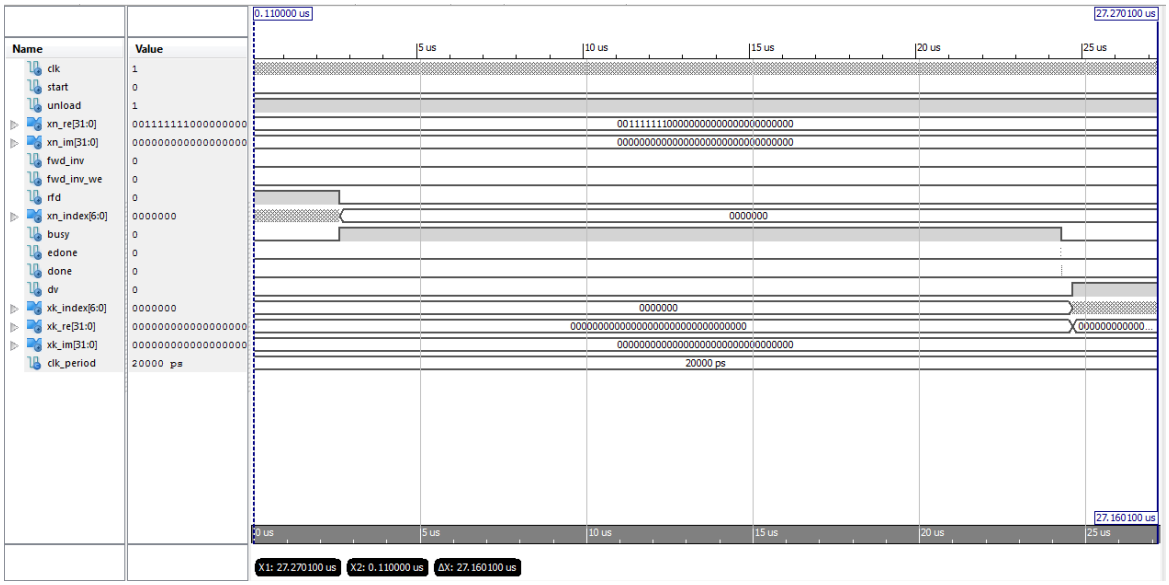
FFT Radix 4 de 64 puntos



Tiempo [us]	Descripción
6.500100	Tiempo transcurrido desde que se pone en 1 la señal *start* hasta que obtenemos el último dato en el bus de salida.
3.560000	Tiempo durante el cual se mantiene en alto la señal *busy*, tiempo que la FPGA tarda en procesar los datos.
1.280000	Tiempo de lectura de los datos reales e imaginarios.
1.280000	Tiempo de despliegue del resultado, datos reales e imaginarios.

Recursos	Usados	Disponibles	Utilización
Numero de Slices	5053	8672	58%
Numero de Slice Flip-Flop	7793	17344	44%
Numero de LUTs de 4 entradas	6609	17344	38%
Numero de IOBs	150	250	60%
Numero de BRAMs	10	28	35%
Numero de Multiplicadores	48	28	171%
Numero de GCLKs	1	24	4%

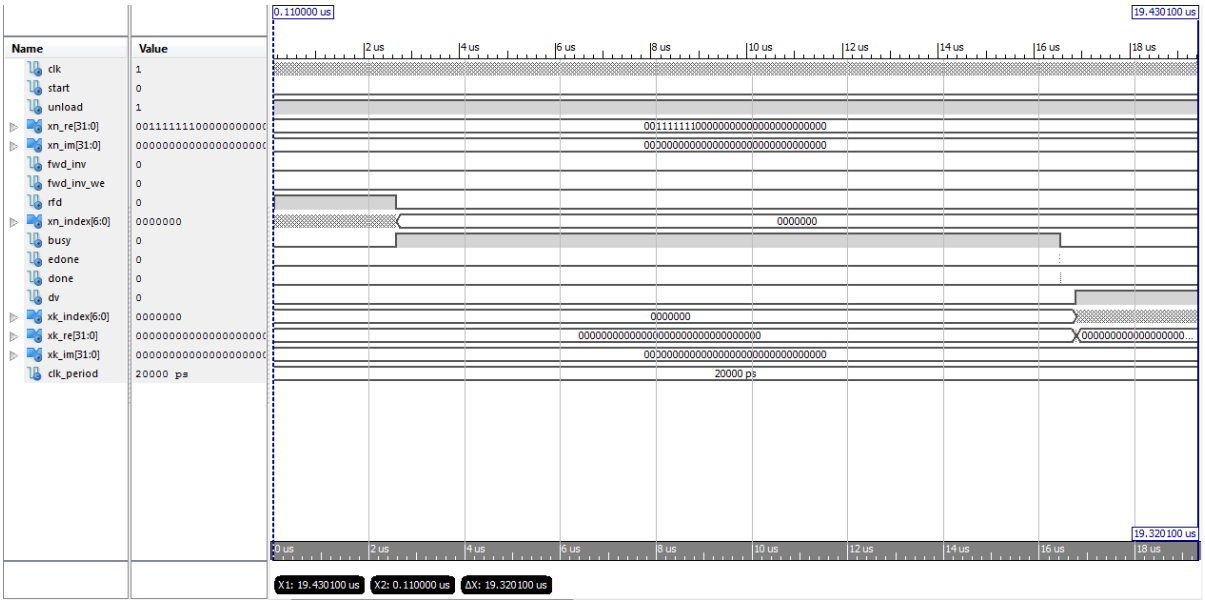
FFT Radix 2 Lite de 128 puntos



Tiempo [us]	Descripción
27.160100	Tiempo transcurrido desde que se pone en 1 la señal *start* hasta que obtenemos el último dato en el bus de salida.
21.700000	Tiempo durante el cual se mantiene en alto la señal *busy*, tiempo que la FPGA tarda en procesar los datos.
2.560000	Tiempo de lectura de los datos reales e imaginarios.
2.560000	Tiempo de despliegue del resultado, datos reales e imaginarios.

Recursos	Usados	Disponibles	Utilización
Numero de Slices	2068	8672	23%
Numero de Slice Flip-Flop	3214	17344	18%
Numero de LUTs de 4 entradas	2661	17344	15%
Numero de IOBs	152	250	60%
Numero de BRAMs	4	28	14%
Numero de Multiplicadores	8	28	28%
Numero de GCLKs	1	24	4%

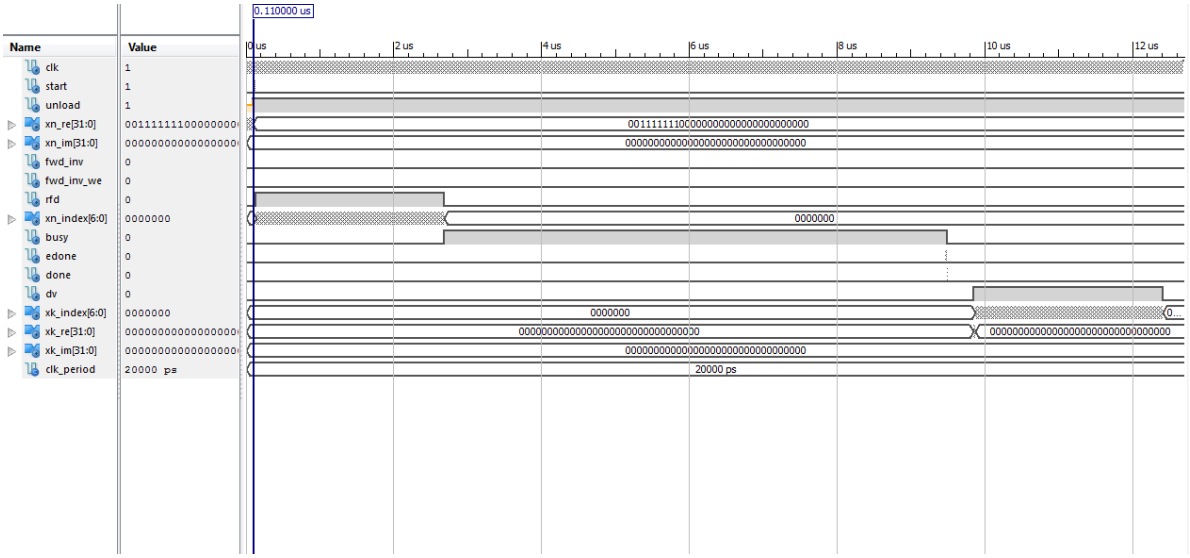
FFT Radix 2 de 128 puntos



Tiempo [us]	Descripción
19.320100	Tiempo transcurrido desde que se pone en 1 la señal *start* hasta que obtenemos el último dato en el bus de salida.
13.860000	Tiempo durante el cual se mantiene en alto la señal *busy*, tiempo que la FPGA tarda en procesar los datos.
2.560000	Tiempo de lectura de los datos reales e imaginarios.
2.560000	Tiempo de despliegue del resultado, datos reales e imaginarios.

Recursos	Usados	Disponibles	Utilización
Numero de Slices	2508	8672	28%
Numero de Slice Flip-Flop	3794	17344	21%
Numero de LUTs de 4 entradas	3455	17344	19%
Numero de IOBs	152	250	60%
Numero de BRAMs	6	28	21%
Numero de Multiplicadores	16	28	57%
Numero de GCLKs	1	24	4%

FFT Radix 4 de 128 puntos



Tiempo [us]	Descripción
12.300100	Tiempo transcurrido desde que se pone en 1 la señal *start* hasta que obtenemos el último dato en el bus de salida.
6.800000	Tiempo durante el cual se mantiene en alto la señal *busy*, tiempo que la FPGA tarda en procesar los datos.
2.560000	Tiempo de lectura de los datos reales e imaginarios.
2.560000	Tiempo de despliegue del resultado, datos reales e imaginarios.

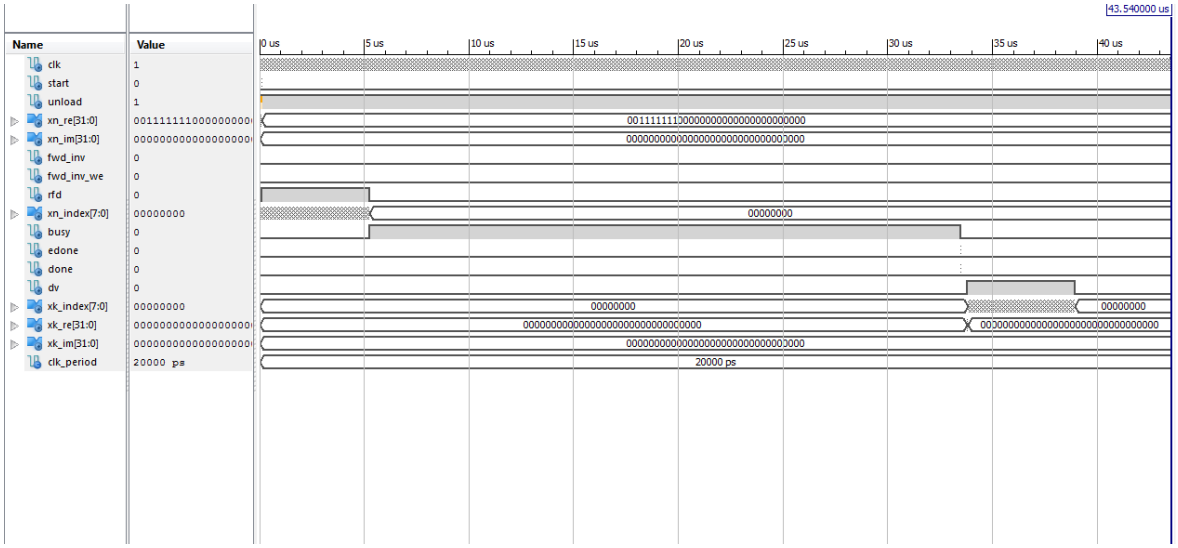
Recursos	Usados	Disponibles	Utilización
Numero de Slices	5209	8672	60%
Numero de Slice Flip-Flop	7867	17344	45%
Numero de LUTs de 4 entradas	6889	17344	39%
Numero de IOBs	152	250	60%
Numero de BRAMs	10	28	35%
Numero de Multiplicadores	48	28	171%
Numero de GCLKs	1	24	4%

FFT Radix 2 Lite de 256 puntos

Tiempo [us]	Descripción
57.960100	Tiempo transcurrido desde que se pone en 1 la señal *start* hasta que obtenemos el último dato en el bus de salida.
47.380000	Tiempo durante el cual se mantiene en alto la señal *busy*, tiempo que la FPGA tarda en procesar los datos.
5.120000	Tiempo de lectura de los datos reales e imaginarios.
5.120000	Tiempo de despliegue del resultado, datos reales e imaginarios.

Recursos	Usados	Disponibles	Utilización
Numero de Slices	2068	8672	23%
Numero de Slice Flip-Flop	3239	17344	18%
Numero de LUTs de 4 entradas	2762	17344	15%
Numero de IOBs	154	250	61%
Numero de BRAMs	4	28	14%
Numero de Multiplicadores	8	28	28%
Numero de GCLKs	1	24	4%

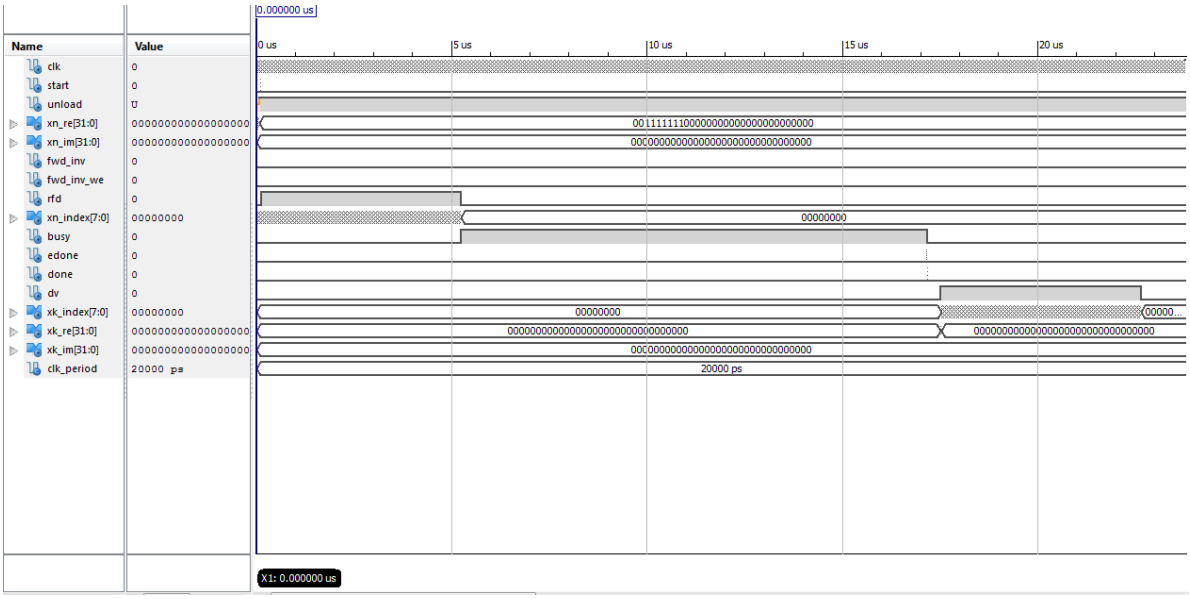
FFT Radix 2 de 256 puntos



Tiempo [us]	Descripción
38.820100	Tiempo transcurrido desde que se pone en 1 la señal *start* hasta que obtenemos el último dato en el bus de salida.
28.240000	Tiempo durante el cual se mantiene en alto la señal *busy*, tiempo que la FPGA tarda en procesar los datos.
5.120000	Tiempo de lectura de los datos reales e imaginarios.
5.120000	Tiempo de despliegue del resultado, datos reales e imaginarios.

Recursos	Usados	Disponibles	Utilización
Numero de Slices	2609	8672	30%
Numero de Slice Flip-Flop	3855	17344	22%
Numero de LUTs de 4 entradas	3627	17344	20%
Numero de IOBs	154	250	61%
Numero de BRAMs	6	28	21%
Numero de Multiplicadores	16	28	57%
Numero de GCLKs	1	24	4%

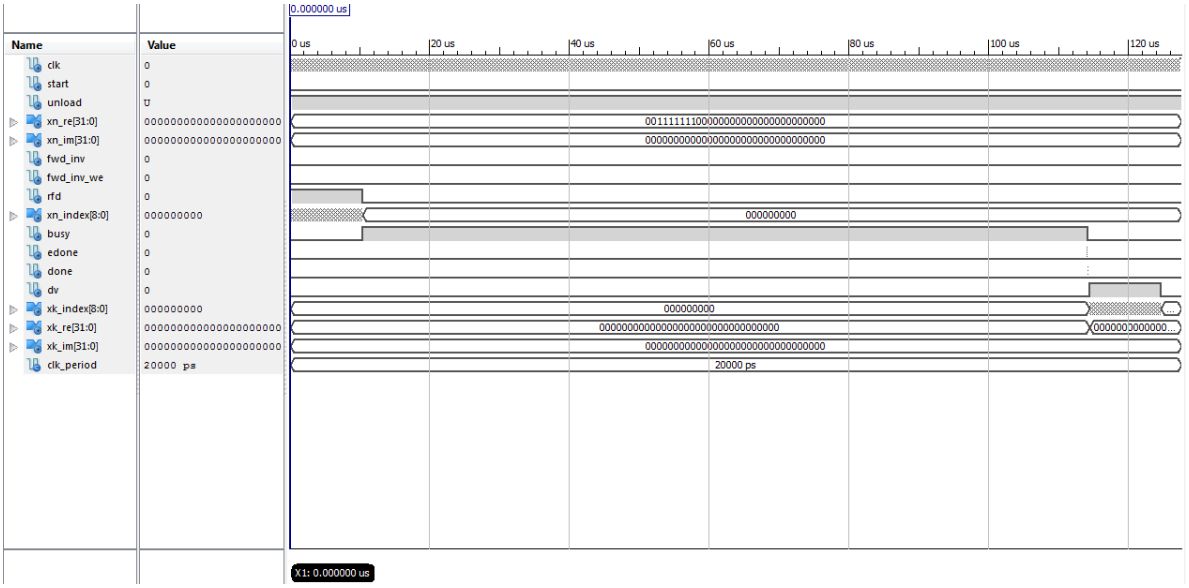
FFT Radix 4 de 256 puntos



Tiempo [us]	Descripción
22.540100	Tiempo transcurrido desde que se pone en 1 la señal *start* hasta que obtenemos el último dato en el bus de salida.
11.920000	Tiempo durante el cual se mantiene en alto la señal *busy*, tiempo que la FPGA tarda en procesar los datos.
5.120000	Tiempo de lectura de los datos reales e imaginarios.
5.120000	Tiempo de despliegue del resultado, datos reales e imaginarios.

Recursos	Usados	Disponibles	Utilización
Numero de Slices	5388	8672	62%
Numero de Slice Flip-Flop	7871	17344	45%
Numero de LUTs de 4 entradas	7269	17344	41%
Numero de IOBs	154	250	61%
Numero de BRAMs	10	28	35%
Numero de Multiplicadores	48	28	171%
Numero de GCLKs	1	24	4%

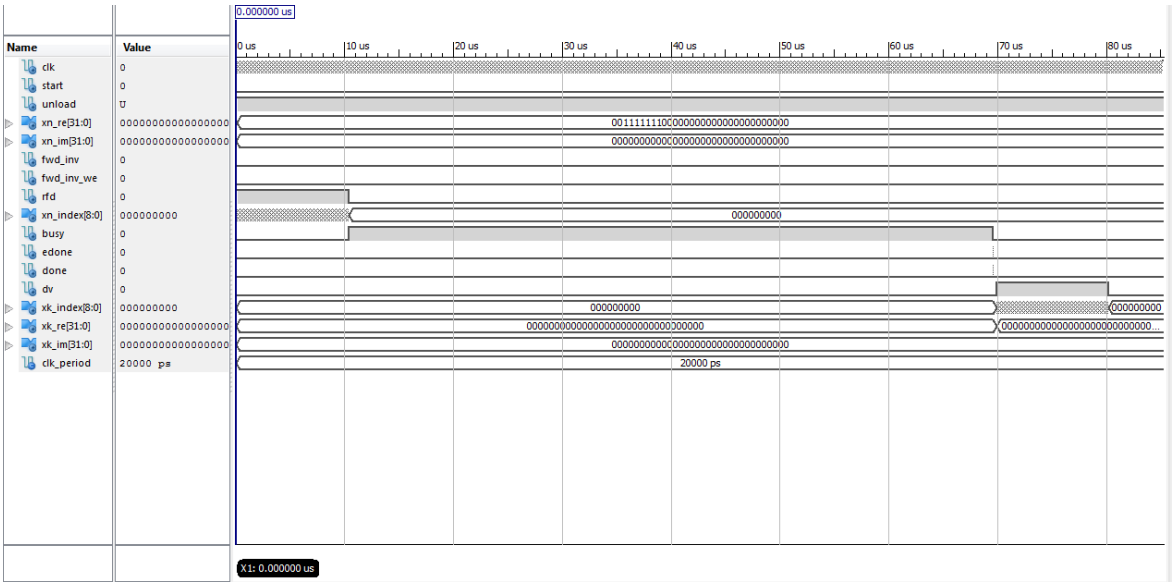
FFT Radix 2 Lite de 512 puntos



Tiempo [us]	Descripción
124.600100	Tiempo transcurrido desde que se pone en 1 la señal *start* hasta que obtenemos el último dato en el bus de salida.
103.780000	Tiempo durante el cual se mantiene en alto la señal *busy*, tiempo que la FPGA tarda en procesar los datos.
10.240000	Tiempo de lectura de los datos reales e imaginarios.
10.240000	Tiempo de despliegue del resultado, datos reales e imaginarios.

Recursos	Usados	Disponibles	Utilización
Numero de Slices	2125	8672	24%
Numero de Slice Flip-Flop	3292	17344	18%
Numero de LUTs de 4 entradas	2971	17344	17%
Numero de IOBs	156	250	62%
Numero de BRAMs	4	28	14%
Numero de Multiplicadores	8	28	28%
Numero de GCLKs	1	24	4%

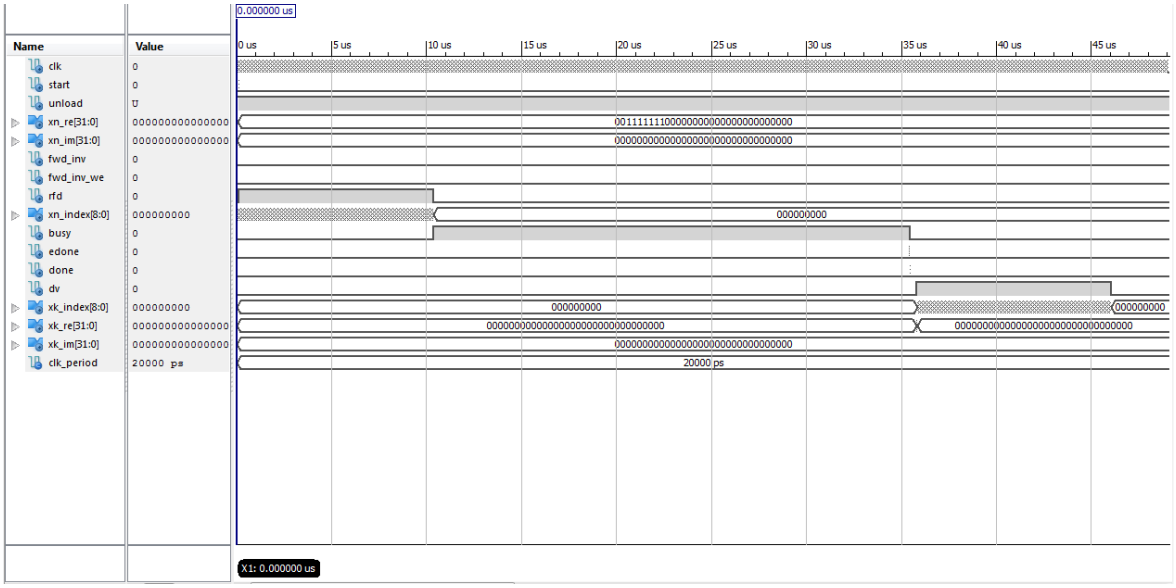
FFT Radix 2 de 512 puntos



Tiempo [us]	Descripción
80.080100	Tiempo transcurrido desde que se pone en 1 la señal *start* hasta que obtenemos el último dato en el bus de salida.
59.260000	Tiempo durante el cual se mantiene en alto la señal *busy*, tiempo que la FPGA tarda en procesar los datos.
10.240000	Tiempo de lectura de los datos reales e imaginarios.
10.240000	Tiempo de despliegue del resultado, datos reales e imaginarios.

Recursos	Usados	Disponibles	Utilización
Numero de Slices	2767	8672	31%
Numero de Slice Flip-Flop	3916	17344	22%
Numero de LUTs de 4 entradas	3953	17344	22%
Numero de IOBs	156	250	62%
Numero de BRAMs	6	28	21%
Numero de Multiplicadores	16	28	57%
Numero de GCLKs	1	24	4%

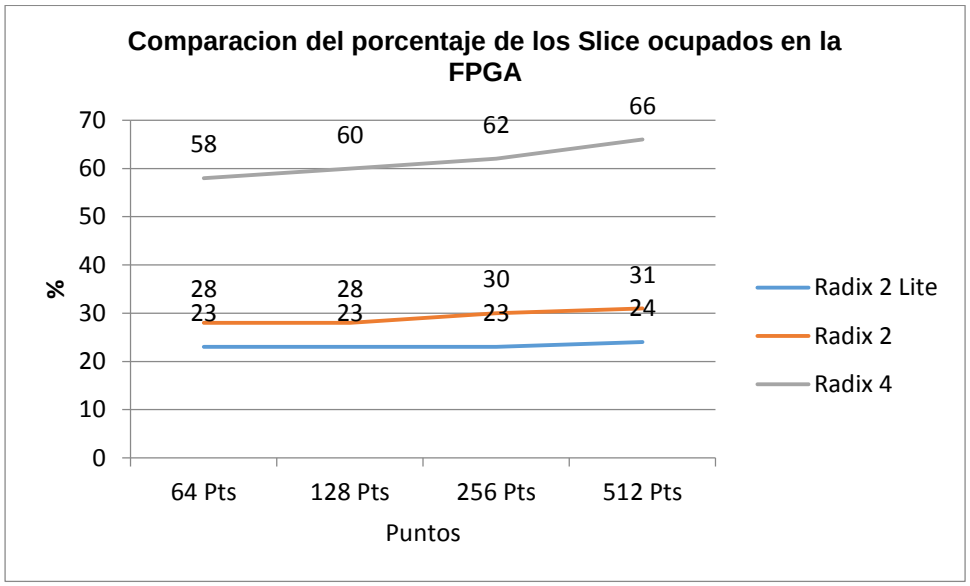
FFT Radix 4 de 512 puntos

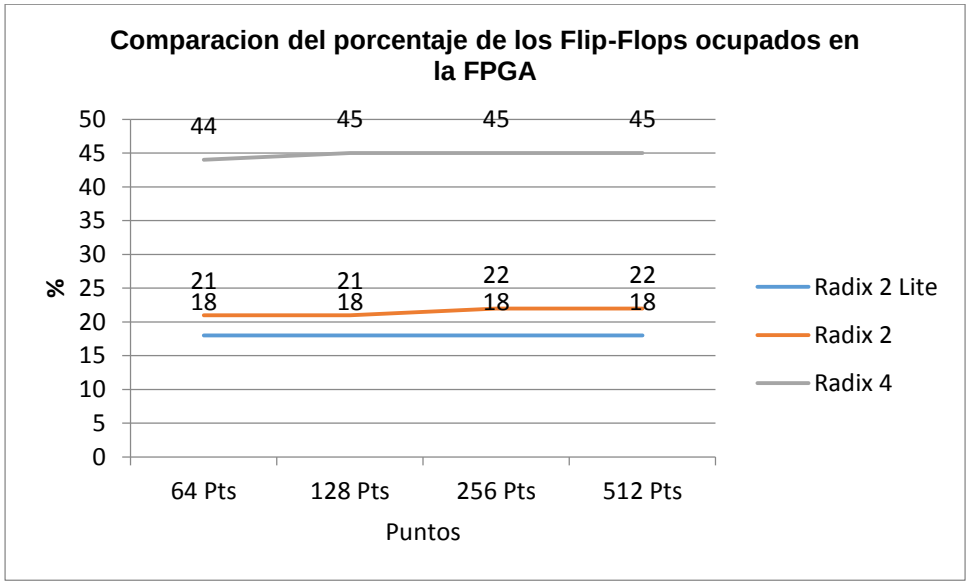


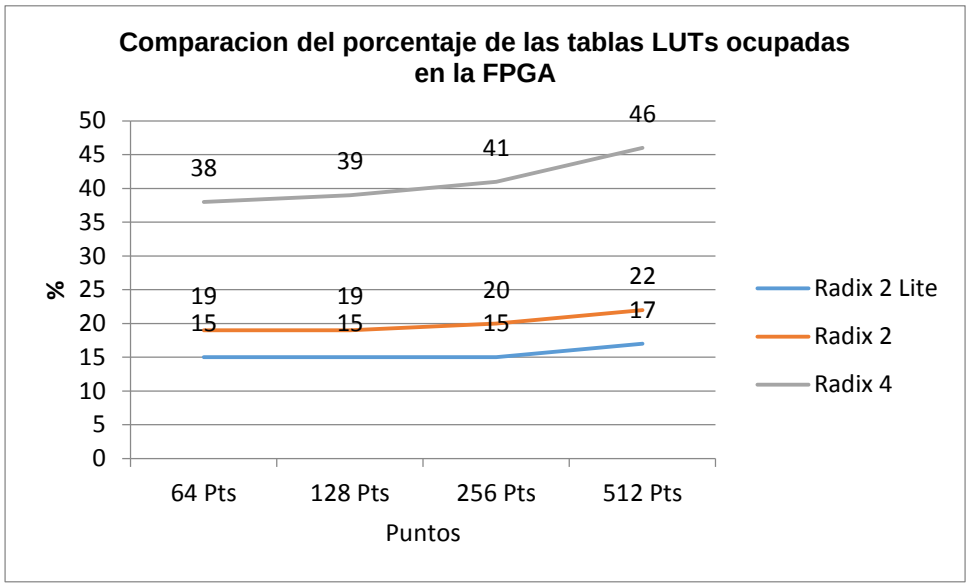
Tiempo [us]	Descripción
45.940100	Tiempo transcurrido desde que se pone en 1 la señal *start* hasta que obtenemos el último dato en el bus de salida.
25.080000	Tiempo durante el cual se mantiene en alto la señal *busy*, tiempo que la FPGA tarda en procesar los datos.
10.240000	Tiempo de lectura de los datos reales e imaginarios.
10.240000	Tiempo de despliegue del resultado, datos reales e imaginarios.

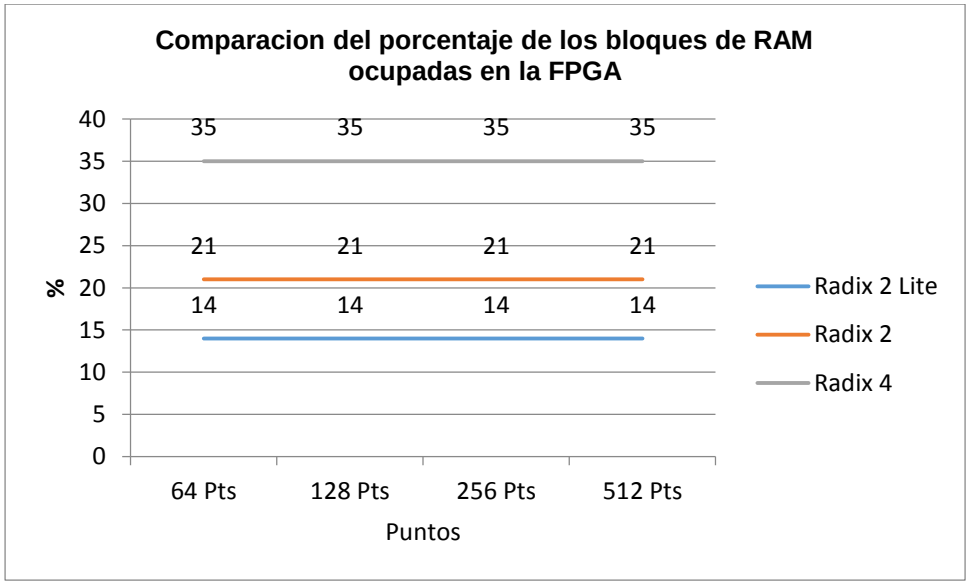
Recursos	Usados	Disponibles	Utilización
Numero de Slices	5790	8672	66%
Numero de Slice Flip-Flop	7961	17344	45%
Numero de LUTs de 4 entradas	8057	17344	46%
Numero de IOBs	156	250	62%
Numero de BRAMs	10	28	35%
Numero de Multiplicadores	48	28	171%
Numero de GCLKs	1	24	4%

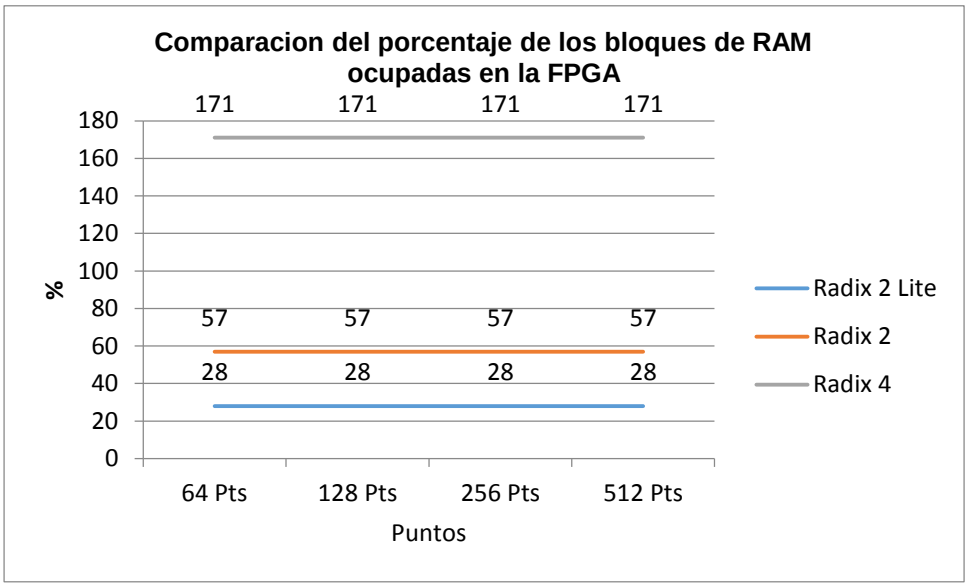
A continuación se muestran las gráficas comparativas para las diferentes arquitecturas y el uso de recursos que emplea cada arquitectura.






Comparacion del porcentaje de las tablas LUTs ocupadas en la FPGA
50
45
40
35
30
25
20
15
10
5
0
%
38
39
41
46
19
19
20
22
15
15
15
17
64 Pts
128 Pts
256 Pts
512 Pts
Puntos
Radix 2 Lite
Radix 2
Radix 4


Comparacion del porcentaje de los bloques de RAM ocupadas en la FPGA
40
35
30
25
20
15
10
5
0
%
35
35
35
35
21
21
21
21
14
14
14
14
64 Pts
128 Pts
256 Pts
512 Pts
Puntos
Radix 2 Lite
Radix 2
Radix 4


Comparacion del porcentaje de los bloques de RAM ocupadas en la FPGA
171
171
171
171
180
160
140
120
100
80
60
40
20
0
%
57
57
57
57
28
28
28
28
64 Pts
128 Pts
256 Pts
512 Pts
Puntos
Radix 2 Lite
Radix 2
Radix 4

Anexo C

Ejemplo de Uso de los bloques DCM de la FPGA Spartan®-3E

Entre los usos de los bloques DCM se encuentra el de crear una señal de reloj con una frecuencia más alta que la señal de reloj base. Para la FPGA Spartan®-3E la frecuencia máxima a la cual podemos llevar la frecuencia base para los DCM es de aproximadamente 334 MHz. Se realizó una prueba para crear una señal de reloj de 100 MHz y 200 MHz a partir del reloj base de 50 MHz ocupando los bloques DCM. Para verificar el correcto funcionamiento de las nuevas señales de reloj se ocupó un osciloscopio con un ancho de banda de 300 MHz y adicionalmente se empleó la herramienta ChipScope™, con el fin de realizar una comparación. Esta última herramienta es un software que funciona como analizador lógico mediante la inserción de un "*core*" dentro de la FPGA que realiza la función de "analizador lógico integrado" (ILA, Integrated Logic Analyzer), este *core* manda las señales a una interfaz gráfica mediante una interfaz de comunicación, permitiendo de esta forma observar las señales lógicas de la FPGA en la PC. Debido a que ChipScope™ no puede mostrar directamente las señales de reloj creadas con los bloques DCM, tenemos que crear nuevas señales lógicas a partir de las señales que nos proporcionan a la salida (señales de 100 y 200 MHZ en este caso) los bloques DCM. Para verificar la señal de reloj, tanto en el ChipScope™ como en el osciloscopio se implementaron dos trenes de pulsos, uno que esta alimentado con un reloj de 100 MHz y tiene 4 veces su periodo y otro que se alimenta con un reloj de 200 Mhz y tiene 4 veces su periodo. En la Figura C.1, para el canal 1 se observa la señal lógica creada a partir de la señal de reloj de 100 MHz y en el canal 2 se observa la señal lógica creada a partir de la señal de reloj de 200 MHz.

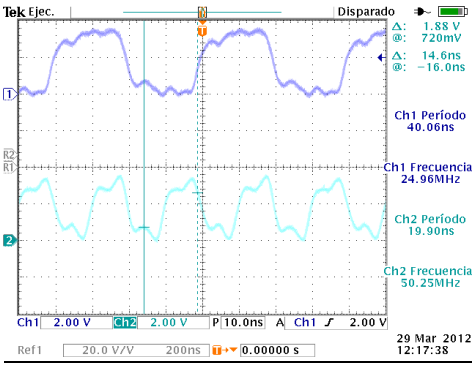


Figura C.1._ En el canal 1 se muestra la señal creada a partir de una señal de 100 MHZ, la cual tiene 4 veces su periodo y en el canal 2 se muestra la señal creada a partir de una señal de 200 MHZ, la cual tiene 4 veces su periodo.

En la Figura C.2 se muestra el tren de pulso obtenido a partir de la frecuencia base de 100 MHz, la imagen se obtuvo con la interfaz gráfica que es parte de ChipScope™. Como se aprecia el periodo de la señal es de 40 ns (la separación entre las marcas de la regla es de 10 ns).

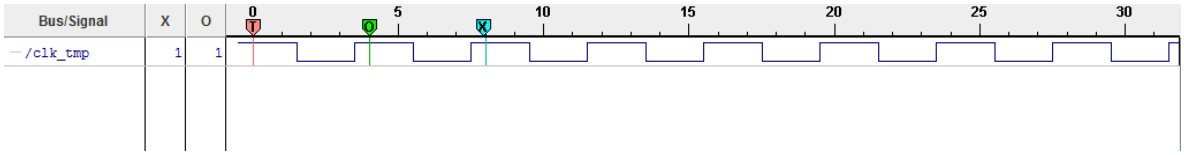


Figura 2._ Señal lógica obtenida con la interfaz gráfica de ChipScope™, esta señal tiene 4 veces el periodo del reloj base de 100 MHz, la separación entre las marcas de la regla es de 10 ns.

En la Figura C.3 se muestra el tren de pulso obtenido a partir de la frecuencia base de 200 MHz, la imagen se obtuvo con la interfaz gráfica que es parte de ChipScope™. Como se aprecia el periodo de la señal es de 20 ns (la separación entre las marcas de la regla es de 5 ns).

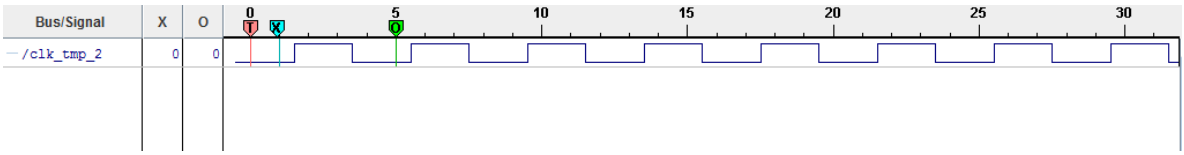


Figura 3._ Señal lógica obtenida con la interfaz gráfica de ChipScope™, esta señal tiene 4 veces el periodo del reloj base de 200 MHz, la separación entre las marcas de la regla es de 5 ns.

Por lo tanto es posible tener un reloj base de 200 MHz en la FPGA Spartan™3E como reloj base para alimentar los diferentes módulos programados sobre esta FPGA, sin que se presente algún problema.

Anexo D

Cartas ASM de los módulos implementados

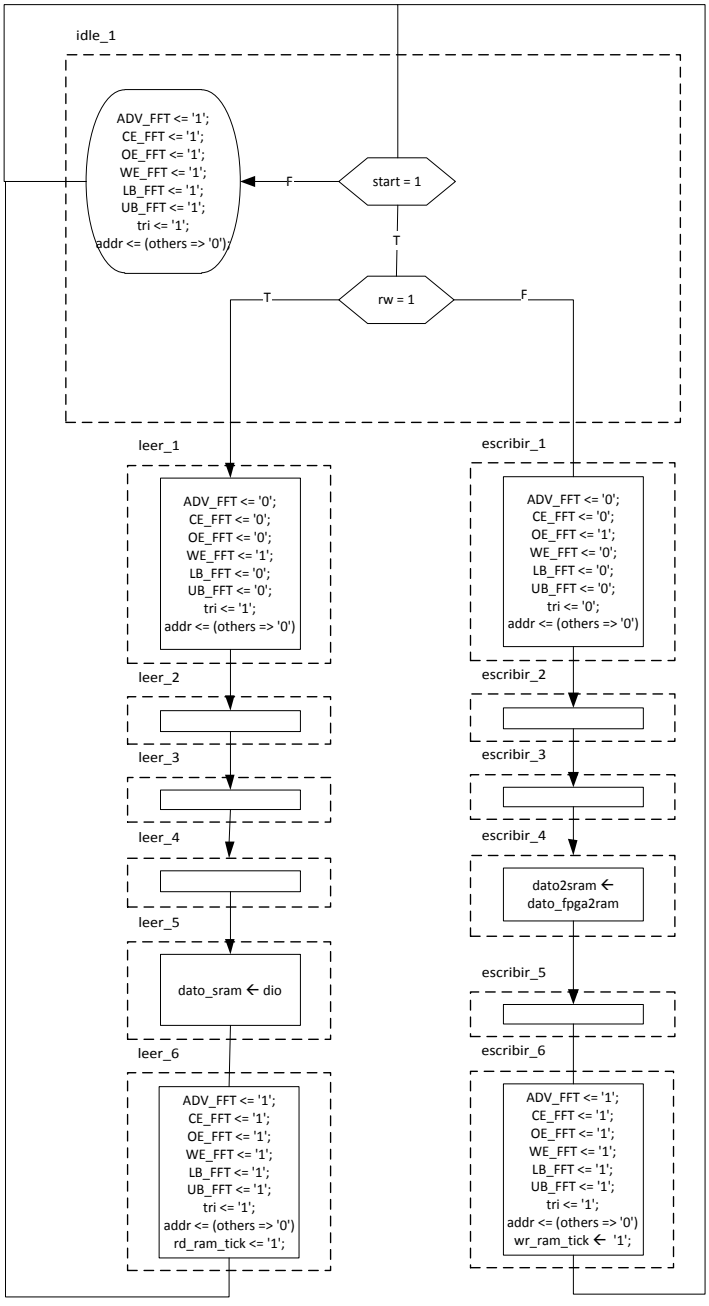


Figura D.1._ Carta ASM para el control de la memoria PSRAM.

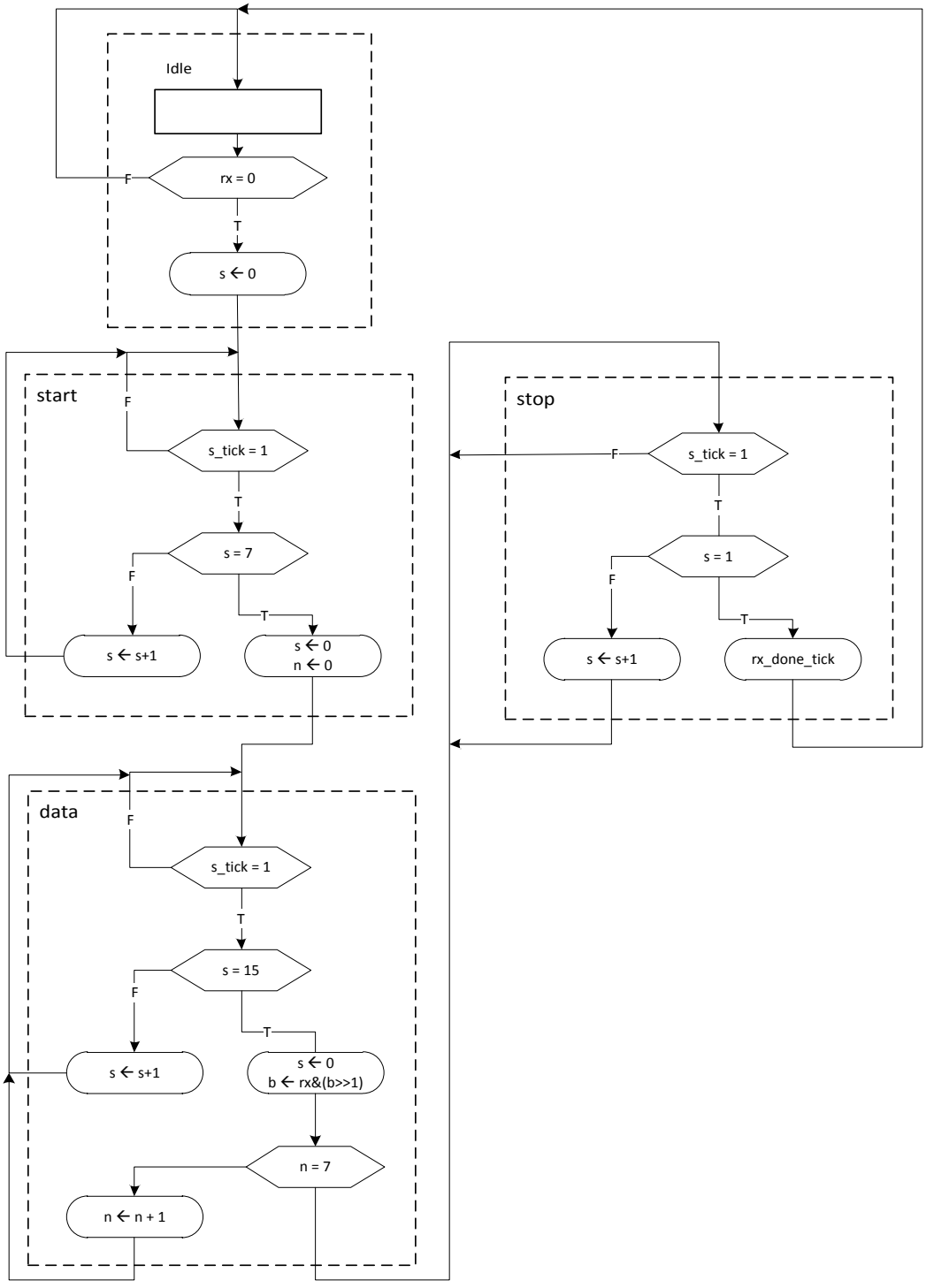


Figura D.2._ Carta ASM del módulo RX de recepción de la UART.

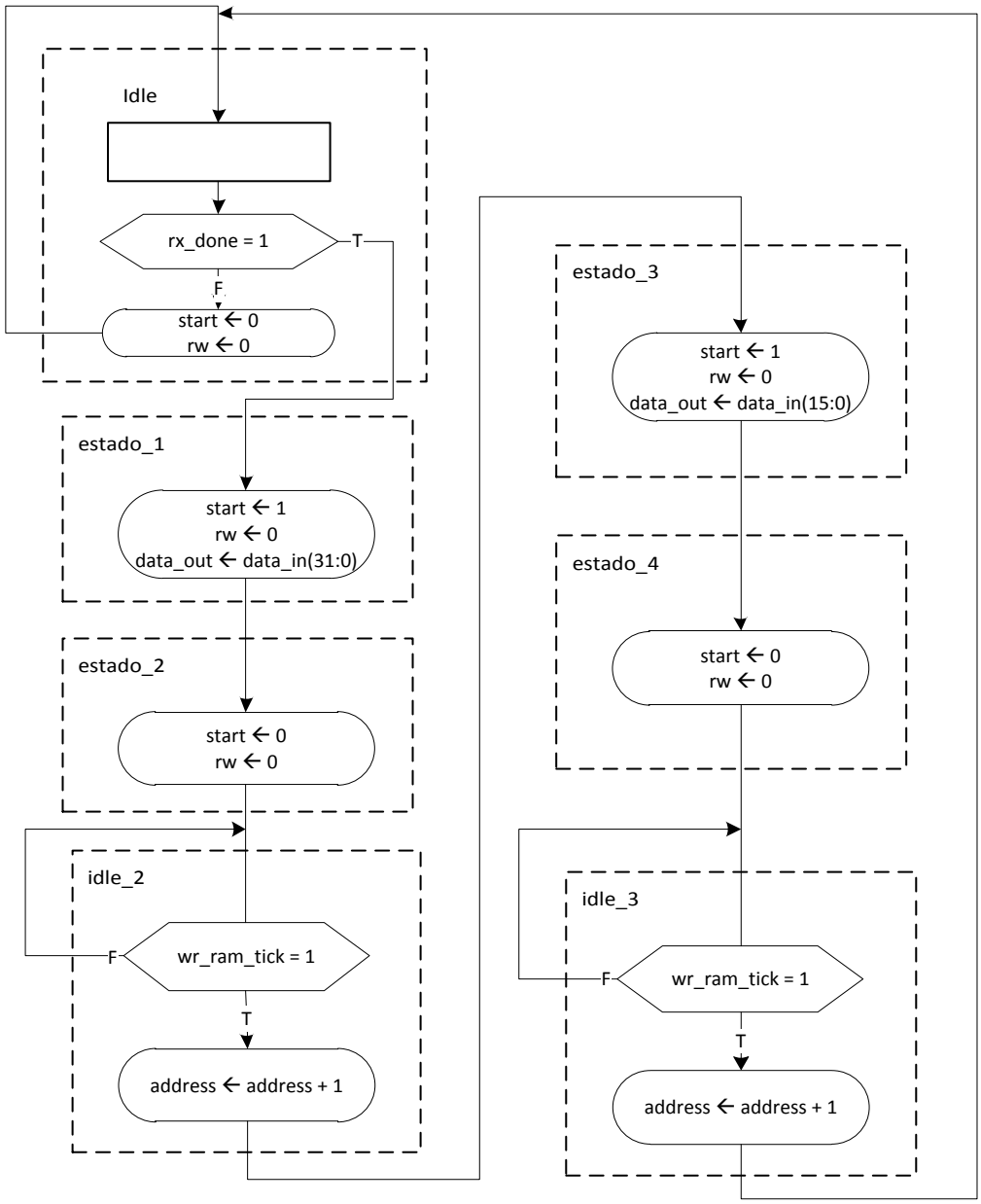


Figura D.3._ Carta ASM del módulo de direccionamiento UART RX –SRAM.

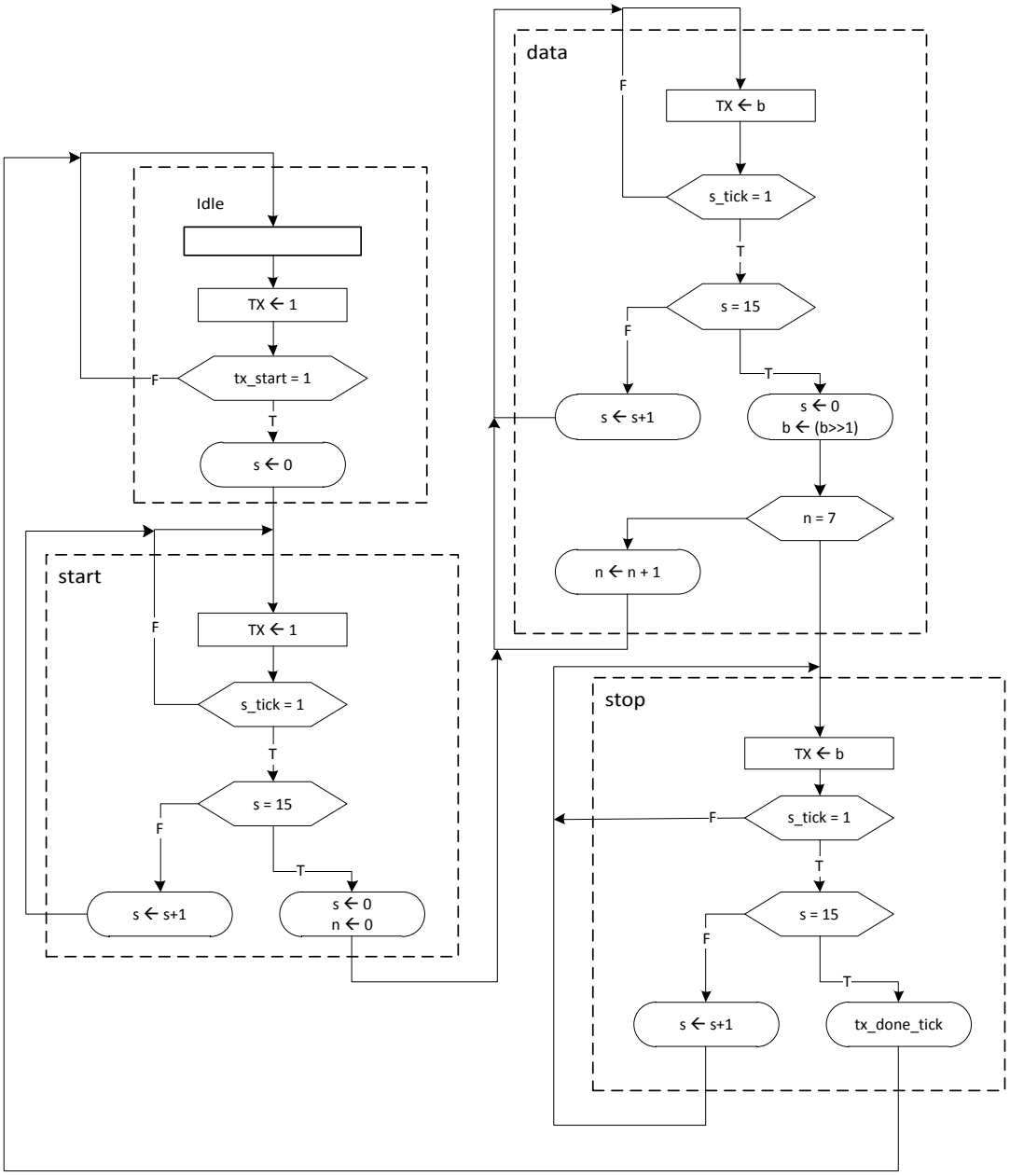
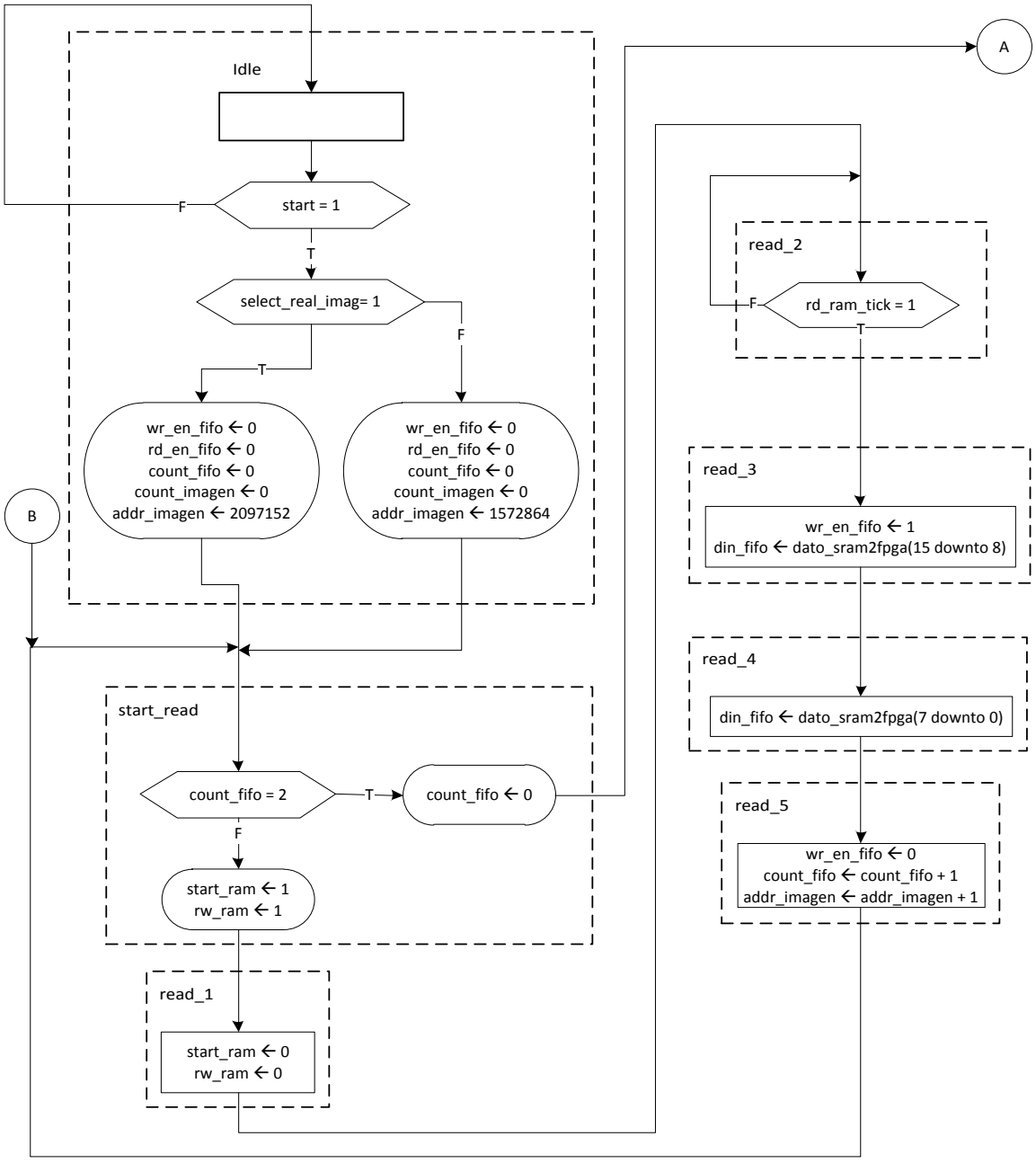
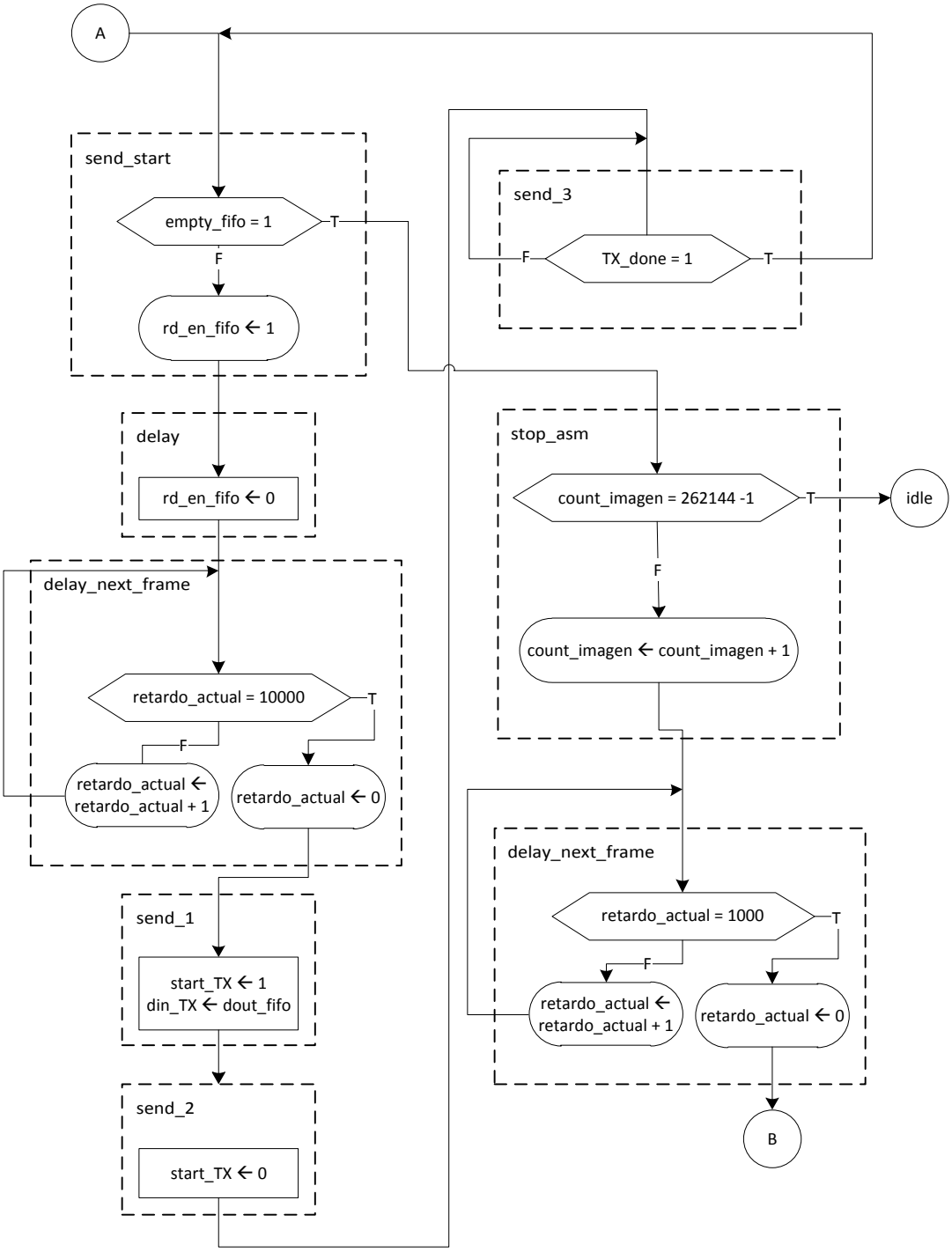


Figura D.4._ Carta ASM del módulo de transmisión de la UART.

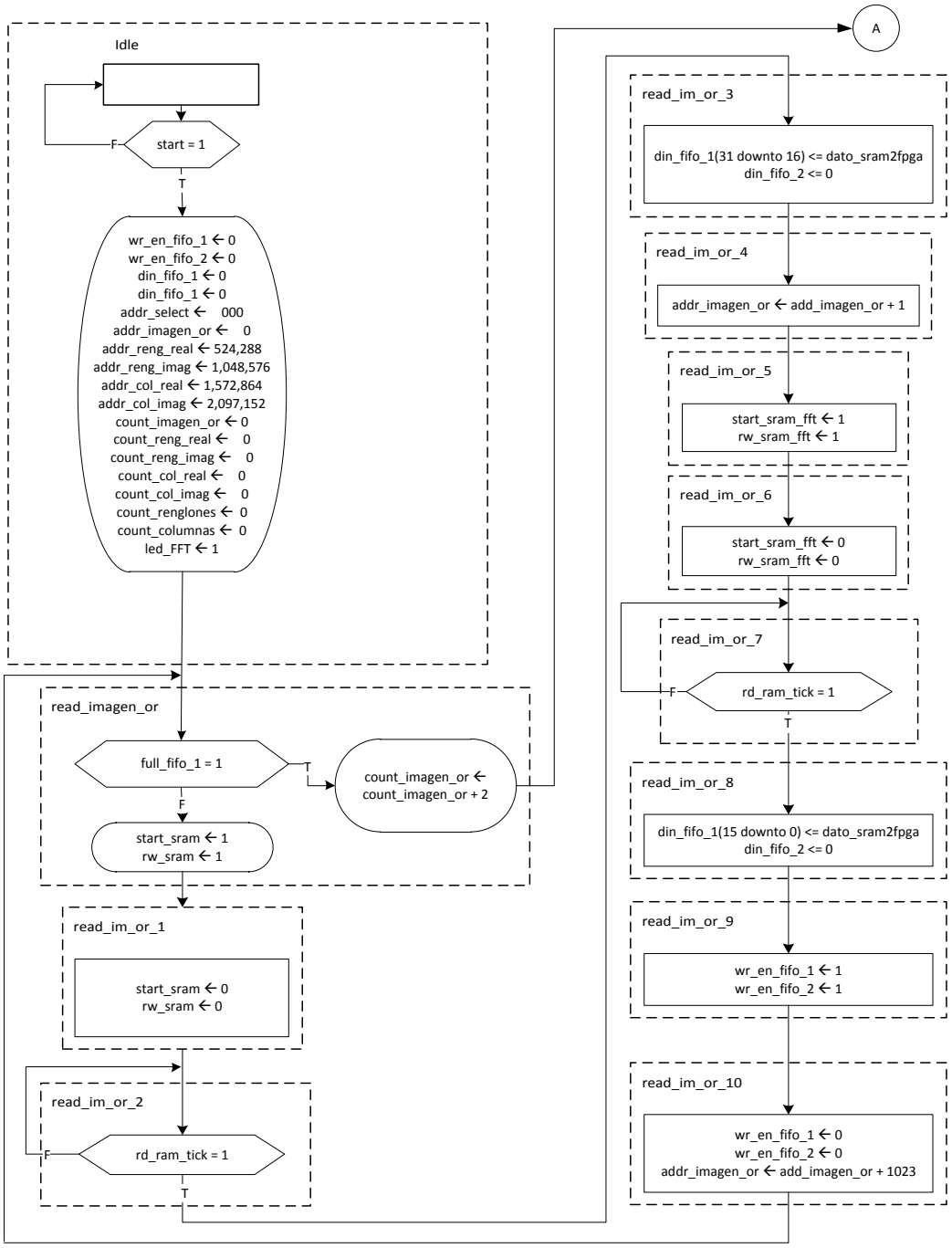

Idle
start = 1
F
T
select_real_imag= 1
T
F
wr_en_fifo ← 0
rd_en_fifo ← 0
count_fifo ← 0
count_imagen ← 0
addr_imagen ← 2097152
wr_en_fifo ← 0
rd_en_fifo ← 0
count_fifo ← 0
count_imagen ← 0
addr_imagen ← 1572864
B
start_read
count_fifo = 2
T
count_fifo ← 0
F
start_ram ← 1
rw_ram ← 1
read_1
start_ram ← 0
rw_ram ← 0
A
read_2
F
rd_ram_tick = 1
T
read_3
wr_en_fifo ← 1
din_fifo ← dato_sram2fpga(15 downto 8)
read_4
din_fifo ← dato_sram2fpga(7 downto 0)
read_5
wr_en_fifo ← 0
count_fifo ← count_fifo + 1
addr_imagen ← addr_imagen + 1

a)

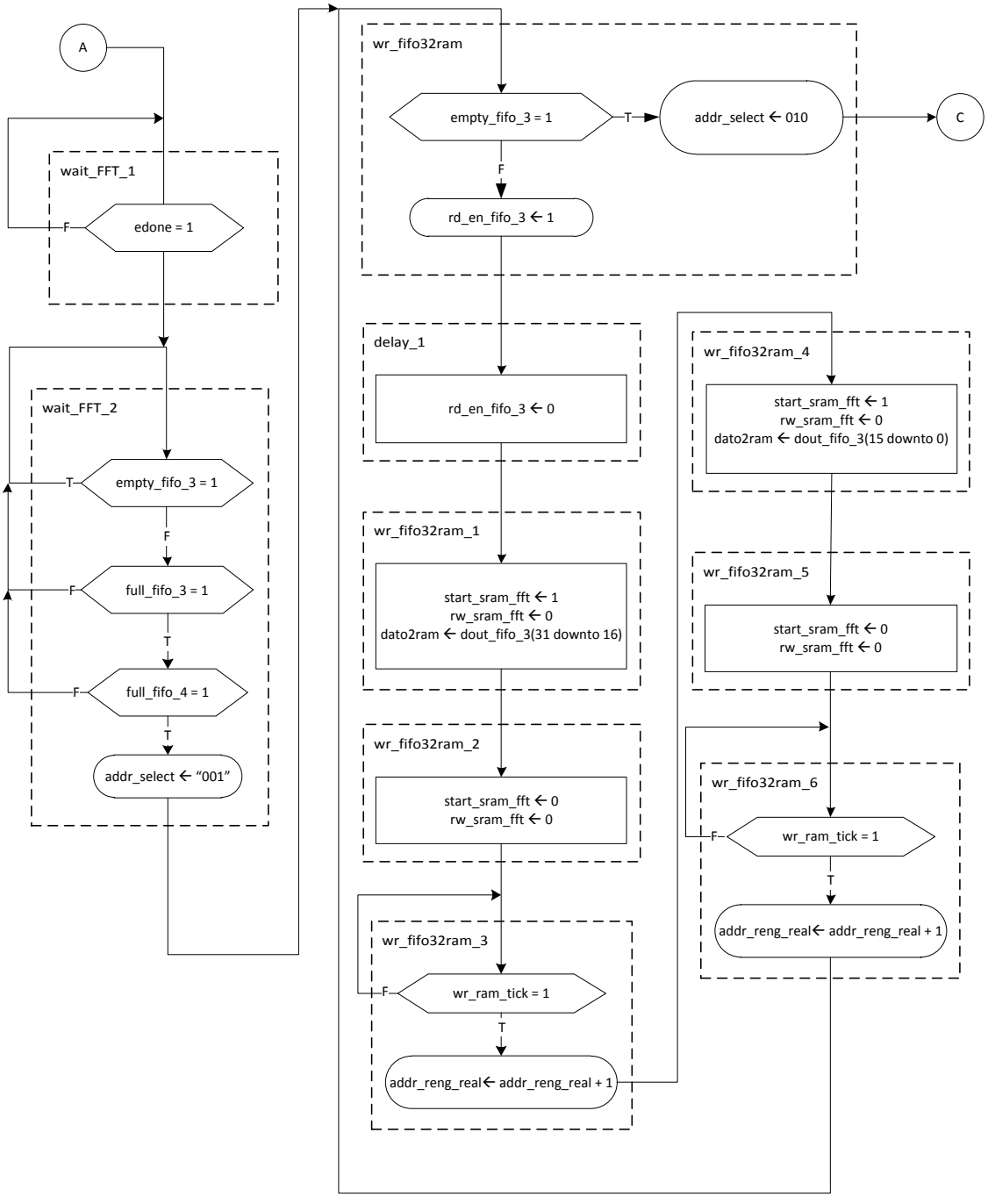


b)

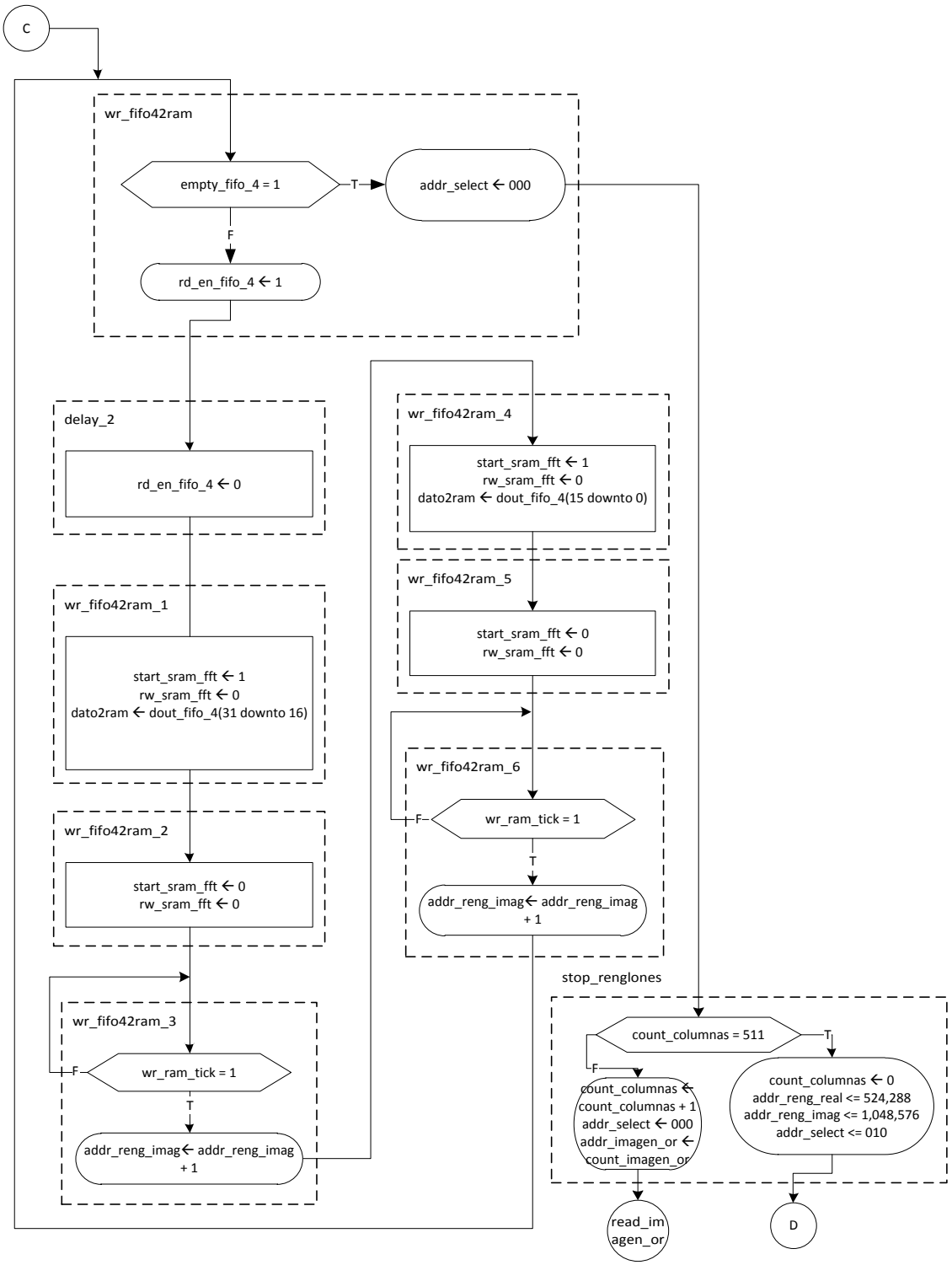
Figura D.5._ Carta ASM del funcionamiento del submódulo de direccionamiento para la UART TX.


Idle
start = 1
F
T
wr_en_fifo_1 ← 0
wr_en_fifo_2 ← 0
din_fifo_1 ← 0
din_fifo_1 ← 0
addr_select ← 000
addr_imagen_or ← 0
addr_reng_real ← 524,288
addr_reng_imag ← 1,048,576
addr_col_real ← 1,572,864
addr_col_imag ← 2,097,152
count_imagen_or ← 0
count_reng_real ← 0
count_reng_imag ← 0
count_col_real ← 0
count_col_imag ← 0
count_renglones ← 0
count_columnas ← 0
led_FFT ← 1
read_imagen_or
full_fifo_1 = 1
T
F
count_imagen_or ← count_imagen_or + 2
start_sram ← 1
rw_sram ← 1
read_im_or_1
start_sram ← 0
rw_sram ← 0
read_im_or_2
rd_ram_tick = 1
F
T
A
read_im_or_3
din_fifo_1(31 downto 16) <= dato_sram2fpga
din_fifo_2 <= 0
read_im_or_4
addr_imagen_or ← add_imagen_or + 1
read_im_or_5
start_sram_fft ← 1
rw_sram_fft ← 1
read_im_or_6
start_sram_fft ← 0
rw_sram_fft ← 0
read_im_or_7
rd_ram_tick = 1
F
T
read_im_or_8
din_fifo_1(15 downto 0) <= dato_sram2fpga
din_fifo_2 <= 0
read_im_or_9
wr_en_fifo_1 ← 1
wr_en_fifo_2 ← 1
read_im_or_10
wr_en_fifo_1 ← 0
wr_en_fifo_2 ← 0
addr_imagen_or ← add_imagen_or + 1023

a)


A
wait_FFT_1
F
edone = 1
wait_FFT_2
T
empty_fifo_3 = 1
F
F
full_fifo_3 = 1
T
F
full_fifo_4 = 1
T
addr_select ← "001"
wr_fifo32ram
empty_fifo_3 = 1
T
addr_select ← 010
C
F
rd_en_fifo_3 ← 1
delay_1
rd_en_fifo_3 ← 0
wr_fifo32ram_1
start_sram_fft ← 1
rw_sram_fft ← 0
dato2ram ← dout_fifo_3(31 downto 16)
wr_fifo32ram_2
start_sram_fft ← 0
rw_sram_fft ← 0
wr_fifo32ram_3
F
wr_ram_tick = 1
T
addr_reng_real← addr_reng_real + 1
wr_fifo32ram_4
start_sram_fft ← 1
rw_sram_fft ← 0
dato2ram ← dout_fifo_3(15 downto 0)
wr_fifo32ram_5
start_sram_fft ← 0
rw_sram_fft ← 0
wr_fifo32ram_6
F
wr_ram_tick = 1
T
addr_reng_real← addr_reng_real + 1

b)


C
wr_fifo42ram
empty_fifo_4 = 1
T
addr_select ← 000
F
rd_en_fifo_4 ← 1
delay_2
rd_en_fifo_4 ← 0
wr_fifo42ram_1
start_sram_fft ← 1
rw_sram_fft ← 0
dato2ram ← dout_fifo_4(31 downto 16)
wr_fifo42ram_2
start_sram_fft ← 0
rw_sram_fft ← 0
wr_fifo42ram_3
F
wr_ram_tick = 1
T
addr_reng_imag← addr_reng_imag + 1
wr_fifo42ram_4
start_sram_fft ← 1
rw_sram_fft ← 0
dato2ram ← dout_fifo_4(15 downto 0)
wr_fifo42ram_5
start_sram_fft ← 0
rw_sram_fft ← 0
wr_fifo42ram_6
F
wr_ram_tick = 1
T
addr_reng_imag← addr_reng_imag + 1
stop_renglones
count_columnas = 511
T
F
count_columnas ← count_columnas + 1
addr_select ← 000
addr_imagen_or ← count_imagen_or
count_columnas ← 0
addr_reng_real <= 524,288
addr_reng_imag <= 1,048,576
addr_select <= 010
read_imagen_or
D

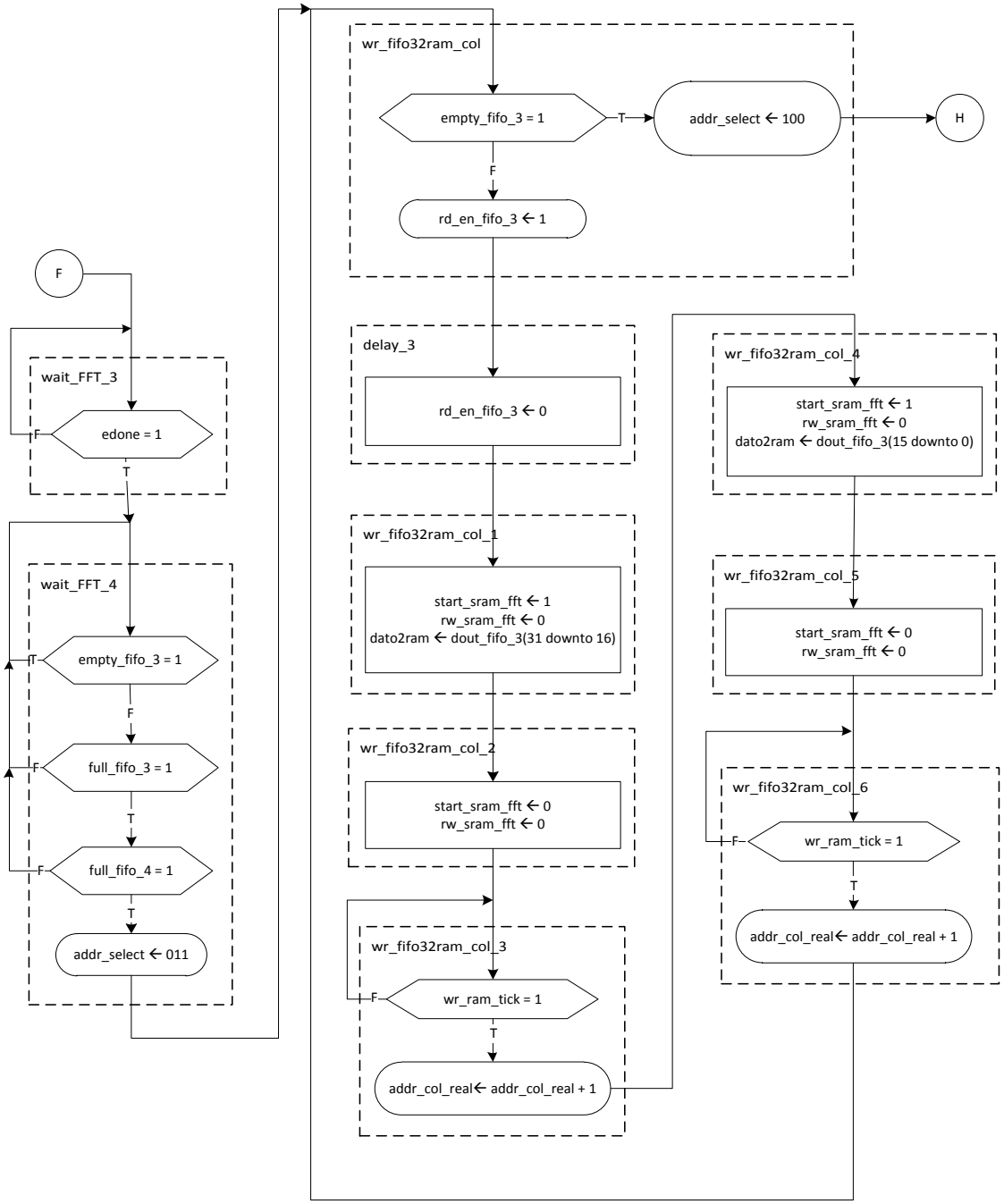
c)


D
read_fft1_imag
full_fifo_2 = 1
T
addr_select <= 001
count_reng_imag <=
count_reng_imag + 2
E
F
start_sram_fft ← 1
rw_sram_fft ← 1
read_fft1_imag_1
start_sram_fft ← 0
rw_sram_fft ← 0
read_fft1_imag_2
F
rd_ram_tick = 1
T
read_fft1_imag_3
din_fifo_2(31 downto 16) <= dato_sram2fpga
read_fft1_imag_4
addr_reng_imag ← add_reng_imag + 1
read_fft1_imag_5
start_sram_fft ← 1
rw_sram_fft ← 1
read_fft1_imag_6
start_sram_fft ← 0
rw_sram_fft ← 0
read_fft1_imag_7
F
rd_ram_tick = 1
T
read_fft1_imag_8
din_fifo_2(15 downto 0) <= dato_sram2fpga;
read_fft1_imag_9
wr_en_fifo_2 ← 1
read_fft1_imag_10
wr_en_fifo_2 ← 0
addr_reng_imag ← add_reng_imag + 1023

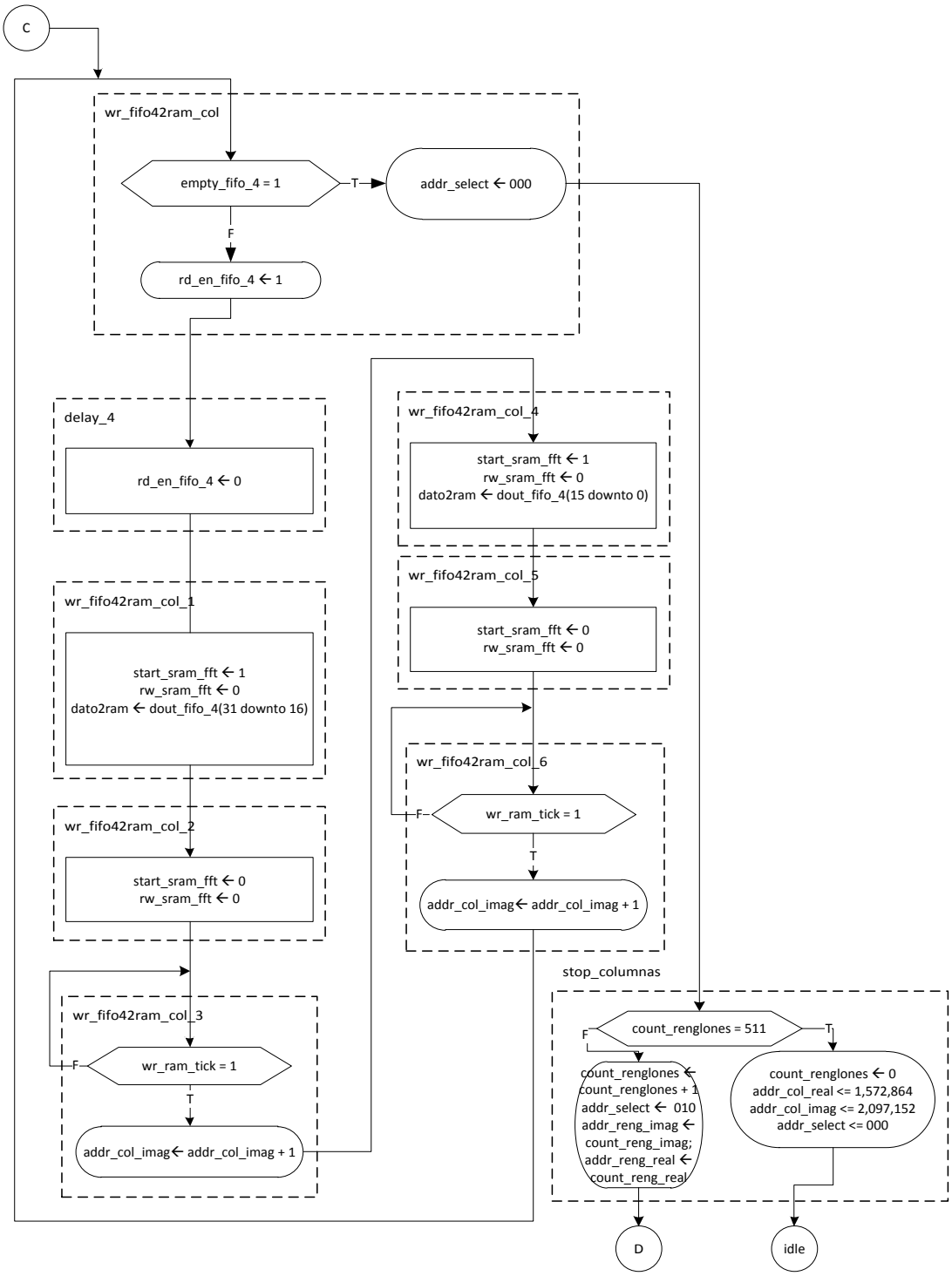
d)


E
read_fft1_real
full_fifo_1 = 1
T
addr_select <= 011
count_reng_real <=
count_reng_real + 2
F
F
start_sram_fft ← 1
rw_sram_fft ← 1
read_fft1_real_1
start_sram_fft ← 0
rw_sram_fft ← 0
read_fft1_real_2
F
rd_ram_tick = 1
T
read_fft1_real_3
din_fifo_1(31 downto 16) <= dato_sram2fpga
read_fft1_real_4
addr_reng_real ← add_reng_real + 1
read_fft1_real_5
start_sram_fft ← 1
rw_sram_fft ← 1
read_fft1_real_6
start_sram_fft ← 0
rw_sram_fft ← 0
read_fft1_real_7
F
rd_ram_tick = 1
T
read_fft1_real_8
din_fifo_1(15 downto 0) <= dato_sram2fpga
read_fft1_real_9
wr_en_fifo_1 ← 1
read_fft1_real_10
wr_en_fifo_1 ← 0
addr_reng_real ← add_reng_real + 1023

e)


F
wait_FFT_3
F
edone = 1
T
wait_FFT_4
T
empty_fifo_3 = 1
F
F
full_fifo_3 = 1
T
F
full_fifo_4 = 1
T
addr_select ← 011
wr_fifo32ram_col
empty_fifo_3 = 1
T
addr_select ← 100
H
F
rd_en_fifo_3 ← 1
delay_3
rd_en_fifo_3 ← 0
wr_fifo32ram_col_1
start_sram_fft ← 1
rw_sram_fft ← 0
dato2ram ← dout_fifo_3(31 downto 16)
wr_fifo32ram_col_2
start_sram_fft ← 0
rw_sram_fft ← 0
wr_fifo32ram_col_3
F
wr_ram_tick = 1
T
addr_col_real← addr_col_real + 1
wr_fifo32ram_col_4
start_sram_fft ← 1
rw_sram_fft ← 0
dato2ram ← dout_fifo_3(15 downto 0)
wr_fifo32ram_col_5
start_sram_fft ← 0
rw_sram_fft ← 0
wr_fifo32ram_col_6
F
wr_ram_tick = 1
T
addr_col_real← addr_col_real + 1

f)



g)

Figura D.6._ Carta ASM del algoritmo para el proceso de la FFT de una imagen (a, b, c, d, e, f, g).

Anexo E

Comparación del tiempo de procesamiento del FPGA, CPU y GPU

En la Tabla E.1 se muestra la comparación de los tiempos de procesamiento para un CPU, GPU, FPGA Virtex®-7, FPGA Spartan®-6 y FPGA Spartan®-3E.

Tabla E.1._ Tabla comparativa de los tiempos de procesamiento para los PFGAs, CPU y GPU.

Dispositivo	Tiempo de procesamiento (ms)
Spartan®-3E	789.88
Virtex®-7	2.9
Spartan®-6	3.13
GPU	5.23
CPU	7.04

El tiempo de procesamiento de las FPGAs Virtex®-7 y Spartan®-6 es el ideal, no se tomaron retrasos que puedan existir, tales como el tiempo de inicialización del acceso a memoria o retardos presentes en la lectura y escritura de la memoria, entre otros retrasos que se puedan presentar en la implementación de algún modulo del sistema.

Las características de la computadora donde se realizaron las pruebas de procesamiento para CPU y GPU son las siguientes:

- Intel® Core™ i5-2320.
- 8 GB memoria RAM.
- Tarjeta de video NVIDIA GeForce® GT 520.
- HDD 500 GB.
- Windows 7 Home Edition.
- Matlab 2011a.

Para realizar las pruebas de procesamiento de CPU y GPU se utilizó Matlab con la función fft2 para obtener la Transformada Rápida de Fourier.

Referencias

[1] Tomasz S. Czajkowski, C. J. (n.d.). *Fast Fourier Transform Implementation for High Speed Astrophysics Applications on FPGAs.*

[2] Uzun, I., Amira, A., & Bouridane, A. (2005). FPGA implementations of fast Fourier transforms for real-time signal and image processing. *Vision, Image and Signal Processing, IEE Proceedings, 152*, 283 - 296.

[3] Ramirez, C. L., Enseñat, R. A., & Mas, F. J. (2009). Acquisition and Digital Images Processing, Comparative Analysis of FPGA, DSP, PC for the Subtraction and Thresholding. In InTech, *Image Processing.* Yung-Sheng Chen (Ed).

[4] Yu, J. S.-L., Deng, L., Kestur, S., Narayanan, V., & Chakrabarti, C. (2009). FPGA architecture for 2D Discrete Fourier Transform based on 2D decomposition for large-sized data. *SiPS 2009* (págs. 121 -126). Tampere, Finlandia: IEEE Workshop on.

[5] Rodríguez-Ramos, J. M., Castelló, E. M., Conde, C. D., Valido, M. R., & Marichal-Hernández, J. G. (2008). 2D-FFT implementation on FPGA for wavefront phase recovery from the CAFADIS camera. *SPIE Proceedings 7015,* Adaptive Optics Systems, 701539*, doi:10.1117/12.789312*.

[6] Hardy, J. W. (1998). *Adaptive Optics for Astronomical Telescopes* (1 ed.). Oxford University Press, USA. p. 27.

[7] Covington, M. A. (2002). *Celestial Objects for Modern Telescopes: Practical Amateur Astronomy* (1 ed., Vol. 2). Cambridge University Press. p.12

[8] Rodríguez , L. F. (1996). *Formación Estelar* (1 ed.). Fondo de Cultura Economica. p. 133-134

[9] Tyson, R. K. (2000). *Introduction to Adaptive Optics* (1 ed.). SPIE Publications. p. 5.

[10] H.I. Campbell, A. G. (2006). Wavefront Sensing: From Historical Roots to the State-of-the-Art. *EAS Publications Series*(22), pp 165-185.

[11] Noll, R. J. (1978). Phase estimates from slope-type wave-front sensors. *Journal of the Optical Society of America, 68*(1), pp 139-140.

[12] Roddier, F., & Roddier, C. (1991). Wavefront reconstruction using iterative Fourier transforms. *Optical Society of America, 30*(11), pp 1325-1327.

[13] Maxfield, C. (2004). *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows* (1 ed.). Newnes. p. 4-5.

[14] GENERA. (01 de Enero de 2008). *GENERA.* Obtenido de GENERA: http://www.generatecnologias.es/aplicaciones_fpga.html

[15] Mandado, E., & Mandado, Y. (2007). *Sistemas electrónicos digitales* (9 ed.). Marcombo.

[16] Smith, S. W. (1997). *The Scientist & Engineer's Guide to Digital Signal Processing* (1 ed.). California Technical Pub. p. 413.

[17] Bober, B. (17 de Julio de 2008). *Altera and Xilinx Report: The Battle Continues.* Obtenido de Seeking Alpha: http://seekingalpha.com/article/85478-altera-and-xilinx-report-the-battle-continues

Bibliografía

Tomasi, W. (2003). *Sistemas de Comunicaciones Electronicas* (4 ed.). Pearson Publications Company.

VanderLugt, A. (1992). *Optical Signal Processing* (1 ed.). Wiley-Interscience.

Dorf, R. C. (2006). *Circuits, Signals, and Speech and Image Processing* (1 ed.). CRC Press.

Russ, J. C. (2011). *The Image Processing Handbook* (6 ed.). CRC Press.

Roberts, M. J. (2005). *SEÑALES Y SISTEMAS* (1 ed.). MCGRAW-HILL / INTERAMERICANA DE MEXICO.

Proakis, J. G., & Manolakis, D. G. (2007). *Digital Signal Processing* (3 ed.). Pearson Prentice Hall.

Mitra, S. K. (2007). *Procesamiento de señales digitales : un enfoque basado en computadora* (3 ed.). McGraw-Hill.

Albertí, E. B. (2010). *Procesado digital de señales* (1 ed., Vol. 2). Edicions UPC SL.

Brown, S. (2000). *Fundamentals of Digital Logic With Vhdl Design* (1 ed.). William C Brown Pub.

Mendoza Manzano, M. A. (2009). *Procesamiento y análisis digital de imágenes mediante dispositivos lógicos programables. Tesis de licenciatura. Universidad Tecnologica de la Mixteca.*

Mano, M. M. (2012). *Digital Design: With an Introduction to the Verilog HDL* (5 ed.). Prentice Hal.

Jara, J. A. (2003). *VHDL El arte de programar sistemas digitales* (1 ed.). Cecsa.

Roth, J. C. (2009). *Fundamentals of Logic Design* (6 ed.). CL Engineering.

Frenzel, L. E. (1998). *Crash Course in Digital Technology* (2 ed.). Newnes.

Maini, A. K. (2007). *Digital electronics: principles, devices and applications* (1 ed.). John Wiley & Sons.

Horenstein, M. N. (1995). *Microelectronic Circuit and Devices* (2 ed.). Prentice Hall.

Mandado, E., & Mandado, Y. (2007). *Sistemas electrónicos digitales* (9 ed.). Marcombo.

Waser, R. (2012). *Nanoelectronics and Information Technology* (3 ed.). Wiley-VCH.

Millman, J. (1987). *Microelectronics* (2 ed.). Mcgraw-Hill.

Nelson, V. P., Nagle, H. T., Caroll, B. D., & Irwin, D. (1995). *Digital Logic Circuit Analysis and Design* (1 ed.). Prentice Hall.

etech. (1 de Marzo de 2011). La flota sigue al buque insignia. *etech*, pág. 23. Obtenido de http://www.slideshare.net/RSComponentsMalta/etech-magazine-issue-2

Ashenden, P. J. (2008). *The Designer's Guide to VHDL* (3 ed.). Morgan Kaufmann.

Chu, P. P. (2006). *RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability* (1 ed.). Wiley-IEEE Press.

Chu, P. P. (2008). *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3* (1 ed.). Wiley-Interscience.

Kafig, W. (2011). *VHDL 101: Everything you need to know to get started* (1 ed.). Newnes.

Kilts, S. (2007). *Advanced FPGA Design: Architecture, Implementation, and Optimization* (1 ed.). Wiley-IEEE Press.

Perry, D. (2002). *VHDL : Programming By Example* (4 ed.). McGraw-Hill Professional.

Rushton, A. (2011). *VHDL for Logic Synthesis* (3 ed.). Wiley.

Axelson, J. (2007). *Serial Port Complete 2nd Editon: COM Ports, USB Virtual Com Ports, and Ports for Embedded Systems* (2nd ed.). LAKEVIEW RESEARCH.

Instruments, N. (2004, 01 02). Retrieved from National Instruments Web Site: http://digital.ni.com/public.nsf/allkb/039001258CEF8FB686256E0F005888D