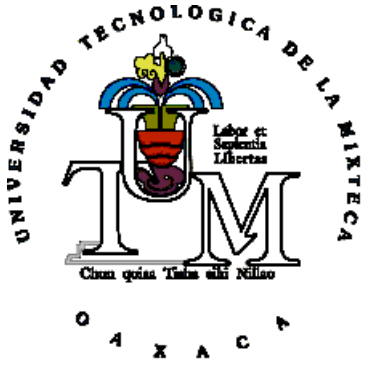


UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

" PROCESAMIENTO Y ANÁLISIS DIGITAL DE IMÁGENES MEDIANTE DISPOSITIVOS LÓGICOS PROGRAMABLES "

TESIS PARA OBTENER EL TÍTULO PROFESIONAL DE:

INGENIERO EN ELECTRÓNICA

PRESENTA:

C. MANUEL ALEJANDRO MENDOZA MANZANO

DIRECTOR DE TESIS:

DR. ENRIQUE GUZMÁN RAMÍREZ

HUAJUAPAN DE LEÓN, OAXACA. FEBRERO DE 2009.

Tesis presentada en Febrero de 2009 ante los sinodales:

Dr. Antonio Orantes Molina

M.C. Felipe Santiago Espinosa

M.C. José Antonio Moreno Espinosa

Director de tesis:
Dr. Enrique Guzmán Ramírez

DEDICATORIA

Dedicada a Jah de los ejercitos. Agradecimientos a mis padres Joel y Evelia por su apoyo incondicional, y a mis hermanos: Carlos, Patricia, Lucia y Fabian por sus consejos, amistad y apoyo.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

El presente trabajo de tesis expone el diseño de una herramienta para el ámbito académico y de investigación, enfocada al área de procesamiento y análisis digital de imágenes sobre Circuitos Digitales Configurables (CDC), específicamente sobre un Arreglo de Compuertas Programable en Campo (FPGA, *Field Programmable Gate Array*). El diseño de la herramienta fue dividido en cuatro fases.

En la primera fase se realizó el diseño de una arquitectura de aplicación específica, orientada al procesamiento y análisis digital de imágenes, mediante un Lenguaje de Descripción de Hardware (HDL, *Hardware Description Language*), el cual facilita el modelado o descripción de un sistema digital. Para el modelado de esta arquitectura en un FPGA, se utilizó uno de los HDLs más populares existentes, el VHDL (*Very High Speed Integrated Circuit Hardware Description Language*), cuya característica más importante es ser un lenguaje estandarizado [43], [40]; además, se ha seguido la metodología de diseño de circuitos digitales descendente (*Top-Down*), cuya principal ventaja es consentir la división del diseño en módulos de menor complejidad, lo que permite detallar un módulo tanto como sea necesario sin llegar a un nivel de abstracción bajo [34], [36].

La segunda fase consiste en el diseño e implementación de un sistema de hardware para aplicaciones de procesamiento y/o análisis digital de imágenes con base en la arquitectura modelada en la primera fase. Considerando el modelado de esta arquitectura como un procesador de aplicación específica, se ha seguido la metodología de diseño de un sistema empotrado para el desarrollo de esta fase.

La tercera fase consta del diseño de una interfaz de usuario con el sistema implementado en las fases anteriores, mediante una computadora personal; el diseño de la interfaz de usuario se basa en la metodología de diseño de aplicaciones de Interfaces de Usuario Gráficas (GUI, *Graphical User Interface*) *Bridge* [51], y en el paradigma de Programación Orientada a Objetos (OOP, *Object Oriented Programming*) [14]; la implementación y codificación se realizó en el Entorno de Desarrollo Integrado (IDE, *Integrated Development Environment*) C++ Builder 6.0. Esta interfaz cumple con la función de permitir al usuario visualizar los resultados del procesamiento y/o análisis aplicado a la imagen por parte del algoritmo implementado en el FPGA. El protocolo de comunicaciones elegido para implementar esta interfaz es el USB, el cual permite alcanzar una velocidad de transferencia de datos elevada.

La unión de estas tres primeras fases genera la herramienta propuesta en este trabajo de tesis.

Finalmente, en la última fase, se crea una metodología integral de diseño que permite al usuario implementar e integrar algoritmos de procesamiento y/o análisis digital de imágenes en la herramienta diseñada. Dicha metodología incluye procedimientos que involucran tanto a la arquitectura de aplicación específica de la primera fase, como a la interfaz de usuario desarrollada en la fase tres.

Las imágenes que esta herramienta procesa son de formato BMP en tonos de gris, 8 bits/píxel. Técnicas para el mejoramiento, restauración, compresión y segmentación de la imagen son algunos de los algoritmos de procesamiento de imágenes que fueron incluidos en la implementación del proyecto.

1.1. Marco teórico

El **Procesamiento** y **Análisis Digital de Imágenes** es un área de la ingeniería que manipula y analiza la información contenida en una imagen digital por medio de un procesador. El procesamiento digital de imágenes es el conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar su calidad, mientras que el análisis de imágenes incluye aquellas técnicas cuyo principal objetivo es facilitar la búsqueda e interpretación de información contenida en ellas [18].

Dentro de lo que se denomina procesamiento de imágenes se engloban una serie de técnicas que comprenden operaciones cuyo origen es una imagen y cuyo resultado final es la imagen procesada. El valor del píxel en la imagen de salida puede estar en función del valor que tenía en la imagen de entrada, de los valores de sus vecinos o del valor de todos los puntos de la imagen de entrada. El análisis de imágenes tiene como parámetro de entrada una imagen y genera como salida información, en la mayoría de los casos estadística, referente al contenido de la imagen de entrada.

En la bibliografía especializada en el tema se puede encontrar una gran variedad de formas de clasificación de las técnicas pertenecientes a esta área de la ingeniería [1], [9]. A continuación se muestra un resumen de estas clasificaciones:

- Mejora de la calidad.
- Técnicas de restauración.
- Compresión.

- Segmentación.

Algunos trabajos similares al propuesto en este trabajo de tesis, y que permitieron determinar las características que debe reunir una herramienta de este tipo para que pueda ser considerada competitiva, son mencionados a continuación.

IPXS es un entorno educacional para el diseño y programación de aplicaciones en un FPGA. Su principal objetivo es el de facilitar el trabajo de los estudiantes en la implementación de algoritmos para el procesamiento de imágenes. Consiste de dos partes: las herramientas IPXStools y el controlador de memoria (Memdrvs) [5].

Anthony Edward propone en [15] la implementación de algoritmos de procesamiento de imágenes sobre un FPGA. Los algoritmos propuestos son: filtro basado en una plantilla, operadores morfológicos y convolución. La implementación de los algoritmos es hecha sobre FPGA's, Altera FLEX 10k100 y Xilinx Virtex XCV300BG352, y en Matlab para hacer una comparación de los resultados obtenidos.

En [29] Medany presenta una arquitectura de hardware para un laboratorio remoto basado en tarjetas de desarrollo de circuitos electronicos digitales que tienen como parte central un FPGA. Este laboratorio remoto es empleado para impartir cursos de diseño digital, en donde el estudiante puede accesar al FPGA mediante internet, ocupando la conexión remota de Windows XP o una página web.

1.2. Planteamiento

Es difícil que alumnos e investigadores que aplican el procesamiento y análisis digital de imágenes en sistemas autónomos, cuenten con una herramienta que permita evaluar los resultados obtenidos por un algoritmo específico antes de ser implementado en la aplicación final. La carencia de una herramienta de este tipo, dificulta y retarda el proceso de diseño y modelado del algoritmo repercutiendo en largos tiempos perdidos en esta fase.

El presente trabajo propone el diseño de una herramienta para el ámbito académico y de investigación enfocada al área de procesamiento y análisis de imágenes aplicado a sistemas autónomos mediante el uso de un FPGA. El sistema propuesto tiene como elemento principal un FPGA Spartan 3E-500 de la compañía Xilinx, cuya función es albergar los algoritmos de procesamiento de imágenes modelados por el usuario; la herramienta es complementada por una aplicación en Windows XP programada en C++ Builder 6.0, la cual permite visualizar los resultados generados por los algoritmos contenidos en el FPGA.

1.3. Justificación

En la actualidad, el procesamiento y/o análisis digital de imágenes es una disciplina ampliamente utilizada por diversas áreas científicas y tecnológicas; algunos ejemplos son citados a continuación:

- Biología y genética.
  - Análisis de huesos y tejidos.
  - Análisis y clasificación de células y material ADN.
- Defensa e inteligencia militar.
  - Interpretación automática de imágenes satelitales en búsqueda de objetivos militares.
  - Reconocimiento y búsqueda de objetivos militares en tiempo real.

- Procesamiento de documentos.
  - Detección y reconocimiento automático de caracteres dentro de un documento.
- Automatización industrial.
  - Inspección visual automática, visión artificial.
  - Análisis de características de piezas manufacturadas en una línea de producción.
- Análisis de materiales.
  - Detección automática de componentes en un material.
  - Creación de superficies 3D y visualización de la estructura interna de un material.
- Fotografía/Video.
  - Composición de escenas con múltiples objetos.
  - Adición de efectos especiales.
- Medicina.
  - Análisis de imágenes de rayos X y resonancia magnética.
  - Sistemas de ayuda al diagnostico, tratamiento y seguimiento de patologías.
- Identificación biométrica.
  - Reconocimiento de personas por medio de huellas digitales, reconocimiento facial, análisis de retina, etc.

Con la finalidad de obtener información objetiva de la escena captada por un sistema de adquisición, se utiliza un procesador en la implementación de algoritmos de procesamiento y análisis digital de imágenes. El procesador empleado puede ser el utilizado por una computadora personal, un microntrolador, un Procesador Digital de Señales (DSP, *Digital Signal Processor*) o un procesador de aplicación específica modelado en un CDC. La primer opción queda descartada cuando se requiere que dicho algoritmo sea realizado por un sistema autónomo.

De las opciones restantes, el modelado de un procesador de aplicación específica en un CDC presenta las siguientes ventajas: 1. Optimización de recursos, permite modelar únicamente las funciones necesarias en la aplicación. 2. Modelado de arquitecturas paralelas, repercutiendo en altas velocidades de procesamiento. 3. Uso de HDLs estandarizados, lo cual genera diseños portables entre diversas tecnológias. Desafortunadamente, las herramientas que permiten agilizar el diseño e implementación de algoritmos de procesamiento y análisis digital de imágenes sobre un CDC son escasas y limitadas en sus características.

Teniendo en cuenta la importancia del papel que desempeña el procesamiento y análisis digital de imágenes en nuestros días, aunado a la falta de herramientas que faciliten la tarea del diseño de nuevos algoritmos sobre un CDC, se propone el diseño e implementación de un **Sistema de Procesamiento y Análisis Digital de Imágenes sobre un FPGA**, el cual será utilizado como una herramienta que brindará una base sólida en el diseño e implementación de prototipos que incluyan algoritmos de procesamiento y/o análisis digital de imágenes. La función principal de esta herramienta es agilizar el diseño y modelado de un algoritmo y permitir visualizar los resultados antes de implementarlos en un sistema final. Esta herramienta tendrá como principales campos de operación los ámbitos académico y de investigación vinculados al área de procesamiento digital de imágenes.

1.4. Objetivos

Diseñar e implementar un sistema de procesamiento y análisis digital de imágenes, cuya unidad de procesamiento se encuentre montada sobre los recursos de un FPGA, además, debe contar con una interfaz de usuario que permita el manejo y la visualización de las imágenes a procesar y los resultados obtenidos.

1.4.1. Objetivos secundarios

A continuación se muestran los objetivos secundarios, mismos que son necesarios para poder cumplir con el objetivo planteado:

- Disponer de un proceso principal dentro del FPGA, procesador de aplicación específica, que sea capaz de administrar el tiempo de ejecución de todos y cada uno de los algoritmos de procesamiento de imágenes implementados, administrar el sistema de memoria usado por los mismos, controlar la recepción de imágenes procedentes de la PC (*Personal Computer*) y la transmisión de los resultados obtenidos hacia ésta.

- Contar con un sistema de memoria que permita el almacenamiento temporal de la imagen a procesar por el algoritmo contenido en el FPGA, así como los resultados obtenidos de aplicar un algoritmo de procesamiento o análisis a dicha imagen.

- Contar con una interfaz de usuario que disponga de las siguientes funciones:

  - Apertura de imágenes para su visualización y transferencia.
  - Control de un puerto USB (*Universal Serial Bus*) para establecer una transferencia de información con la unidad montada en el FPGA.
  - Guardar imágenes obtenidas del procesamiento y otros resultados.
  - Para cada algoritmo, la interfaz debe contar con una serie de parámetros de control que le permitan su correcto funcionamiento.

- Implementar un protocolo de transferencia de información entre el procesador implementado en el FPGA y la interfaz de usuario programada en una computadora personal; este protocolo servirá para transferir comandos, imágenes y resultados.

- Proponer una metodología en software y hardware que permitan al usuario del sistema agregar o quitar cierto algoritmo de procesamiento de imágenes.

1.5. Contribuciones

En este trabajo de tesis fue diseñado e implementado un sistema de procesamiento y análisis digital de imágenes sobre los recursos de un FPGA, empleando un lenguaje descriptor de hardware. Se diseñó una interfaz de usuario de tipo GUI diseñada mediante el paradigma de programación orientada a objetos, la cual es utilizada en el manejo de las imágenes a procesar y/o analizar y la visualización de los resultados obtenidos para la interpretación de los mismos. Se creó una metodología utilizada en la integración de algoritmos de procesamiento y/o análisis de imágenes en el sistema. La unión de estos módulos, forma la principal contribución de este trabajo de tesis: “herramienta para el procesamiento y análisis de imágenes sobre un FPGA”.

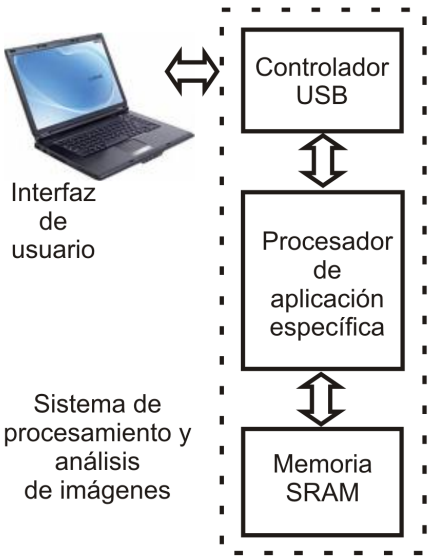


Figura 1.1: Diagrama a bloques del sistema propuesto.

1.6. Metodología de desarrollo

La metodología de desarrollo utilizada en la implementación de este trabajo de tesis se divide en cuatro fases:

- Modelado de un procesador de aplicación específica para el procesamiento y análisis digital de imágenes.
- Diseño e implementación de un sistema con base en el procesador modelado.
- Diseño de una interfaz de usuario para el sistema diseñado.
- Propuesta de una metodología para el diseño y modelado de algoritmos de procesamiento y análisis digital de imágenes en el procesador modelado.

En la Figura 1.1 se muestra el diagrama de bloques de cada fase que compone al sistema de procesamiento y análisis digital de imágenes propuesto.

1.6.1. Modelado de un procesador de aplicación específica

Esta fase del proyecto consiste en el modelado de un procesador de aplicación específica, este procesador estará enfocado al procesamiento digital de imágenes.

El uso de herramientas de Automatización para Diseño Electrónico (EDA, *Electronic Design Automation*) facilitan el diseño de sistemas electrónicos; éstas son divididas en herramientas hardware y herramientas software. Las herramientas EDA software son también denominadas como herramientas de Diseño Asistida por Computadora (CAD, *Computer Aided Design*) o simplemente EDA-CAD.

En el modelado del procesador se utiliza como herramienta EDA-hardware la tarjeta Nexys-2, plataforma de diseño de sistemas digitales construida con base en el FPGA Spartan 3E-500 de la compañía Xilinx; la herramienta EDA-CAD empleada es Xilinx ISE (*Integrated Software Environment*) Foundation 8.2i y VHDL.

La metodología utilizada en el modelado del procesador es la descendente (*Top-down*), la cual inicia visualizando al sistema a diseñar con un nivel de abstracción alto, para luego dividirlo en módulos jerárquicamente inferiores, cada uno de estos módulos puede ser también

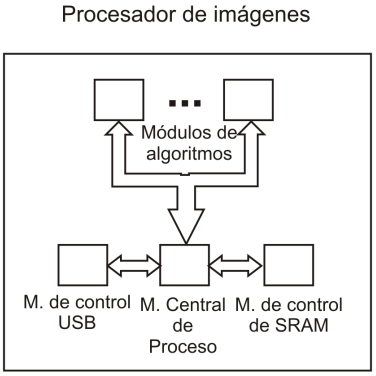


Figura 1.2: Procesador de aplicación específica propuesto.

dividido; el nivel de detalle del módulo de menor jerarquía depende directamente de las primitivas proporcionadas por la herramienta utilizada.

El modelado del procesador de imágenes se subdivide en los siguientes módulos HDL:

- *Módulo principal*. Se encarga de administrar todas las funciones desarrolladas por el procesador para el procesamiento y análisis de imágenes, como pueden ser el administrar las lecturas y escrituras al módulo de control de comunicación USB, de almacenar las imágenes para su procesamiento y/o análisis así como de los resultados obtenidos, generar los tiempos de ejecución de los distintos módulos de algoritmos incluidos en el procesador y otras peticiones hechas de forma externa.

- *Módulo de control de comunicación USB*. Controla las escrituras y lecturas a un dispositivo externo USB.

- *Módulo de control de almacenamiento SRAM (Static Random Access Memory)*. Controla los accesos de escritura y lectura de un dispositivo de almacenamiento temporal externo de tipo SRAM.

- *Módulos de algoritmos de procesamiento y análisis de imágenes*. Es el conjunto de algoritmos de procesamiento y análisis de imágenes que se encuentran programados dentro del procesador, ejemplos de éstos son: el negativo de la imagen, el histograma, la Trasnformada Discreta Wavelet (DWT, *Discrete Wavelet Transform*), etc.

En la Figura 1.2 se muestra un diagrama a bloques del procesador de aplicación específica propuesto.

1.6.2. Diseño e implementación de un sistema de procesamiento y análisis de imágenes

Considerando que este sistema tiene como elemento central a un procesador de aplicación específica, para su diseño e implementación se sigue la metodología de diseño de un sistema empotrado. Un sistema empotrado o embebido es aquel que combina una parte de hardware con otra parte de software para realizar una función específica [20].

En la Figura 1.3 se muestran las fases de desarrollo para un sistema empotrado. Los elementos que integran al sistema a implementar en esta fase son los siguientes:

- Procesador de aplicación específica modelado en la fase anterior.

- Controlador USB.

- Memoria externa SRAM.

Figura 1.3: Ciclo de desarrollo de un sistema empotrado.

1.6.3. Interfaz de usuario

La interfaz de usuario ha sido desarrollada utilizando el lenguaje de alto nivel C++ Builder 6.0, el cual es un lenguaje de programación visual orientado a objetos. C++ Builder es el encargado de generar el entorno de la interfaz gráfica a partir de sus componentes. Además, cuenta con componentes de entrada o captura de datos y salidas o despliegue de resultados, que para el caso de estudio es de suma importancia, ya que se requiere de apertura de imágenes, cuadros de texto para la descripción de algoritmos, la visualización de resultados, etc. Genera aplicaciones ejecutables en Windows XP portables con sólo agregar algunas Bibliotecas de Enlace Dinámico (DLL, *Dynamic Link Library*), y se pueden desarrollar aplicaciones con varias ventanas hijas que es el caso de la interfaz de usuario que se desea obtener.

Cabe mencionar que la interfaz de usuario es una fase que complementa la implementación del sistema propuesto. La interfaz de usuario es una herramienta que permite hacer peticiones para el procesamiento y/o análisis de imágenes y ayuda en la visualización de los resultados. También ayuda en la agilización del proceso de diseño-modelado-implementación de algoritmos que requieran ser aplicados en sistemas autónomos.

1.6.4. Metodología para la implementación e integración de algoritmos

En esta fase del proyecto se propone una metodología integral de diseño, la cual es una secuencia de procedimientos que facilitan la implementación y evaluación del desempeño de un algoritmo, y permite una fácil implementación e integración del mismo en el sistema de procesamiento y análisis de imágenes. La metodología propuesta fue dividida en cinco fases, la descripción de la mismas es la siguiente:

- *Fase 1. Análisis y especificación del algoritmo.* En esta fase se definen en forma clara y precisa los aspectos relevantes de la función que va a desempeñar el algoritmo de procesamiento y/o análisis de imágenes que se va a integrar al sistema.

- *Fase2. Diseño conceptual integrado.* Para esta fase de la metodología se realiza el diseño conceptual del algoritmo a implementar, es decir, se realiza un diseño basado en las señales de entrada/salida del algoritmo y se describe una aproximación del

funcionamiento del mismo. Por otro lado, para el manejo y la ejecución del algoritmo desde la interfaz de usuario, en esta fase también se realiza un diseño conceptual visual (botones, entradas de texto, imágenes, etc.) de la interfaz de algoritmo a implementar con la cual el usuario interactuará, y que está contenida dentro de la interfaz de usuario.

- *Fase 3. Implementación paralela.* De acuerdo a los diseños conceptuales del algoritmo a implementar y la modificación de la interfaz de usuario, en esta fase se realizan una serie de procedimientos de manera separada para la descripción de hardware del algorimo dentro del procesador, y la implementación y codificación de la ventana de interfaz del nuevo algoritmo, dentro de la interfaz de usuario (software).

- *Fase 4. Integración de los componentes.* En esta fase se integran la nuevas versiones del procesador, que contiene al nuevo algoritmo implementado, y la interfaz de usuario, para la evaluación del funcionamiento del algoritmo en la siguiente fase.

- *Fase 5. Evaluación del algoritmo.* En esta última fase se realiza la evaluación del algoritmo sumado al sistema. La evaluación consiste en determinar si el algorimo ha cumplido con los requerimientos de la Fase 1, y que los resultados obtenidos concuerden con la teoría; de no ser así, es posible regresar a cualquier fase de la metodología para determinar cual o cuales son las posibles causas y poder corregirlas.

2. PROCESAMIENTO Y ANÁLISIS DE IMÁGENES

En este capítulo se exponen los conceptos básicos relacionados con imágenes digitales; además, después de analizar la bibliografía especializada, se muestra una clasificación de las diversas técnicas de análisis y procesamiento digital de imágenes en función de la sub-área de aplicación. Finalmente, se presenta el estado del arte acerca del procesamiento y/o análisis de imágenes, mediante la utilización de Dispositivos Lógicos Programables, aplicado a sistemas autónomos.

2.1. Representación digital de una imagen

Una imagen analógica puede ser representada en forma aproximada por una serie de muestras igualmente espaciadas, a este proceso se le conoce como discretización; una imagen digital es la discretización tanto en coordenadas como en tonos de gris de una imagen analógica. Entonces, una imagen puede ser definida por una función $f(x, y)$, donde los valores x y y son coordenadas espaciales, y el valor de f en (x, y) es conocido como *intensidad*

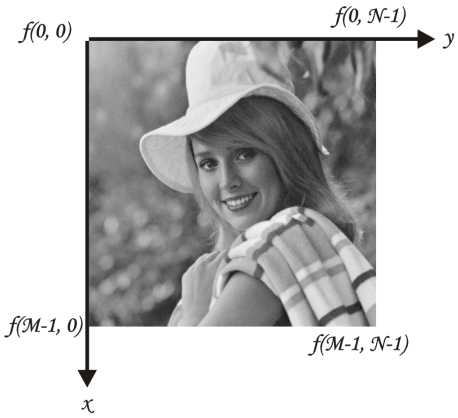


Figura 2.1: Convención de coordenadas para una imagen.

o *nivel de gris* de la imagen en ese punto; se habla de una imagen digital cuando x, y y los valores de f son cantidades finitas y discretas. Una imagen digital está compuesta por un número finito de elementos que tienen posición y valor particulares. Estos elementos son conocidos como *elementos de la imagen*, *pels* o *píxeles* [19], [37].

Para representar una imagen digital se emplea una matriz de M renglones y N columnas, cuyo contenido son cantidades discretas:

$$f(x,y) = \begin{pmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{pmatrix} \tag{2.1}$$

Las posiciones de los píxeles de una imagen se representan con el uso de coordenadas; por convención el origen de la imagen se encuentra en el extremo izquierdo superior, denotado por la coordenada $(x,y) = (0,0)$, que indica que se encuentra en el primer renglón y en la primera columna; la siguiente coordenada del primer renglón es $(x,y) = (0,1)$ como se muestra en la Figura 2.1.

Durante el proceso de digitalización de la imagen se toman decisiones respecto a M, N y a la cantidad discreta de niveles de gris permitidos para cada píxel, denotado por L; el valor de L se restringe a potencias de 2:

$$L = 2^k \tag{2.2}$$

donde k representa el número de bits necesarios para representar un valor de L.

Los niveles de gris están espaciados por igual y se encuentran en el intervalo $[0, L-1]$ conocido como *rango dinámico* de una imagen.

El número de bits que se necesitan para almacenar una imagen digital de $M \times N$ con L diferentes niveles de gris es:

$$b = M \times N \times k \tag{2.3}$$

Así para una imagen típica de 512×512 con 256 niveles de gris ($k = 8$) se necesitan 2,097,512 bits o 262,144 bytes.

La resolución de una imagen expresa con cuanto detalle podemos ver la imagen y depende directamente de M, N y k.

2.2. Criterios de evaluación del procesamiento sobre una imagen digital

Los criterios tomados en cuenta, cuando se valoran los resultados de aplicar algoritmos de procesamiento sobre imágenes digitales, hacen uso de la *distorsión* que sufre la imagen recuperada al aplicar dicho procesamiento.

Distorsión. La medición de la distorsión se subdivide en dos categorías: la subjetiva y la objetiva, [1], [13]. Cuando la calidad de los resultados es medida o evaluada por el humano, se dice que la medida de la distorsión es subjetiva. El uso de esta categoría es poco práctico; es por eso que se emplea con muy poca frecuencia.

Cuando se hace uso de la segunda categoría, los criterios objetivos, para calcular la distorsión entre dos imágenes, comúnmente se emplea una función predefinida. Esta función calcula la diferencia de ambas imágenes, la original y la recuperada, y todos los cambios localizados entre ambas los considera como distorsión, aunque el observador humano no los distinga. De los métodos de medición de la distorsión cuantitativa en imágenes reconstruidas, se encuentran el Error Absoluto Medio (MAE, *Mean Absolut Error*), el Error Cuadrático Medio (MSE, *Mean Square Error*) y la Relación Señal a Ruido de Pico (PSNR, *Peak Signal to Noise Ratio*) [1], [47], donde el MAE se define como:

$$\mathbf{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\tilde{x}_i - x_i| \tag{2.4}$$

el MSE como:

$$\mathbf{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\tilde{x}_i - x_i)^2 \tag{2.5}$$

y la PSNR como:

$$\mathbf{PSNR} = 10\log_{10}\left(\frac{(2^n-1)^2}{\mathbf{MSE}}\right) \tag{2.6}$$

donde N es el número de píxeles en la imagen, n es el número de bits por píxel, x son los elementos de la imagen original y $\tilde{x}$ son los elementos de la imagen recuperada.

Las medidas objetivas no siempre coinciden con las subjetivas, y poseen algunas deficiencias. Una de las deficiencias es que únicamente miden diferencias locales entre píxeles y omiten detalles o efectos visuales globales, como el caso de la distorsión por bloques o efecto pixelado (*blockiness*), que se genera con el uso de algoritmos basados en bloques; la correlación espacial entre bloques adyacentes no es considerada durante el procesamiento y se hacen visibles los límites de cada bloque cuando se reconstruye la imagen. Otros casos de deficiencias ocurren en la distorsión por el efecto del desenfoque (*blurring*), que se encuentra en las imágenes fuera de foco, además, en la distorsión de escalera (*jaggedness of the edge* o *edge jittering*), que se presenta como pasos visibles de escalera donde debe de haber líneas rectas o curvas lisas [31], [25].

2.3. Definición de procesamiento y análisis digital de imágenes

El procesamiento y análisis digital de imágenes es un área de la ingeniería que se dedica a la manipulación y análisis de la información contenida en una imagen digital por medio de un procesador.

Análisis digital de imágenes. Se encarga de la extracción de mediciones, datos o información contenida en una imagen. Incluye aquellas técnicas cuyo principal objetivo es facilitar la búsqueda e interpretación de información contenida en ellas. Un sistema de análisis de imágenes se distingue debido a que tiene como parámetro de entrada una imagen, y cuyo producto es comúnmente una salida numérica, en lugar de otra imagen. Esta salida es información, en la mayoría de los casos estadística, referente al contenido de la imagen de entrada [24].

Procesamiento digital de imágenes. Es el conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar su calidad. Engloba una serie de técnicas que comprenden operaciones donde el origen es una imagen, y el resultado final es otra, ya procesada. El valor del píxel en la imagen de salida puede estar en función del valor que tenía en la imagen de entrada, de los valores de sus vecinos o del valor de todos los puntos de la imagen.

Según González & Woods [19], [18], el objetivo de estas técnicas es procesar y/o analizar una imagen, de tal modo que la resultante sea más adecuada que la imagen original, para cierta aplicación específica. El término *específico* es importante, porque establece que el valor de la resultante esté en función del problema que se trata. De esta manera, un método que es útil para realzar un determinado tipo de imágenes puede no serlo para otras.

El procesamiento y análisis de imágenes se ha desarrollado en respuesta a tres de los más grandes problemas concernientes a imágenes [37]:

- La digitalización y codificación de imágenes que facilite la transmisión, impresión y almacenamiento de las mismas.
- Mejora y restauración de una imagen para interpretar más fácilmente su contenido sobre una superficie.
- Descripción y segmentación de imágenes para una etapa inicial de visión robótica.

En la bibliografía especializada en el tema del procesamiento y análisis de imágenes se pueden encontrar una gran variedad de formas de clasificación de las técnicas pertenecientes a esta área de la ingeniería [1], [9]. La mayoría de las técnicas de procesamiento y análisis de imágenes entran dentro de la clasificación que se resume a continuación:

- Mejora de la imagen.
- Restauración de la imagen.
- Compresión de imágenes.
- Segmentación.

2.3.1. Mejora de la imagen

Se trata del conjunto de técnicas más sencillas y más utilizadas en el tratamiento digital de imágenes. La mejora de la imagen es un proceso cuyo objetivo principal es destacar detalles de interés de una imagen, y/o mejorar la calidad de la misma con la finalidad de que el aspecto resultante sea más adecuado para una aplicación específica. El resultado obtenido de una técnica empleada en la mejora de una imagen es evaluado mediante un criterio subjetivo; es decir, el encargado de determinar cuanto una técnica ha mejorado cierto aspecto de una imagen es un experto en la materia mediante la inspección de los resultados [6].

Todos aquellos algoritmos de procesamiento de imágenes destinados a resaltar, agudizar y/o contrastar determinados aspectos de la imagen, y también aquellos que ayudan a eliminar efectos no deseados sobre ellas, como toda clase de ruido (aditivo, sustractivo, multiplicativo, etc.), son técnicas de mejora de la imagen [52].

Es importante destacar que al utilizar algoritmos de mejora de la imagen se deben considerar los siguientes aspectos [37]:

- No se añade información nueva que no este presente en la imagen. Tan sólo se resalta la información existente, para que pueda ser apreciada de mejor manera por el ojo humano.
- La valoración de los resultados es subjetiva, debido a que no existe un criterio para saber que tanto se mejoró la imagen original, por lo regular se emplean varias pruebas sobre la imagen hasta obtener los resultados más adecuados.
- Muchos de estos algoritmos suelen ser utilizados para formar parte de otros algoritmos de procesamiento de imágenes más complejos.

El conjunto de algoritmos de mejora de imagen comúnmente es dividido en dos grandes grupos:

- *Algoritmos en el dominio espacial.* Se refiere a métodos que procesan una imagen píxel por píxel, y en ocasiones tomando en cuenta un conjunto de píxeles vecinos.
- *Algoritmos en el dominio de la frecuencia.* Frecuentemente, estos métodos son aplicados sobre los coeficientes resultantes de la Transformada de Fourier de una imagen.

2.3.1.1. Métodos en el dominio espacial

Las operaciones espaciales procesan a una imagen haciendo un recorrido por cada uno de sus píxeles, y aplicando una transformación sobre ellos, esto es:

$$g(x, y) = T[f(x, y)] \tag{2.7}$$

donde $f(x, y)$ es la imagen de entrada, $g(x, y)$ es la imagen procesada o resultante, y T es un operador que se aplica sobre la imagen, el cual es definido sobre los vecinos del píxel (x, y). La manera más simple de definir a T, es cuando los vecinos tomados en cuenta son una matriz de 1×1, de esta forma $g(x, y)$ depende únicamente de (x, y), que es el píxel que se está procesando. La función de transformación o también llamada *mapping* queda de la siguiente manera:

$$s = T(r) \tag{2.8}$$

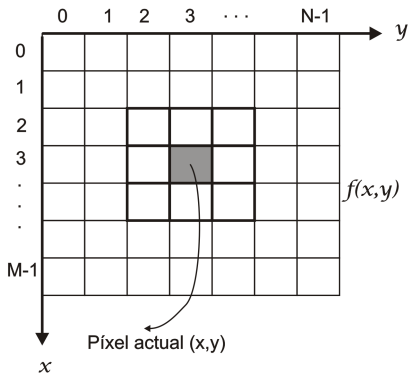


Figura 2.2: Plantilla de vecinos del píxel actual de 3x3.

que por simplicidad se usa r para representar al píxel que se está procesando, y s para definir al píxel resultante.

El conjunto de píxeles vecinos al píxel actual suele llamarse *kernel*, *ventana* o *plantilla*. El ejemplo de una plantilla de píxeles vecinos de 3×3 se muestra en la Figura 2.2.

Algunas de las técnicas de mejora de la imagen más empleadas son: el negativo de la imagen, el histograma, el aplanado o ecualización del histograma, y varios filtros espaciales; estas técnicas son descritas a continuación.

2.3.1.1.1. Negativo de la imagen

El negativo de una imagen con niveles de gris en el rango $[0, L-1]$, es obtenido al restar cada uno de sus píxeles r del valor máximo de nivel de gris $(L-1)$ permitido en la imagen [18], esto es:

$$s = (L-1) - r \tag{2.9}$$

La Figura 2.3 muestra la función de transformación empleada en el negativo de la imagen. Se trata de una función que modifica cada píxel de la imagen de acuerdo a la Ec. 2.9.

Este procesamiento resulta útil cuando se quiere resaltar detalles blancos o grises que se encuentran en regiones oscuras de una imagen, especialmente cuando las áreas negras son dominantes en tamaño. Además, este procesamiento es análogo al negativo fotográfico que suele ser usado en áreas como la medicina o la industria.

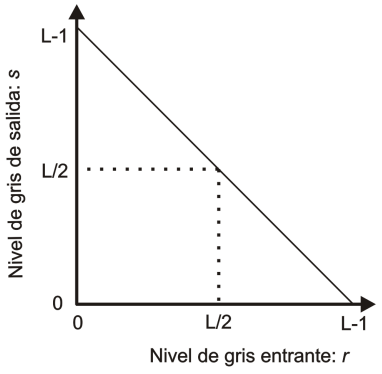


Figura 2.3: Función de transformación del negativo de la imagen

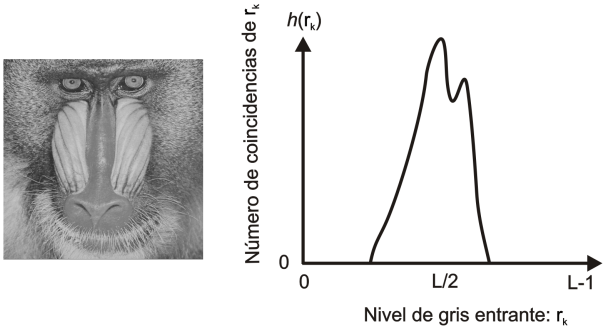


Figura 2.4: Imagen de Baboon y su histograma.

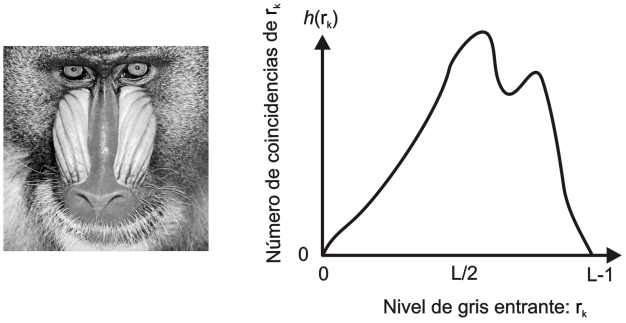


Figura 2.5: Imagen e histograma de Baboon después de la ecualización.

2.3.1.1.2. Histograma

El histograma de una imagen con niveles de gris en el rango $[0, L-1]$, es una función discreta que representa el número de píxeles que posee cada nivel de gris en la imagen, o dicho en otras palabras, es la frecuencia relativa de ocurrencia de cada nivel de gris en la imagen [37], [18]:

$$h(r_k) = n_k \tag{2.10}$$

donde r_k es el k-ésimo nivel de gris y n_k es el número de píxeles en la imagen que contienen dicho nivel.

La Figura 2.4 muestra un ejemplo de la imagen de Baboon y su histograma. Como se aprecia, el rango dinámico de esta imagen es estrecho.

2.3.1.1.3. Ecualización del histograma

La forma del histograma de una imagen revela información acerca de su *contraste*; por contraste se entiende a la facilidad con que el ojo humano puede diferenciar a varios objetos o áreas en una imagen; aquellas con histograma estrecho o rango dinámico bajo, posee poco contraste, mientras que otras con histograma amplio o rango dinámico alto, poseen mucho contraste. Así, una técnica de mejora de contraste ayudará a expandir el histograma de una imagen de manera uniforme en todo el rango dinámico de niveles de gris, con el objetivo de aumentar el contraste de la imagen.

La ecualización o aplanado del histograma es una técnica de mejora de la imagen que mapea el histograma de una imagen de entrada hacia otro histograma, que posee un rango dinámico más uniforme y distribuido con el objetivo de obtener una imagen resultante con

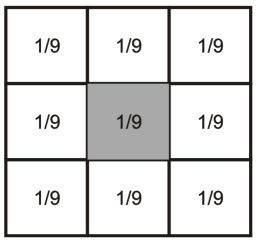


Figura 2.6: Ejemplo de plantilla de píxeles vecinos utilizada en el filtrado espacial.

mayor contraste que el que tenía la original [1].

$$v_k = \frac{(L-1)(S_k - S_k min)}{(S_k max - S_k min)} \tag{2.11}$$

donde v_k es el k-ésimo nivel de gris de la imagen resultante y S_k es la k-ésima probabilidad acumulada de los niveles de gris.

La Figura 2.5 muestra un ejemplo de ecualización o aplanado del histograma. Al histograma de Baboon de la Figura 2.4 se le aplica la ecualización, obteniendo un histograma con mayor rango dinámico y una imagen con más contraste.

2.3.1.1.4. Filtrado espacial

El filtrado espacial es una técnica de mejora de la imagen empleada para eliminar el ruido que se encuentra presente en las imágenes. Este filtro se encuentra en función de los píxeles vecinos al píxel que se está procesando en dicha imagen. El grado de filtrado sobre la imagen depende de la cantidad de píxeles vecinos que se están utilizando [52].

La matriz de píxeles vecinos se conoce como plantilla. El píxel nuevo es obtenido al promediar el conjunto de píxeles del vecindario.

Una interpretación del filtrado espacial es convolucionar la imagen original con la respuesta impulsional del filtro, la cual se representa mediante la plantilla de promediado.

La Figura 2.6 muestra un ejemplo de una plantilla de píxeles de 3×3 usada para el filtrado espacial.

2.3.1.1.5. Suavizado direccional

El filtrado espacial tiene el inconveniente de desenfocar los contornos de la imagen procesada, una alternativa que subsana este problema es el filtrado direccional. Este método de mejora de la imagen consiste en aplicar promediados espaciales en diversas direcciones sobre la plantilla de promediado. De todos los promedios en todas las direcciones se elige aquella dirección cuya diferencia entre el valor filtrado y el píxel actual sea la menor [1], [52].

Como se muestra en la Figura 2.7, para cada plantilla ya sea de 3×3, 5×5, etc., se tienen 4 posibles direcciones: D1, D2, D3 y D4, las cuales pasan por el píxel que se está procesando.

Considerando la plantilla ejemplo de la Figura 2.8 se tiene lo siguiente:
D1 = (35 + 55 + 255) / 3 = 115
D2 = (68 + 55 + 5) / 3 = 42.6 $\approx$ 43
D3 = (200 + 55 + 30) / 3 = 95
D4 = (102 + 55 + 78) / 3 = 78.3 $\approx$ 78.
Luego se prosigue a calcular las diferencias entre el píxel actual y cada uno de los promedios.
$|55 - 115| = 60$

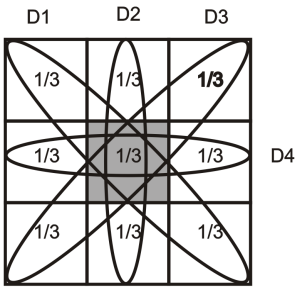


Figura 2.7: Ejemplo de plantilla de promediado empleada por el suavizado direccional.

$|55 - 43| = 12$
$|55 - 95| = 40$
$|55 - 78| = 23$

Para este caso la diferencia menor fue 12 que pertenece a D2, por lo tanto el valor del píxel nuevo será el valor del promedio de la dirección D2 o sea: 43. Este valor se tiene que almacenar en la posición del píxel que se está procesando, pero en la imagen resultante.

2.3.1.1.6. Filtrado de mediana

El filtrado de mediana es un método análogo al filtrado espacial, sólo que en lugar de calcular el promedio se calcula la mediana del conjunto de píxeles vecinos al píxel que se está procesando [1], [52].

La mediana se obtiene ordenando los valores contenidos en la plantilla y escogiendo el que está en el centro. En la Figura 2.8 se muestra un ejemplo de plantilla de 3 x 3 con el

35	68	200
102	55	78
30	5	255

Figura 2.8: Plantilla de píxeles vecinos utilizada en el filtro de mediana.

píxel actual situado en el centro. Para calcular la mediana del píxel actual, primero, éste junto con los 8 píxeles vecinos son ordenados usando un ordenamiento natural: 5, 30, 35, 55, 68, 78, 102, 200, 255, después, se localiza el elemento que se encuentra a la mitad de todos :5, 30, 35, 55, **68**, 78, 102, 200, 255. De esta manera el valor del píxel nuevo será 68, y se tendrá que almacenar en la imagen resultante sobre la posición del píxel actual.

2.3.1.1.7. Ampliación y reducción de imágenes por interpolación bilineal

La interpolación bilineal es una técnica de mejora de la imagen que entra dentro de las *operaciones geométricas* de la imagen. Una operación geométrica es aquella que cambia el tamaño, forma u orientación de una imagen. No se puede considerar como un filtro, pero corresponden a transformaciones útiles en el procesamiento digital de imágenes. Se usan cuando se requiere escalar, orientar o empalmar una imagen. En ocasiones las operaciones geométricas cambian el tamaño de la imagen, como es el caso de la interpolación bilineal que aumenta o disminuye el tamaño de una imagen en la proporción deseada.

Figura 2.9: (a) Imagen de Boat reducida al 75 %. (b) Imagen de Boat original. (c) Imagen aumentada al 125 %

Dada una función $f(i,j)$ con M renglones y N columnas, la cual representa a una imagen, el método de la interpolación bilineal consiste en encontrar el cuadro definido por los puntos (i,j) y $(i+1,j+1)$ que contiene al punto (p,q) que proviene del mapeo de un punto exacto (i',j'), para esto, se deben calcular las contribuciones por cercanía de los valores conocidos de la función en $\{(i,j),\ (i+1,j),\ (i,j+1),\ (i+1,j+1)\}$.

$$p = i'F_1,\ q = j'F_2 \ donde\ F_1 = \frac{N}{N'}\ y\ F_2 = \frac{M}{M'} \tag{2.12}$$

M' y N' son las dimensiones de la nueva imagen, p y q son los valores escalados de la imagen original en el punto(i,j). Se tendrán cercanías horizontales:

$$\alpha = 1 - C_1,\ \beta = 1 - C_2 \tag{2.13}$$

y verticales:

$$\alpha' = 1 - D_1,\ \beta' = 1 - D_2 \tag{2.14}$$

donde

$$C_1 = p - [p]\,,\ C_2 = 1 - C1\ y\ D_1 = q - [q]\,,\ D_2 = 1 - D_1 \tag{2.15}$$

el operador [ ] calcula la parte entera de su argumento. Las contribuciones se obtienen al multiplicar las cercanías por los valores de la función conocida:

$$f'(i',j') = \alpha\alpha' f(i,j) + \beta\alpha' f(i+1,j) + \alpha\beta' f(i,j+1) + \alpha'\beta' f(i+1,j+1) \tag{2.16}$$

donde $f'(i',j')$ es la imagen resultante de aplicar la ampliación o reducción por interpolación bilineal a $f(i,j)$, mediante un *escalado isométrico*, en el cual se guardan las proporciones verticales y horizontales de la imagen original.

En la Figura 2.9, la imagen de Boat se ha ampliado y reducido utilizando el método de interpolación bilineal, como se muestra, al aplicar este proceso, las proporciones horizontales y verticales de la imagen original (b) se mantienen.

2.3.1.2. Métodos en el dominio de la frecuencia

En 1822, Jean Baptiste Joseph Fourier publicó su libro titulado "La Teoría Analítica del Calor" (*La Théorie Analitique de la Chaleur*) [16], teniendo entre sus principales aportaciones la demostración de que cualquier función que se repite periódicamente puede ser expresada como la suma de senos y/o cosenos de diferentes frecuencias, cada uno multiplicado por coeficientes diferentes (*Series de Fourier*).

Una función no periódica-finita, también pueden ser llevada al "dominio de Fourier", expresándola como una integral de senos y/o cosenos. A esto se le conoce como la *Transformada de Fourier* de dicha función, que al igual que las series de Fourier, recuperan a la

función sin perdida de información al aplicar el proceso inverso (*Transformada de Fourier Inversa*).

La transformada de Fourier de dos dimensiones $F(u, v)$, es una herramienta útil cuando se requiere aplicar mejoras a una imagen en el dominio de la frecuencia, ya que una imagen es considerada como una función $f(x, y)$ finita:

$$F(u,v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-j2\pi(ux+vy)} dx\, dy \tag{2.17}$$

y la transformada inversa se expresa de manera similar:

$$f(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u,v) e^{j2\pi(ux+vy)} du\, dv \tag{2.18}$$

Sin embargo, debido a que las imágenes a tratar son de tipo discreto, es de interes trabajar con la Transformada Discreta de Fourier (DFT, *Discrete Fourier Transform*).

La DFT de una imagen de tamaño $M \times N$ está dada por la siguiente ecuación:

$$F(u,v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux/M+vy/N)} \tag{2.19}$$

donde $u = 0, 1, 2, ..., M-1$ y $v = 0, 1, 2, ..., N-1$ son las variables en frecuencia. Teniendo inicialmente a $F(u, v)$, $f(x, y)$ se obtiene mediante la transformada inversa:

$$f(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi(ux/M+vy/N)} \tag{2.20}$$

donde $x = 0, 1, 2, ..., M-1$ y $y = 0, 1, 2, ..., N-1$ son las variables espaciales de la imagen.

La DFT de una imagen está compuesta por una parte real $R(u, v)$ y otra imaginaria $I(u, v)$. A continuación se expresan el espectro de Fourier, el ángulo de fase y el espectro de potencia respectivamente, útiles cuando se trabaja en el dominio de la frecuencia:

$$|F(u,v)| = \left[R^2(u,v) + I^2(u,v)\right]^{1/2} \tag{2.21}$$

$$\Phi(u,v) = \arctan\left[\frac{I(u,v)}{R(u,v)}\right] \tag{2.22}$$

$$P(u,v) = |F(u,v)|^2 = R^2(u,v) + I^2(u,v) \tag{2.23}$$

Además, existen dos relaciones entre las muestras que se tienen en el dominio espacial y el de la frecuencia:

$$\Delta u = \frac{1}{M\Delta x} \tag{2.24}$$

y

$$\Delta v = \frac{1}{N\Delta y} \tag{2.25}$$

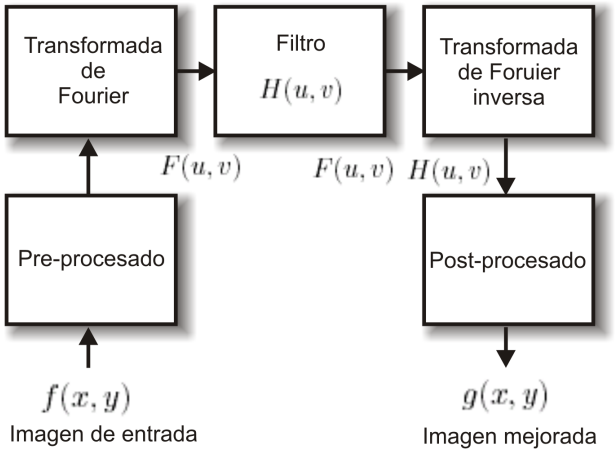


Figura 2.10: Proceso de filtrado en el dominio de la frecuencia.

2.3.1.2.1. Filtrado en el dominio de la frecuencia

Una función de transferencia de filtrado $H(u,v)$ es aquella que actúa sobre la transformada de Fourier de una imagen $F(u,v)$, y permite suprimir ciertas frecuencias mientras deja a otras sin cambio alguno. Las frecuencias bajas son responsables de la mayoría de los niveles de gris de una imagen sobre áreas suaves. Mientras que las frecuencias altas tienen que ver con los detalles de la imagen, como son los bordes y el ruido [18], [1].

La realización de filtros en el dominio de la frecuencia para mejoras en las imágenes, requiere del proceso que se describe a continuación:

1. Multiplicación de la imagen digital $f(x,y)$ por un factor de $(-1)^{x+y}$ con el fin de recorrer la transformada a la coordenada $(M/2, N/2)$.

2. Calcular $F(u,v)$ con la ayuda de un procesador, la DFT de (1).

3. Multiplicar $F(u,v)$ por la transformada de Fourier de la función de transferencia del filtro, $H(u,v)$.

4. Calcular la DFT inversa de (3).

5. Obtener la parte real de (4).

6. Multiplicar el resultado en (5) por $(-1)^{x+y}$.

La Figura 2.10 muestra el diagrama de bloques del procedimiento de filtrado en el dominio de la frecuencia, incluye etapas de pre- y post-procesamiento. Las operaciones que realiza un procesador para llevar a cabo este proceso, en ocasiones pueden costarle demasiado tiempo y/o un excesivo uso de sus recursos. Cuando esto sucede, se prefiere llevar a cabo la especificación del filtro en el dominio de la frecuencia, para después implementarlo en el dominio espacial mediante la reducción del filtro a una plantilla del orden de una matriz de 3×3, que se aplica sobre cada píxel de la imagen en forma de convolución.

A continuación se describen dos de los filtros en el dominio de la frecuencia más importantes, estos son el Butterworth y el Gaussiano, ambos en sus versiones pasa-bajas y pasa-altas.

2.3.1.2.2. Filtro Butterworth

La función de transferencia del Filtro Butterworth Pasa Bajas (BLPF, *Butterworth Low Pass Filter*) de orden n con frecuencia de corte D_0 a partir del origen es:

$$H(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}} \tag{2.26}$$

donde

$$D(u,v) = \left[(u - M/2)^2 + (v - N/2)^2\right]^{1/2} \tag{2.27}$$

La función de transferencia del Filtro Butterworth Pasa Altas (BHPF, *Butterworth High pass Filter*) de orden n con frecuencia de corte localizado en D_0 desde el origen está dada por:

$$H(u,v) = \frac{1}{1 + [D_0/D(u,v)]^{2n}} \tag{2.28}$$

2.3.1.2.3. Filtro Gaussiano

El Filtro Gaussiano Pasa Bajas (GLPF, *Gaussian Low pass Filter*) con frecuencia de corte D_0 está dado por la siguiente expresión:

$$H(u,v) = e^{-D^2(u,v)/2D_0^2} \tag{2.29}$$

donde $D(u,v)$ es la distancia desde la frecuencia cero hasta la posición $(M/2, N/2)$.

La función de transferencia del Filtro Gaussiano Pasa Altas (GHPF, *Gaussian High pass Filter*) con frecuencia de corte localizada a una distancia D_0 desde el origen esta dada por:

$$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2} \tag{2.30}$$

2.3.2. Restauración de la imagen

Estas técnicas tienen por objetivo corregir la degradación, en donde se supone un mecanismo concreto de degradación, sufrida por una imagen. El porqué utilizar la restauración de la imagen es debido a que en algunas ocasiones estas suelen ser degradadas al alterar los niveles de gris de cada píxel o distorsionadas al desplazar la posición de los píxeles varias posiciones de su posición original. En general, las imágenes pueden ser degradadas debido a las siguientes razones [1]:

- Imperfección en el sistema de captura de la imagen.
- Imperfección en el canal de transmisión de la imagen.
- Degradación debida a condiciones atmosféricas.
- Degradaciones sobre la imagen generadas por movimientos relativos entre la cámara de captura y el objeto.

Al igual que la mejora de la imagen, la restauración de la imagen incluye un conjunto de técnicas de procesamiento que busca obtener mejoras en las imágenes. En la restauración de la imagen generalmente se requiere definir un criterio objetivo, Sección 2.2, para valorar la calidad de la imagen resultante; en este aspecto, es diferente a las técnicas de mejora de la calidad, las cuales basan la evaluación de sus resultados en procedimientos heurísticos y criterios personales [37].

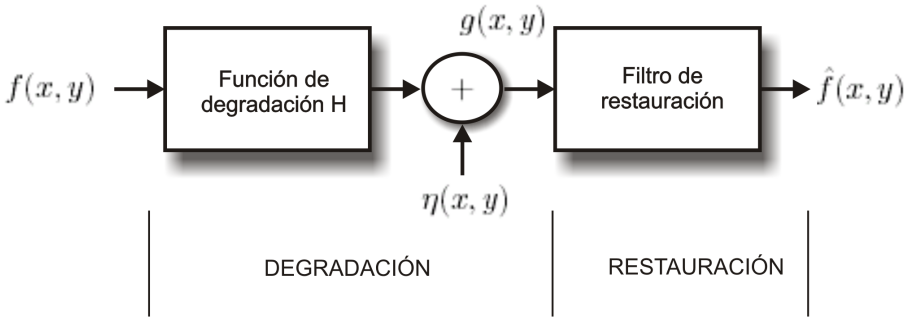


Figura 2.11: Modelo del proceso de degradación-restauración de la imagen.

2.3.2.1. Modelo del proceso de degradación-restauración de la imagen

El proceso de degradación de la imagen es modelado mediante una función de degradación y ruido de tipo aditivo, que operan sobre la imagen $f(x, y)$ para producir la imagen degradada $g(x, y)$. Cuando se cuenta con la imagen degradada, conocimiento acerca de la función de degradación H y el ruido aditivo $\eta(x, y)$, se puede obtener una estimación de la imagen original $\hat{f}(x, y)$. La imagen degradada puede expresarse en el dominio espacial como:

$$g(x, y) = h(x, y) * f(x, y) + \eta(x, y) \tag{2.31}$$

donde $*$ indica la convolución de la función de degradación con la imagen original. Aplicando la transformada de Fourier a la Ecuación 2.31 se llega a la expresión de la imagen degradada en el dominio de la frecuencia:

$$G(u, v) = H(u, v)F(u, v) + N(u, v) \tag{2.32}$$

La Figura 2.11 muestra el modelo del proceso de degradación-restauración que sufre la imagen, que es la base teórica para la restauración de la imagen con ayuda de la transformada de Fourier.

Dentro de las técnicas de restauración de la imagen más empleadas, se encuentran el filtro inverso y el filtro Wiener, que suelen ser usados para eliminar la distorsión por el efecto del desenfoque en las imágenes.

2.3.2.2. Filtro inverso

El filtro inverso es la aproximación más directa hacia la restauración de la imagen. Asume que la degradación de la imagen es causada por una función lineal $h(x, y)$ y que el ruido aditivo es independiente de la imagen. La estimación $\hat{F}(u, v)$ es obtenida al dividir la transformada de Fourier de la imagen degradada por la transformada de la función de degradación [37], [18], [1], esto es:

$$\hat{F}(u, v) = \frac{G(u, v)}{H(u, v)} \tag{2.33}$$

De acuerdo con la Ec. 2.33, la transformada de Fourier de la estimación de la imagen original se puede escribir como:

$$\hat{F}(u, v) = F(u, v) + \frac{N(u, v)}{H(u, v)} \tag{2.34}$$

Esta expresión indica que aún conociendo la función de degradación, no se puede conocer a $F(u, v)$ de forma exacta, debido a que $N(u, v)$ es una función aleatoria cuya transformada de Fourier es desconocida. Además, cuando $H(u, v)$ es o tiene valores cercanos a cero, $N(u, v)/H(u, v)$ se vuelve muy grande afectando a la estimación de $\hat{F}(u, v)$.

(a) (b)

Figura 2.12: (a) Imagen de Man desenfocada. (b) Imagen restaurada al aplicar el filtro Wiener.

2.3.2.3. Filtro Wiener

El filtro Wiener incorpora un conocimiento previo del ruido en el proceso de restauración. Está fundado en la consideración de que la imagen y el ruido son procesos aleatorios. Su objetivo es encontrar una estimación $\hat{f}$ de la imagen original f, tal que el error cuadrático medio entre éstas sea mínimo [37], [18], [1]:

$$e^2 = E\left\{(f - \hat{f})^2\right\} \tag{2.35}$$

La Ecuación 2.35 asume que el ruido y la imagen son no correlacionados y que los niveles de gris de la estimación son una función lineal de los niveles de gris de la imagen degradada. Con estas condiciones, al desarrollar el error mínimo en el dominio de la frecuencia se obtiene:

$$\hat{F}(u,v) = \left[\frac{H^*(u,v)S_f(u,v)}{S_f(u,v)\left|H(u,v)\right|^2 + S_\eta(u,v)}\right] G(u,v) \tag{2.36}$$

$$= \left[\frac{H^*(u,v)}{\left|H(u,v)\right|^2 + S_\eta(u,v)/S_f(u,v)}\right] G(u,v) \tag{2.37}$$

$$= \left[\frac{1}{H(u,v)}\frac{\left|H(u,v)\right|^2}{\left|H(u,v)\right|^2 + S_\eta(u,v)/S_f(u,v)}\right] G(u,v) \tag{2.38}$$

Este resultado es conocido como filtro Wiener, el cual evita los problemas de los ceros en $H(u,v)$ del filtro inverso, donde:

$H(u,v)$ es la transformada de Fourier de la función de degradación
$H^*(u,v)$ es el complejo conjugado de $H(u,v)$
$\left|H(u,v)\right|^2 = H^*(u,v)H(u,v)$
$S_\eta(u,v) = \left|N(u,v)\right|^2$ es el espectro del ruido
$S_f(u,v) = \left|F(u,v)\right|^2$ es el espectro de la señal original.

En el caso donde el ruido es cero, su espectro se desvanece y el filtro Wiener se reduce al filtro inverso. La Figura 2.12(a) muestra el ejemplo de una imagen degradada (desenfocada) y en (b) se tiene la imagen recuperada al aplicar el filtrado Wiener.

2.3.3. Compresión de imágenes

La compresión de una imagen tiene por objetivo reducir la cantidad de bits que ocupa dicha imagen en su representación. La forma más comúnmente utilizada para clasificar la compresión de imágenes, es si dicha compresión introduce pérdida de información o si no lo hace; resultando los métodos de compresión con pérdida de información y métodos de compresión sin pérdida de información [52].

Cuando una imagen es comprimida usando el método de compresión sin pérdida de información, se habla de un proceso reversible. A partir de la representación de la imagen en forma compacta, es posible recuperar la imagen original previa al proceso de compresión. Sin embargo, utilizando estas técnicas la relación de compresión alcanzada no es significativa (aproximádamente 3:1 [19]).

Un método de compresión con pérdida de información es catalogado como un proceso irreversible, esto es, al aplicar el proceso inverso la imagen recuperada no representará fielmente a la imagen original. Estos métodos poseen la ventaja de permitir factores de compresión elevados y posiblemente las pérdidas que el ojo humano logre captar al ver ambas imágenes, la original y la recuperada después de la compresión, sean nulas.

Para el caso de codificación de ficheros de texto se aplican de forma válida los métodos de compresión sin error. En cuanto a las imágenes y al video, se utilizan más los métodos del segundo grupo, esto debido al gran ancho de banda que ocupan en la transmisión por algún medio como podría ser el Internet.

La compresión de datos es el proceso de reducción del volumen de datos necesarios para representar una determinada cantidad de información. Para la descripción de una cantidad de información, se pueden emplear distintas cantidades de datos; una proporción de estos datos contiene información sin relevancia, conocida como redundancia de los datos. La redundancia de datos es un término importante en la compresión digital de imágenes, donde se identifican tres tipos de redundancias: redundancia de codificación, redundancia entre píxeles y la redundancia psicovisual, y la compresión de datos se consigue cuando una o más de estas redundancias se reduce o elimina.

La redundancia entre píxeles, también conocida como redundancia espacial, es la única que entra dentro del procesamiento y análisis de imágenes; las otras redundancias son reducidas o eliminadas en etapas del sistema compresor de imágenes, Sección 2.3.3.1, que no entra dentro de este trabajo.

Redundancia espacial. La reducción de redundancia espacial en imágenes para comprimirlas hace uso de transformadas, como la Transformada Discreta Coseno (DCT, *Discrete Cosine Transform*) y la DWT, las cuales mapean cada píxel de una imagen hacia su respectivo dominio, concentrando la energía de la imagen en regiones de baja frecuencia y generando coeficientes de transformación usados en las etapas siguientes de un sistema de compresión de imágenes [17].

2.3.3.1. Sistema de compresión de imágenes

Un sistema de compresión de imágenes es aquel que tiene por objetivo codificar los datos de una imagen para llevarla a una representación compacta, reduciendo la cantidad de bits que emplea la imagen para su almacenamiento o transmisión, e intentando minimizar la distorsión debida al proceso de compresión. Un sistema de este tipo está compuesto por tres etapas: transformación, cuantización y codificación [21].

En la Figura 2.13 se muestra un diagrama de bloques de un sistema de compresión de imágenes.

Eficiencia en la compresión. Existen algunas maneras de medir que tan eficiente es un sistema de compresión, una de ellas es midiendo el promedio de bits por píxel almacenados en una imagen [21]:

$$\text{bit rate} = \frac{\text{tamaño de la imagen comprimida}}{\text{píxeles en la imagen}} = \frac{C}{N}(\text{bits por píxel}) \tag{2.39}$$

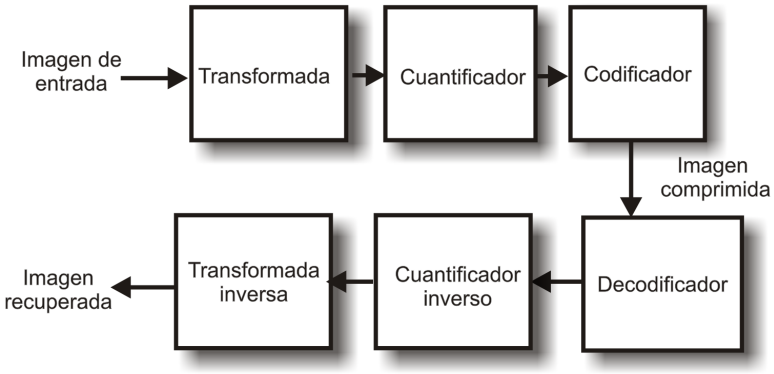


Figura 2.13: Sistema de compresión de imágenes con transformada.

donde N es la multiplicación de las columnas y renglones de la imagen. Otra forma de medir la eficiencia es mediante la relación de compresión:

$$\text{relación de compresión} = \frac{\text{tamaño de imagen original}}{\text{tamaño de imagen comprimida}} = \frac{N \times k}{C} \tag{2.40}$$

donde k es el número de bits por píxel de la imagen original.

De las etapas que forman a un sistema de compresión de imágenes, únicamente la transformación es considerada como una técnica de procesamiento de imágenes; por tal motivo, en este trabajo sólo se contempla analizar a esta etapa en particular.

Existe una gran variedad de algoritmos de transformación que pueden ser utilizados en la etapa de transformación de un sistema de compresión de imágenes; sin embargo, únicamente dos de ellos han conseguido amplia aceptación para este propósito, la DCT y la DWT. A continuación se describen los aspectos más relevantes de estos importantes algoritmos.

2.3.3.2. La Transformada Discreta del Coseno

La DCT fue originalmente propuesta por N. Ahmed *et al.* en el año de 1974 [2]. Actualmente es usada en el bloque de transformación de sistemas de compresión-descompresión de imágenes, en estándares como JPEG y MPEG [49], [23]. Se ha convertido en una transformada muy popular debido a que sus propiedades permiten una forma fácil y efectiva de comprimir imágenes y a su eficiente implementación en software y hardware [21], [1].

De las propiedades de la DCT en la compresión de imágenes, destacan que concentra la energía en un número reducido de coeficientes (compactación de la energía) y que minimiza la interdependencia entre los mismos (decorrelación) [31].

En la etapa de transformación de un sistema de compresión de imágenes, la DCT de dos dimensiones, divide a una imagen en varios bloques. La DCT 2-D de uno de estos bloques $f_{x,y} \| x, y = 0, 1, \cdots, M-1$, con $M \times M$ muestras es:

$$F_{i,j} = \frac{4C(i)C(j)}{M^2} \sum_{x=0}^{M-1} \sum_{y=0}^{M-1} f_{x,y} \cos\left(\frac{\pi(2x+1)i}{2M}\right) \cos\left(\frac{\pi(2y+1)j}{2M}\right) \tag{2.41}$$

donde $i, j = 0, 1, \cdots, M-1$.

La DCT inversa (IDCT,*Inverse Discrete Cosine Transform*) es definida por:

$$f_{x,y} = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} C(i)C(j)F_{i,j} \cos\left(\frac{\pi(2x+1)i}{2M}\right) \cos\left(\frac{\pi(2y+1)j}{2M}\right) \tag{2.42}$$

donde $x, y = 0, 1, \cdots, M-1$, M es un entero potencia de 2, $F_{i,j}$ son los coeficientes de la DCT de dos dimensiones y $C(p),\ p = i, j$ está dado por:

$$C(p) = \begin{cases} \frac{1}{\sqrt{2}} & , \ p = 0 \\ 1 & , \ p \neq 0 \end{cases} \tag{2.43}$$

Debido al amplio uso de la DCT en la compresión de imágenes, han surgido varias versiones que tratan de implementarla de manera más eficiente, como la implementación rápida de la DCT propuesta por Chen en [11], que aprovecha la simetría de la función coseno para reducir el número de operaciones.

Arai propone en [4] un esquema rápido de la DCT aplicado a imágenes. Para el cálculo de la DCT, hace uso únicamente de la parte real de la DFT, y de acuerdo a Winograd [50] emplea la Transformada Rápida de Fourier (FFT, *Fast Fourier Transform*) para el cálculo de los coeficientes de la DCT.

2.3.3.3. La Transformada Discreta Wavelet

Los *wavelets* son funciones generadas a partir de una función base llamada *wavelet madre* por *dilatación* y *translación* en el dominio del tiempo. La wavelet madre es expresada por $\psi(t)$ y las otras wavelets por:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}} \psi(\frac{t-b}{a}) \tag{2.44}$$

donde a y b son dos números reales arbitrarios. Las variables a y b representan la dilatación y la translación en el tiempo de $\psi_{a,b}(t)$ con respecto a $\psi(t)$. La teoría de wavelets se explica de manera similar a la teoría de Fourier, donde una señal se puede descomponer en una serie de funciones sinusoidales con la finalidad de facilitar su análisis. Las wavelets son funciones que se encuentran en el espacio y se emplean para analizar a una señal de interés, con la finalidad de obtener sus características de espacio, tamaño y dirección.

En 1989, Mallat [27] propuso una aproximación a la *multiresolución* mediante la descomposición wavelet de una señal, usando una estructura de filtro piramidal llamada Cuadratura de Filtros Espejo (QMF), y está compuesta por los siguientes filtros FIR [1]:

$$\left.\begin{array}{l} c_{m,n}(f) = \sum_k g_{2n-k} a_{m-1,k(f)} \\ a_{m,n}(f) = \sum_k h_{2n-k} a_{m-1,k(f)} \end{array}\right\} \tag{2.45}$$

donde g y h son filtros pasa-altas y pasa-bajas respectivamente, $a_{m,n}(f)$ es la *aproximación* de la función $f(t)$ en resolución 2^m y $c_{m,n}(f)$ son los *detalles* de la información de $f(t)$ y los coeficientes wavelets en 2^m.

La implementación de la DWT basada en filtros FIR, donde una señal x es filtrada de manera separada por un filtro pasa-bajas $\tilde{h}$ y el filtro pasa-altas $\tilde{g}$ se muestra en la Figura 2.14. Estos filtros forman el banco de *análisis*, el cual produce las sub-bandas y_L y y_H. La

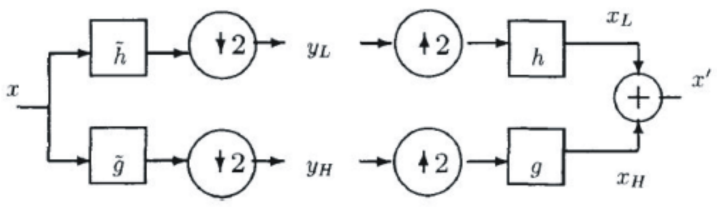


Figura 2.14: Análisis y síntesis de la implementación de la DWT mediante filtros FIR.

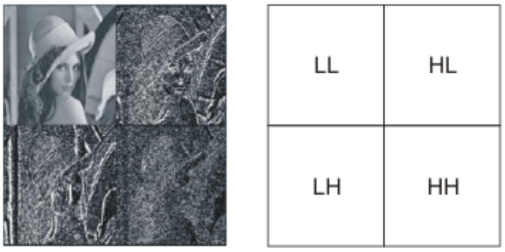


Figura 2.15: DWT de la imagen Lena y las sub-bandas LL, LH, HL y HH.

señal original es reconstruida mediante los filtros del banco de *síntesis* h y g. De esta manera se tiene la siguiente relación:

$$y_L(n) = \sum_{i=0}^{\tau_L - 1} \tilde{h}(i)x(2n-1),\ y_H(n) = \sum_{i=0}^{\tau_H - 1} \tilde{g}(i)x(2n-1) \tag{2.46}$$

donde τ_L y $\tau_H - 1$ son las longitudes de $\tilde{h}$ y $\tilde{g}$ respectivamente.

La extensión de la DWT a dos dimensiones es esencial para la transformación de imágenes con dimensiones $M \times N$. La DWT se aplica a toda la imagen a diferencia de la DCT que se aplica por bloques. La DWT 2-D está compuesta por las siguientes ecuaciones:

$$f_{i+1}(x,y) = \sum_{k_1}\sum_{k_2} g(k_1)g(k_2)f_i(2x-k_1, 2y-k_2) \tag{2.47}$$

$$W^H_{i+1}(x,y) = \sum_{k_1}\sum_{k_2} h(k_1)g(k_2)f_i(2x-k_1, 2y-k_2) \tag{2.48}$$

$$W^V_{i+1}(x,y) = \sum_{k_1}\sum_{k_2} g(k_1)h(k_2)f_i(2x-k_1, 2y-k_2) \tag{2.49}$$

$$W^D_{i+1}(x,y) = \sum_{k_1}\sum_{k_2} h(k_1)h(k_2)f_i(2x-k_1, 2y-k_2) \tag{2.50}$$

Estas ecuaciones generan las sub-bandas LL, LH, HL y HH respectivamente como se muestra en la Figura 2.15. De las Ecuaciones 2.47-2.50, $h(k)$ y $g(k)$ son filtros wavelet de una dimensión, $f_{i+1}(x,y)$ es una aproximación de la imagen $f_i(x,y)$, y es donde se almacena la mayor parte de la información de la imagen, $W^H_{i+1}(x,y)$, $W^V_{i+1}(x,y)$ y $W^D_{i+1}(x,y)$ son los detalles o contornos de la imagen, es decir, las frecuencias altas.

En la Figura 2.15 se muestran las 4 sub-bandas LL, LH, HL y HH generadas a partir de aplicar la DWT a la imagen de Lena.

Dentro de los motivos de aplicar la DWT en tratamiento de señales se encuetra que al tener las *wavelets* localización espacial y en frecuencia, hace viable el tratamiento de cambios bruscos en la imagen, además, como se aplica sobre toda la imagen, no presenta la distorsión por división de la misma y se consigue compresión de energía elevada [21]. La DWT se emplea como la base del estándar de compresión JPEG2000 [1].

2.3.4. Segmentación

A diferencia de los demás algoritmos de procesamiento de imágenes, la segmentación es un proceso mediante el cual se toma como entrada una imagen y genera como salidas atributos extraídos de dichas imágenes.

La segmentación subdivide a una imagen en sus regiones u objetos constituyentes, de tal manera que los píxeles de esas regiones poseen propiedades o atributos idénticos, como niveles de gris, contraste o texturas.

La mayoría de los algoritmos de segmentación están basados en dos propiedades básicas de intensidad de la imagen: la discontinuidad y la similitud.

En la categoría de segmentación mediante discontinuidad, el proceso se realiza dividiendo a la imagen basándose en cambios abruptos en intensidad, como es el caso de la detección de bordes en una imagen.

Con respecto a la segmentación con base en la similitud, ésta es lograda mediante la partición de una imagen en regiones que son similares de acuerdo a un conjunto de criterios predefinidos [18].

2.3.4.1. Detección de contornos

La detección de contornos es por mucho la técnica más común de acercamiento para detectar discontinuidades significativas en los valores de intensidad de una imagen. Estas discontinuidades son detectadas usando la primera y la segunda derivada. La opción de la derivada de primer orden es procesar el gradiente de la imagen. El gradiente de una función en 2-D $f(x, y)$ es definida como el siguiente vector:

$$\nabla \mathbf{f} = \begin{pmatrix} G_x \\ G_y \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} \tag{2.51}$$

Se usa una aproximación de la magnitud del gradiente para realizar los cálculos en los algoritmos de detección de contornos, con el fin de evitar el número de operaciones, esto es:

$$\nabla f \approx |G_x| + |G_y| \tag{2.52}$$

La propiedad fundamental del vector gradiente es que siempre apunta en la dirección del índice máximo de cambio de la función $f(x, y)$, en las coordenadas (x, y).

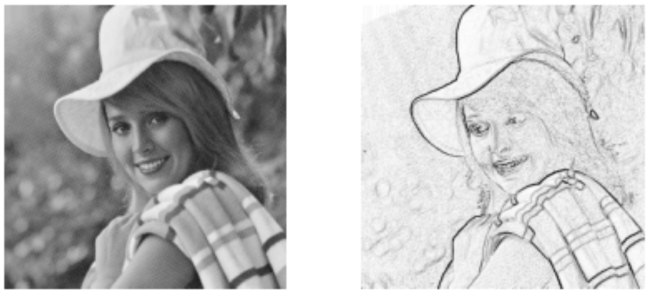
Los *contornos* son los límites de un objeto en una imagen. Para su obtención se usa extracción de bordes.

Un contorno se caracteriza por tener una transición de claro a oscuro o viceversa. Esta transición se puede detectar calculando el gradiente de la imagen en dos direcciones ortogonales. Los operadores gradientes más comunes son los operadores de Roberts y Sobel [18], [52], como se muestra en la Tabla 2.1.

	Horizontal	**Vertical**
Roberts	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Sobel	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

Tabla 2.1: Gradientes Roberts y Sobel

Una de las plantillas detecta contornos horizontales y la otra los verticales, obteniéndose así dos imágenes con dos gradientes: Ig_1, Ig_2.



(a) (b)

Figura 2.16: (a) Imagen de Elaine. (b) Imagen de contornos de Elaine.

Para definir si en un píxel determinado hay un contorno o no, se define un umbral a partir del cual se considera la existencia del contorno. La imagen de contornos *Ic(x,y)* se puede formar a partir de:

$$I_c(x,y) = \begin{cases} 1 & ,\text{si } |Ig_1(x,y)| + |Ig_2(x,y)| > umbral \\ 0 & ,\text{en otro caso} \end{cases} \tag{2.53}$$

De esta manera todos los píxeles que presentan un gradiente mayor al umbral son declarados contornos.

La Figura 2.16 muestra un ejemplo de la detección de contornos, en (a) se encuentra la imagen de Elaine y en (b) la imagen de contornos de la misma.

2.4. Estado del Arte

En [10] se describe un sistema de procesamiento de imágenes flexible utilizando un FPGA por ser un medio apropiado para el desarrollo de operaciones de procesamiento de imágenes a nivel-bit en tiempo real usando el concepto de nivel-bit. Propone la arquitectura flexible PIPS (*Programmable Image Processing System*) para la integración de este hardware programable y DSP, que maneja operaciones aritméticas tan utilizadas en el procesamiento de imágenes, y lleva a cabo la implementación de un filtro de mediana 1D.

En [12], Joselyn Clouter presenta la implementación de un procesador para el procesamiento de imágenes y redes neuronales, llamado VIP. Este multiprocesador de tipo SIMD (*Single Instruction, Multiple Data*) es construido a partir de un FPGA, que unido con una topología torus 2-D, es apto para el procesamiento de imágenes, reconocimiento de patrones y la implementación de redes neuronales. Este sistema es programado en un nivel lógico bajo, permitiendo algoritmos con hardware óptimo dedicado.

IPXS es un entorno educacional para el diseño y programación de aplicaciones en un FPGA. Su principal objetivo es el de facilitar el trabajo de los estudiantes en la implementación de algoritmos para el procesamiento de imágenes. Consiste de dos partes: las herramientas IPXStools y el controlador de memoria (Memdrvs) [5]. La primera es una aplicación de Windows que permite desarrollar procesamiento de imágenes usando una tarjeta XS40, las imágenes que procesa son mayores a 8 bits/píxel de tipo BMP y PGM, lee y escribe imágenes a la XS40. La segunda son controladores de memoria hardware, implementa un protocolo de comunicación entre la PC-host y la tarjeta XS40 que consume de 5 % al 15 % de los recursos del FPGA.

Anthony Edward propone en [15] la implementación de algoritmos de procesamiento de imágenes sobre un FPGA. Los algoritmos propuestos son: filtro basado en una plantilla, operadores morfológicos y convolución. La implementación de los algoritmos es hecha sobre

FPGAs, Altera FLEX 10k100 y Xilinx Virtex XCV300BG352, y en Matlab para hacer una comparación de los resultados obtenidos. La integración de los algoritmos se lleva a cabo en ISIS (*Institute for Software Integrated Systems*), que es un sistema configurable basado en FPGAs de la compañía Altera. Cada algoritmos es integrado en un entorno de modelado, ACS (*Adaptive Computing System*), con la ayuda de GME (*Graphical Model Editor*), herramienta que sintetiza archivos de un FPGA, de un DSP, o la combinación de ambos para la sintetización e implementación de sistemas de mayor velocidad.

En [41] se presenta una arquitectura SIMD para la segmentación de imágenes implementada sobre la tecnología FPGA. Presenta unidades de procesamiento paralelo con pipeline interno y usa el operador gradiente Sobel para la detección de bordes en la imagen. Además, es aplicado en el tratamiento de imágenes de tarjetas electrónicas con dispositivos de montaje superficial. Esta arquitectura hace uso de la capacidad de los FPGA de realizar procesamiento paralelo con el fin de reducir los tiempos necesarios en las máquinas de estados. Este sistema es capaz de segmentar 43 imágenes de 640×480 píxeles en un segundo con una frecuencia de reloj de 40MHz.

Uzun I. S. propone en [48] la implementación de la FFT para señales en tiempo real y procesamiento de imágenes sobre un FPGA. Tomando en cuenta un alto rendimiento y facilidad de desarrollo, en este trabajo se muestra el diseño y realización de un marco de trabajo a alto nivel de la implementación de 1D y 2D FFTs para aplicaciones de tiempo real. Se implementan varios tipos de algoritmos de la FFT como son radix-2, radix-4, split-radix y la Transformada Rápida de Hartley (FHT, *Fast Hartley Transform*) en un marco de trabajo común con resultados de las implementaciones paralelas de velocidad elevada.

En [3] se expone un algoritmo para el reconocimiento de imágenes y/o patrones implementado sobre un Dispositivo Lógico Programable, FPGA, con el objetivo de obtener una arquitectura de Integración en Escala Extendida (VLSI, *Very Large Scale Integration*) para reconocer imágenes en la visión artificial de robots móviles. Dicho algoritmo hace uso de la DWT 2D utilizada para reducir la información de la imagen a procesar con el objetivo de optimizar los recursos de hardware, y emplea Memorias Asociativas Morfológicas (MAM, *Morphological Associative Memories*) para el reconocimiento de las imágenes o patrones.

En [39] se propone a STREAM, que es un procesador basado en un FPGA, este procesador es usado para el tratamiento de secuencias de imágenes, es una aplicación a la visión estéreo de tipo densa. Basa su funcionamiento en el apareamiento de primitivas destinadas al procesamiento de un par de imágenes estereoscópicas que requieren una importante potencia de cálculo, y que sean aplicados en robótica móvil por su respuesta en tiempo real. Provee un mapa denso de disparidad en tiempo real basado en el método de correlación.

Chin-Fa Hsieh *et al.* proponen en [22], la implementación de la DWT 1-D sobre una arquitectura VLSI basada en el *lifting* o filtrado FIR. Esta arquitectura es programada en Verilog HDL, implementada sobre los recursos de un FPGA y evaluada por la plataforma Quartus-II, que es una arquitectura de tiempo real, compuesta por la imagen proveniente de sensores CMOS, el FPGA y un panel TFT-LCD.

En [29] Medany presenta una arquitectura de hardware para un laboratorio remoto basado en tarjetas de desarrollo de circuitos electrónicos digitales que tienen como parte central un FPGA. Este laboratorio remoto es empleado para impartir cursos de diseño digital, en donde el estudiante puede accesar al FPGA mediante internet, ocupando la conexión remota de Windows XP o una página web. El FPGA es configurado a través de un puerto USB de la PC con el archivo de tipo *bit* recibido de la conexión; Xilinx ISE Foundation y el lenguaje VHDL han sido usados para la entrada del diseño. Este laboratorio está formado por 20 PCs y 20 tarjetas de desarrollo Spartan 3E de la compañía Xilinx.

3. MÉTODOS EMPLEADOS

El presente capítulo incluye una descripción de los Circuitos Digitales Configurables (CDCs), de sus arquitecturas y las ventajas de su uso respecto a dispositivos de tecnología ASIC (*Application Specific Integrated Circuit*). Además, se realiza una introducción al lenguaje descriptor de hardware VHDL, cuestiones básicas que ayudarán a diseñar sistemas digitales usando la metodología descendente (Top-Down) y se presenta la descripción de las herramientas EDA fundamentales en el diseño y modelado de circuitos digitales.

3.1. Circuitos digitales configurables

Los circuitos integrados digitales monolíticos son aquellos sistemas electrónicos que poseen todos sus componentes pasivos, dispositivos electrónicos y conexiones entre ellos, implementados en un solo bloque semiconductor. De la clasificación de los circuitos monolíticos, de acuerdo a la cantidad de componentes que los forman, surgen los circuitos VLSI, que actualmente exceden los 10 millones de compuertas lógicas. Los CDCs forman parte de la familia de dispositivos VLSI.

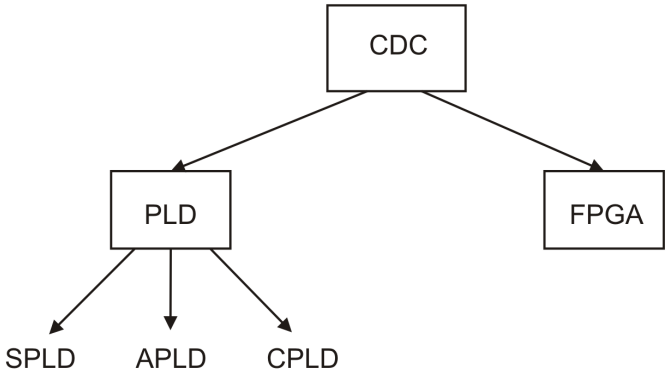


Figura 3.1: Clasificación de los CDCs en función de su arquitectura.

La configurabilidad es un concepto asociado a los sistemas digitales, en donde cierta parte de los elementos que lo componen suele ser utilizada para modificar la función del sistema. Esta modificación se lleva a cabo mediante la programación de un conjunto de variables binarias independientes o asociadas; a este proceso de programación se le conoce como "configuración". En los CDCs la configuración se realiza a través de ciertas terminales externas, que por medio de recursos lógicos, actúan sobre los dispositivos electrónicos programables que los hacen conducir o poner en estado de corte [28].

Como ventajas generales de los CDCs se pueden mencionar: costos de inicio bajos, riesgo financiero bajo, el usuario final es quien programa el dispositivo, se realizan cambios al diseño de manera fácil y las ganancias de manufactura son rápidas debido a que el producto final está disponible en corto tiempo.

Los CDCs pueden ser clasificados de acuerdo a diversos aspectos que les caracterizan: tecnología de reconfiguración, nivel de integración y arquitectura interna.

3.1.1. Clasificación de los CDCs en función de su arquitectura

La forma más comun de clasificar a un CDC es de acuerdo al tipo de organización que guarda su arquitectura interna. La Figura 3.1 muestra como son clasificados los CDCs en función de su arquitectura interna.

3.1.1.1. Dispositivos Lógicos Programables Simples (SPLDs)

El primer dispositivo programable por el usuario que surgió fue la PROM (*Programmable Read Only Memory*), aunque la función principal de este dispositivo es contener el programa que será ejecutado por un procesador, puede ser usado para la implementación de funciones lógicas combinacionales, donde el bus de direcciones de la memoria sirve como entradas lógicas y su el bus de datos como salidas. Una de sus desventajas es que no son aptas para aplicaciones de gran velocidad.

El primer diseño exclusivo para la implementación de circuitos lógicos fue el Arreglo Lógico Programable (PLA, *Programmable Logic Array*), el cual consistía de dos niveles de compuertas lógicas: el primer nivel compuesto por un arreglo de compuertas AND programable; cuyas salidas se expresan como el producto de las entradas, y el segundo nivel formado por un arreglo de compuertas OR programable, que tienen por función obtener la suma de productos del nivel anterior. Las desventajas encontradas por Philips, quien lo introdujo al mercado en la década de los 70's, eran un costo de manufactura y tiempo de retardo de las señales elevados. Debido a esto, Micro Devices desarrolló la PAL (*Programmable Array Logic*), formada por un conjunto de compuertas AND programables y un número fijo de suma

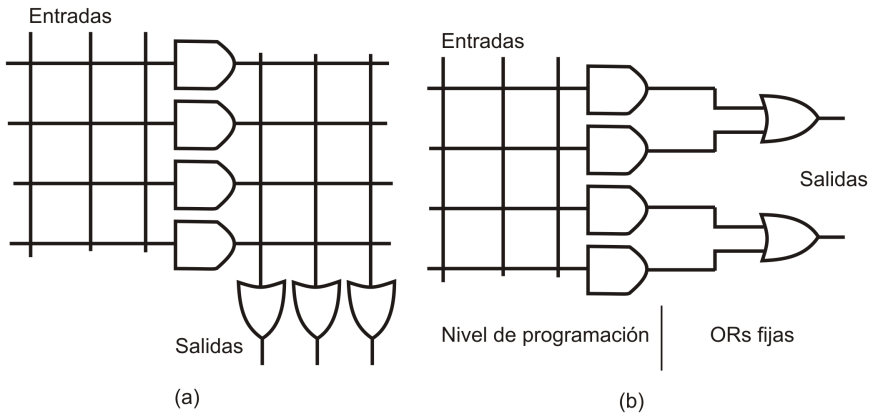


Figura 3.2: (a) Arquitectura PLA. (b) Arquitectura PAL.

de productos (OR). De esta manera surgió un rango de productos PAL con distintos números de entradas, salidas y tamaños de las compuertas OR. Contaban con otros dispositivos en su interior, como multiplexores, ORs exclusivas y latches en la entrada y en la salida. Además contenían varios flip-flops; ahora estos dispositivos permitían construir cierto número de funciones secuenciales cubriendo así la necesidad de implementar máquinas de estados. La PAL remplazó a la lógica estándar en muchos diseños por la rapidez de estos circuitos.

3.1.1.1.1. Arquitectura PLA

La arquitectura PLA se muestra en la Figura 3.2 (a). Posee las siguientes características:

- Programación de dos niveles.
- Cualquier combinación de ANDs/ORs.
- Compartimiento de términos AND a través de múltiples OR.
- Una mayor densidad de lógica disponible al usuario.
- Número de fusibles elevado.

3.1.1.1.2. Arquitectura PAL

La Figura 3.2 (b) muestra la arquitectura PAL. Posee las siguientes caracterísiticas:

- Sólo un nivel de programación AND.
- Nivel OR fijo.
- Combinación finita de ANDs/ORs.
- Densidad lógica media disponible al usuario.
- Número de fusibles bajo.

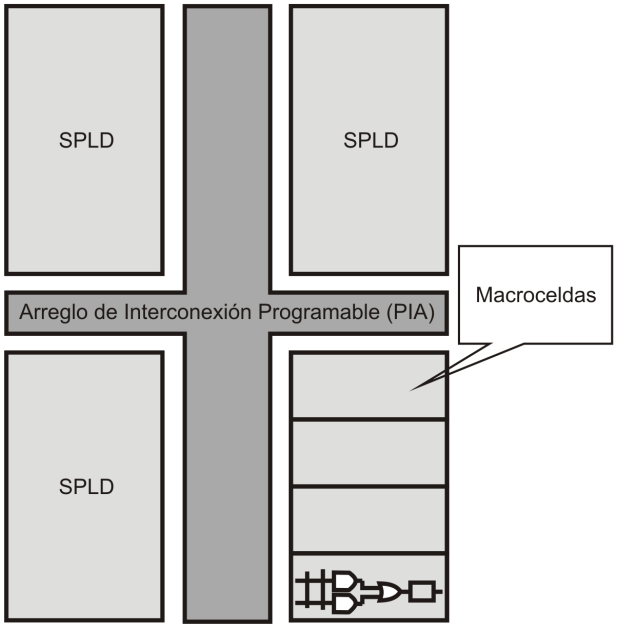


Figura 3.3: CPLD

3.1.1.2. Dispositivos Lógicos Programables Complejos (CPLDs)

Con el crecimiento de la tecnología, fue posible crear nuevos dispositivos con capacidades más elevadas que los SPLDs. Incrementando la capacidad de la matriz de compuertas AND programables no ayudaría en mucho, porque también se incrementaría el número de entradas. La solución fue integrar varios SPLDs dentro de un mismo chip que fueran vistos como bloques lógicos que pudieran ser interconectados por rutas específicas, surgiendo así los CPLDs. Actualmente las capacidades de un CPLD se comparan con las capacidades de 50 o más SPLDs.

3.1.1.2.1. Arquitectura CPLD

Los CPLDs extienden el concepto de un PLD a un mayor nivel de integración ya que permite implementar sistemas más eficaces, al utilizar menor espacio, mejorar la fiabilidad del diseño, y reducir costos. Un CPLD se forma con múltiples bloques lógicos, cada uno similar a un SPLD. Los bloques lógicos se comunican entre sí utilizando un arreglo de interconexión programable, lo cual hace más eficiente el uso del silicio, conduciendo a una mejor eficiencia a menor costo. Existen variaciones en las arquitecturas de los CPLDs de acuerdo al fabricante, pero en general, este dispositivo está formado por tres bloques principales: bloques lógicos, bloques de entrada/salida y un arreglo de interconexión programable, Figura 3.3.

Bloque lógico. Un bloque lógico es similar a un PLD, cada uno posee un bloque de compuertas AND y OR en forma de suma de productos, una configuración para la distribución de estas sumas de productos, y macroceldas. El tamaño del bloque lógico es una medida de la capacidad del CPLD, ya que de esto depende el tamaño de la función booleana que pueda ser implementada dentro del bloque. Los bloques lógicos usualmente tienen de 4 a 20 macroceldas. Las macroceldas de un CPLD son similares a las de un PLD. Estas también están provistas con registros, control de polaridad, y buffers para salidas en alta impedancia. Por lo general un CPLD tiene macroceldas de entrada/salida, macroceldas de entrada y macroceldas internas u ocultas (*buried macrocells*).

Arreglo de interconexión programable. El arreglo de interconexión programable permite unir los pines de entrada/salida a las entradas de un bloque lógico, o las salidas de un bloque lógico a las entradas de otro bloque lógico o inclusive a las entradas del mismo. La mayoría de los CPLDs usan una de dos configuraciones para esta matriz: interconexión mediante bloques o interconexión mediante multiplexores. El primero se basa en una matriz de filas y columnas con una celda programable de conexión en cada intersección. Al igual que en las GAL esta celda puede ser activada para conectar/desconectar la correspondiente fila y columna. Esta configuración permite una interconexión total entre las entradas y salidas del dispositivo o bloques lógicos. Sin embargo, estas ventajas provocan que disminuya el rendimiento del dispositivo, además de aumentar el consumo de energía y el tamaño del componente. En la interconexión mediante multiplexores, existe un multiplexor por cada entrada al bloque lógico. Las vías de interconexión programables son conectadas a las entradas de un número de multiplexores por cada bloque lógico. Las líneas de selección de estos multiplexores son programadas para permitir que sea seleccionada únicamente una vía de la matriz de interconexión por cada multiplexor, la cual se propagará hacia el bloque lógico. Cabe mencionar que no todas las vías son conectadas a las entradas de cada multiplexor. La rutabilidad se incrementa usando multiplexores de mayor tamaño, permitiendo que cualquier combinación de señales de la matriz de interconexión pueda ser enlazada hacia cualquier bloque lógico. Sin embargo, el uso de grandes multiplexores incrementa el tamaño de dispositivo y reduce su eficiencia.

Bloques de entrada/salida. Son usados para conducir las señales hacia los pines del dispositivo CPLD con un voltaje y corriente apropiados. Usualmente un flip-flop es usado en las salidas, con el fin de que las señales que dependen del reloj, sean conectadas a los pines sin retardos excesivos. En ocasiones otros recursos lógicos son incluidos en estos bloques para agregarles más funcionalidad.

3.1.1.2.2. Ventajas del uso de CPLDs

Algunas de las ventajas que ofrece el diseñar sistemas digitales utilizando CPLDs, con respecto a sus antecesores, son las siguientes:

- *Facilidad de diseño.* Los CPLDs ofrecen una manera sencilla de implementar el diseño. Cuando el diseño ha sido descrito mediante esquemático o algún HDL, el usuario puede usar las herramientas de desarrollo para optimizar, ajustar o simular dicho diseño. Estas herramientas generan un archivo que luego puede ser personalizado hacia un CPLD específico con la funcionalidad deseada. De esta manera se genera un prototipo de hardware instantáneo y listo para su depuración, si fuera necesario, mediante la herramienta de diseño.

- *Costo de desarrollo bajo.* Los CPLDs ofrecen un costo de desarrollo bajo, debido a que son fáciles de reprogramar los diseñadores realizan cambios en el diseño sin costo extra alguno. Esto permite optimizar sus diseños y continuar agregando nuevas características al mismo. Además, las herramientas de diseño son económicas y, en el caso de Xilinx, algunas son gratuitas.

- *Más ingresos en los productos.* Como los CPLDs tienen un ciclo de desarrollo corto, la implementación de productos con base en ellos es más rápida, repercutiendo en una mayor generación de ingresos económicos.

- *Reducen el área de la tarjeta.* Los CPLDs ofrecen un nivel de integración alto y están disponibles en pequeñas encapsulados de fábrica, proveyendo una solución perfecta para diseños de productos que requieren ser ajustados a espacios pequeños.

3.1.1.3. Arreglo de Compuertas Programables en Campo (FPGAs)

Los FPGAs fueron inventados en el año 1984 por Ross Freeman, co-fundador de la compañia Xilinx, y surgen como una evolución de los CPLDs. En el año de 1985, la compañía Xilinx introdujo al mercado este nuevo dispositivo como una idea nueva resultante de combinar el control de usuario y el tiempo de lanzamiento al mercado de los PLDs con la densidad y ventaja de los arreglos de compuertas, surgiendo así el FPGA. Un FPGA es un dispositivo semiconductor que contiene bloques lógicos cuya interconexión y funcionalidad se puede configurar. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip.

Se podría decir que los FPGAs, al igual que los CPLDs, entran dentro de la tecnología de los ASICs puesto que los FPGAs son circuitos integrados de aplicación específica. Por tanto, los FPGAs se utilizan en aplicaciones similares a los ASICs sin embargo son más lentos, tienen un mayor consumo de potencia y no pueden abarcar sistemas tan complejos como ellos. A pesar de esto, las FPGAs tienen las ventajas de ser reconfigurables (lo que añade una enorme flexibilidad al flujo de diseño), sus costes de desarrollo y adquisición son mucho menores para pequeñas cantidades de dispositivos y el tiempo de desarrollo es también menor. Además, generalmente los ASICs requieren de otras fases de procesamiento de semicoductores, los cuales los proveen de un mayor rendimiento pero también de altos costos en el diseño. Por otro lado, el diseñador de arreglos de compuertas tiene el control completo sobre la implementación de sus diseños sin la necesidad de tiempos perdidos en la fabricacion de circuitos integrados [30].

Tanto los CPLDs como los FPGAs contienen un gran número de elementos lógicos programables. Si se mide la densidad de los elementos lógicos programables en puertas lógicas equivalentes (número de puertas NAND equivalentes que podrían ser programadas en un dispositivo) se podría decir que en un CPLD se hallarían del orden de decenas de miles de puertas lógicas equivalentes y en una FPGA del orden de cientos de miles hasta millones de ellas.

Aparte de las diferencias en densidad entre ambos tipos de dispositivos, la diferencia fundamental entre los FPGAs y los CPLDs es su arquitectura. La arquitectura de los CPLDs es más rígida y consiste en una o más sumas de productos programables cuyos resultados van a parar a un número reducido de biestables síncronos. La arquitectura de los FPGAs, por otro lado, se basa en un gran número de pequeños bloques utilizados para reproducir sencillas operaciones lógicas, que cuentan a su vez con biestables síncronos. La enorme libertad disponible en la interconexion de dichos bloques confiere a los FPGAs una gran flexibilidad.

Otra diferencia importante entre FPGAs y CPLDs es que en la mayoría de los FPGAs se pueden encontrar funciones de alto nivel (como sumadores y multiplicadores) embebidas en la propia matriz de interconexiones, así como bloques de memoria y en algunos casos procesadores.

3.1.1.3.1. Arquitectura de un FPGA

Existe una gran variedad de FPGAs provistos por varias compañías como Xilinx, Altera, Atmel y Lattice. Cada fabricante provee a su FPGA con una arquitectura única. Un FPGA típico está formado por bloques lógicos configurables, bloques configurables de entrada/salida e interconexiones programables como se muestra en la Figura 3.4.

Bloques lógicos configurables. Los Bloques Lógicos Configurables (CLBs, *Configurable Logic Blocks*) son recursos lógicos que permiten al usuario realizar diferentes funciones; los CLBs están distribuidos en forma matricial en el dispositivo. En el caso donde estos recursos

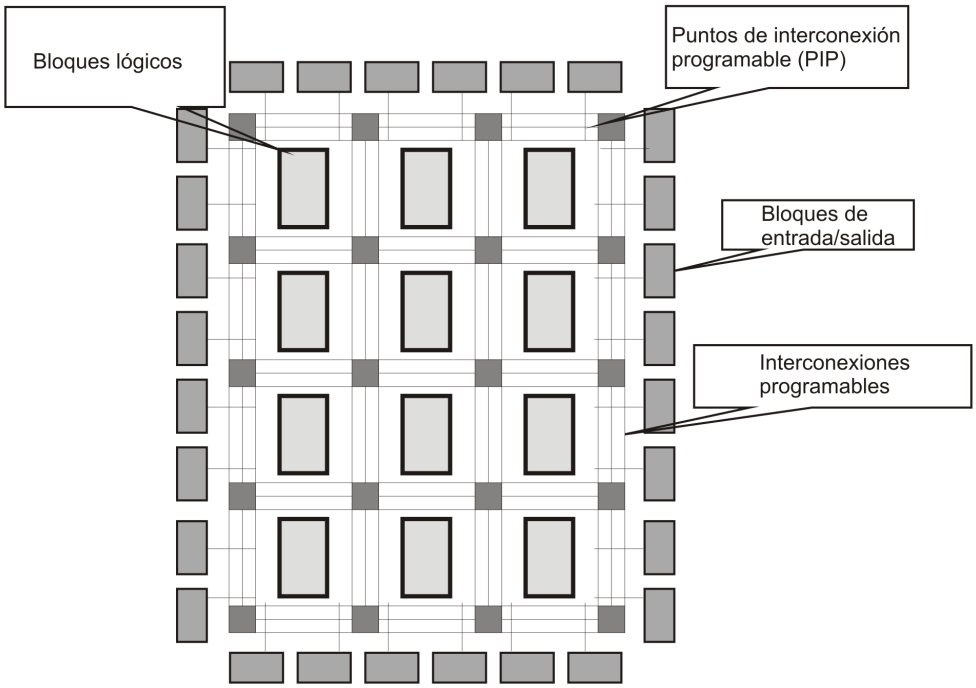


Figura 3.4: FPGA.

son de complejidad baja, es decir, las funciones lógicas que se pueden realizar en ellos son sencillas, y existe un número considerable de ellos, se dice que el FPGA es de granularidad fina. Cuando los recursos lógicos están formados por memorias de acceso aleatorio llamadas Tablas de Búsqueda (LUT, *Look-up Tables*), flip-flops para almacenamiento de elementos que dependen de la señal de reloj, multiplexores que permiten la selección, reset y puesta a uno lógico de elementos, se dice que el FPGA es de granularidad gruesa; en un FPGA de este tipo el número de CLBs que lo integran es reducido pero tienen la característica de poder implementar funciones de mayor complejidad.

Bloques configurables de entrada/salida. La matriz de bloques de CLBs está rodeada por un anillo de bloques de interfaz, denominados "bloques configurables de entrada/salida". Estos bloques están dedicados a proporcionar la interconectividad entre el FPGA y el exterior, es decir, controlan la entrada y salida de datos entre los pines de entrada/salida y la lógica interna. Cada bloque es bidireccional y soporta operaciones de tercer estado, para conseguir estas características un bloque está dotado de flip-flops, latches y buffers de tercer estado. Además, en ocasiones incluye resistores pull-up y/o pull-down en la salida. La polaridad de la señal de salida es programable.

Interconexiones programables. Están formados por: *recursos de interconexión*, *conjunto de líneas* y/o *interruptores programables* que permiten transmitir las señales entre los bloques lógicos internos y entre estos y los bloque de entrada/salida, y de *matriz de interconexión*, elementos lógicos que facilitan la comunicación entre los buses de comunicación (recursos de interconexión).

Circuitería de reloj Existe un tercer tipo de recursos exclusivos de conexión: las líneas dedicadas a la transmisión de las señales de reloj. Esto es así debido a que las señales de reloj tienen la característica especial de que han de estar conectadas a un gran número de bloques por lo que han de llegar a todos los rincones del FPGA en el menor tiempo posible. Estas

líneas de reloj global son diseñadas para obtener tiempos de propagación pequeños y similares. Estas señales de reloj son distribuidas mediante buffers de reloj especiales, conocidos como drivers de reloj, y están distribuidos alrededor del FPGA. Estos buffers se encuentran conectados al reloj principal y lo llevan hacia todas la líneas de reloj global con el fin de que puedan ser utilizadas por cada CLB.

3.1.1.3.2. Ventajas del uso de los FPGAs

La razón por la cual los FPGAs están siendo preferidos es porque ofrecen muchas de las ventajas de los circuitos ASICs, como son:

- Reducción en tamaño, peso y disipación de potencia.
- Mayor desempeño.
- Mejor seguridad contra las copias no autorizadas.
- Dispositivos y costos de inventarios reducidos.
- Costos de tarjetas de prueba reducidos.

y además poseen otras características que los circuitos ASICs no tienen, como son:

- Una reducción considerable en los tiempos de desarrollo (prototipado rápido) en un factor de 3 ó 4.
- Reconfigurabilidad del circuito aún en sistema (ISP).
- Costos bajos de ingeniería no-recurrente, resultando en diseños más económicos para soluciones que requieren de menos de 1000 unidades.

3.1.2. Clasificación de los CDCs por su tecnología de configuración

Los CDCs son fabricados utilizando diversas tecnologias en sus elementos de configuración, las más comunes son: SRAM, EPROM, E^2PROM y antifusible. El tipo de tecnología define si los dispositivos son reconfigurables o simplemente Configurables Una sola Vez (OTP, *One Time Programmable*).

Los CDCs basados en SRAM son la tecnología predominante, están basados en memorias CMOS y son reprogramables aun en sistema (ISP, *In System Programmable*), además requieren de un "*boot*" para su configuración. Los basados en memorias de sólo lectura programables eléctricamente (EPROM), usualmente son utilizados en modelos CMOS OTP, debido a que es necesaria la exposición a luz ultravioleta de dichas memorias para su reprogramación. Los modelos de CDCs basados en E^2PROM son ISP al igual que los basados en tecnología "*flash*".

La tecnología antifusible es usada en los CDCs OTP, que al contrario de los fusibles, las conexiones son hechas sin fundirse durante la programación. Los antifusibles realizan conexiones permanentes en el interior de los FPGAs y los CPLDs sin la necesidad de SPROMs u otros componentes para descargar el programa dentro del chip. Aquí las celdas lógicas son similares a los PLDs con compuertas dedicadas y flip-flops [35]. En la Tabla 3.1 se hace una comparación entre varios CDCs con diversas tecnologías de configuración.

Tecnología	SRAM	EPROM	E^2PROM	Flash	Antifusible
Reprogramable	✓	✓	✓	✓	-
ISP	✓	-	✓	✓	-
Volatil	✓	-	-	-	-
Protección de copia	-	✓	✓	✓	✓
Ejemplos	Xilinx XC4k	Altera MAX5K	AMD MACH	Xilinx XC9500	Actel ACT
	Altera Flex	Xilinx XC7K	Altera MAX9K		Cypress Ultra 37K

Tabla 3.1: Tecnología de los CDCs.

3.2. VHDL

Alrededor de 1981 el Departamento de Defensa de los Estados Unidos desarrolló un proyecto llamado VHSIC (*Very High Speed Integrated Circuit*) su objetivo era rentabilizar las inversiones en hardware haciendo más sencillo su mantenimiento. Se pretendía con ello resolver el problema de modificar el hardware diseñado en un proyecto para utilizarlo en otro, lo que no era posible hasta entonces porque no existía una herramienta adecuada que armonizase y normalizase dicha tarea. La solución propuesta fue un HDL, en un principio este lenguaje fue concebido con el propósito de especificar y documentar sistemas digitales mediante un lenguaje textual.

En 1983, IBM, Intermetrics y Texas Instruments empezaron a trabajar en el desarrollo de un lenguaje de diseño que permitiera la estandarización, facilitando con ello, el mantenimiento de los diseños y la depuración de los algoritmos, para ello el IEEE propuso su estándar en 1984. Tras varias versiones llevadas a cabo con la colaboración de la industria y de las universidades, que constituyeron a posteriori etapas intermedias en el desarrollo del lenguaje, el IEEE publicó en diciembre de 1987 el estándar IEEE std 1076-1987 que constituyó el punto firme de partida de lo que después de cinco años sería ratificado como VHDL, Lenguaje Descriptor de Hardware para Circuitos Integrados de muy alta Velocidad (VHSICHDL, *Very High Speed Integrated Circuit Hardware Description Language*).

VHDL se ha convertido en el lenguaje descriptor de hardware más popular; como una manera de expresar los conceptos de diseños digitales complejos en simulación y síntesis. Con VHDL es fácil describir diseños y sistemas digitales muy extensos con la ayuda de una gran variedad de herramientas de diseño disponibles, las cuales trasladan dicha descripción hacia elementos de hardware. VHDL es un lenguaje de programación que ha sido diseñado y optimizado para describir el comportamiento de sistemas digitales [36].

VHDL es la suma de características importantes como son: 1) un lenguaje de simulación de modelos, que puede ser utilizado para construir bloques de circuitos muy extensos usando esquemáticos o en modo texto, con el propósito de simularlos, 2) un lenguaje de entrada de diseño, lenguaje de programación de alto nivel que permite expresar conceptos de diseño como programas de computadora. Cuenta con síntesis automática de circuitos, simulación de diseños, un conjunto de variables de control y la representación de datos para un diseño estructurado, y como los sistemas de hardware, permite eventos concurrentes, 3) puede ser utilizado como un lenguaje *Netlist*, de esta manera es útil como una forma de comunicación de bajo nivel entre distintas herramientas-entorno de diseño basadas en computadoras, 4) es un lenguaje estándar, que garantiza que los diseños digitales implementados tengan portabilidad hacia generaciones de herramientas de diseño más recientes y 5) es tan simple o tan complejo como la aplicación lo requiera y el tipo de descripción elegida.

3.2.1. 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.
- Se encuentra basado en el estándar IEE Std 1076-1987 y IEEE Std 1076-1993 y así se minimizan los errores de comunicación y problemas de compatibilidad. Además, al no ser un lenguaje de propietario, cualquier usuario puede desarrollar un herramienta para VHDL y comercializarla.
- Permite el diseño Top-Down, se puede describir o modelar el comportamiento de los bloques de un circuito a un alto nivel.
- Con la modularidad que permite VHDL, se puede dividir o descomponer los diseños hardware y su descripción en unidades más pequeñas.

3.2.2. Niveles de abstracción en VHDL

VHDL soporta varios estilos para la descripción de diseños. Estos estilos de diseño son distintos de acuerdo al nivel de abstracción que manejan, y son: el estilo comportamental, el de flujo de datos y el estructural.

El estilo comportamental o algorítmico es el nivel de abstracción más elevado que soporta VHDL. Cuando se describe un circuito usando este nivel de abstracción, el circuito se describe 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 pueden 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 [40], [36].

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 comportamiento por medio de ecuaciones lógicas y sentencias de asignación. Es comúnmente llamada Transferencia Lógica de Registros (RTL, *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 especificados 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 descrición 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 de un circuito.

3.2.3. La dupla Entidad-Arquitectura

Toda descripción de diseños en VHDL requiere de una entidad y una o más arquitecturas. La declaración de la entidad define la interfaz del circuito digital que se está diseñando con

el mundo exterior. La declaración de una arquitectura complementa el diseño del sistema, la cual describe la estructura o el comportamiento de la entidad a la cual pertenece.

La entidad es la abstracción de un circuito, ya sea desde un complejo sistema electrónico hasta una simple compuerta. Sólo describe la forma externa del circuito, definiendo las entradas y salidas de éste. Es análoga a un símbolo en esquemático. La entidad sirve para relacionar el diseño con el mundo exterior, es decir, se analiza lo que se intenta modelar como una "caja negra", de la que sólo se conocen sus entradas, sus salidas y la disposición de las mismas. La arquitectura es el complemento de la entidad para describir completamente el funcionamiento de un circuito o sistema digital. La arquitectura representa la estructura interna del bloque declarado por la entidad y describe la forma en que la información en las entradas es procesada para obtener las correspondientes salidas, modelando de esta manera el funcionamiento del circuito diseñado.

3.3. Herramientas EDA

Con el advenimiento de los circuitos con tecnología VLSI, con más de 100,000 transistores en un circuito integrado digital monolítico, y con la complejidad de verificación de estos diseños, las herramientas asistidas por computadora comenzaron a surgir como una forma de apoyo en la verificación y diseño de circuitos de escala de integración muy elevada [33].

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 del Diseño Electrónico (EDA). 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

Las herramientas EDA software, también conocidas como herramientas de Diseño Asistida por Computadora (CAD) 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 específico.

En el diseño de sistemas o circuitos para su implementación en CDCs, las herramientas EDA resultan de vital importancia debido a varios factores:

- Automatizan el proceso de desarrollo de aplicaciones, repercutiendo en la disminución del tiempo de diseño, el aumento de la calidad del producto y la reducción de los tiempos de producción.
- Facilita a los ingenieros el desarrollo de circuitos
- Permiten implementar sistemas cada vez mas sofisticados.
- El uso de las herramientas EDA junto con los dispositivos lógicos programables, que pueden ser utilizados en diferentes aplicaciones e inclusive reprogramados, han cambiado el concepto de diseño de circuitos digitales.

En el caso de los SPLDs, las herramientas CAD incluyen las siguientes tareas: entrada de diseño inicial, la optimización de la lógica, ajuste del dispositivo, simulación del diseño y configuración. La entrada del diseño se realiza al crear un diagrama esquemático con una herramienta CAD gráfica o usando un sistema basado en texto que hace uso de un lenguaje descriptor de hardware simple, es la única herramienta manual, puesto que las demás son automáticas. Debido a que la lógica resultante de la entrada del diseño suele ocupar muchos recursos, se emplean herramientas de optimización de la entrada de diseño, y para finalizar, entran las herramientas que ajustan las ecuaciones resultantes en los recursos del SPLD. Por

otra parte, la simulación permite verificar el correcto funcionamiento del diseño y detectar errores, que son corregidos en la entrada del diseño.

Similar al proceso de diseño de los SPLDs pero con herramientas más sofisticadas, el proceso de diseño de circuitos en CPLDs, con la ayuda de herramientas CAD, soporta diseños más extensos, y para la entrada del diseño suele ser necesaria una combinación de captura en modo texto, con lenguajes de descripción de hardware, y herramientas de captura esquemática con símbolos. Para el caso de los FPGAs, con el incremento de la complejidad, se suman otras herramientas, principalmente en el ajuste del dispositivo, que se encuentran después de la optimización y antes de la simulación, requiere de por lo menos tres pasos que son: tecnología de mapeo de compuertas lógicas hacia bloques lógicos del FPGA, una colocación (*placement*) que elige qué bloque lógico se debe usar para implementar cierta función lógica, y un ruteador (*router*) que asigna los segmentos de conexión para interconectar los bloques lógicos. A continuación se describen de forma general las fases del proceso de diseño en FPGAs haciendo uso de herramientas CAD:

- *Captura del diseño.* Permite al usuario modelar sus diseños en base a las siguientes entradas de diseño: diagrama esquemático (símbolos), lenguaje descriptor de hardware (HDL) en modo texto y editor de máquinas de estados.

- *Simulación comportamental.* Es usada después del modelado digital con el fin de verificar el correcto funcionamiento del diseño a un nivel de transferencia de registros, sin considerar los retardos de las señales en la implementación.

- *Síntesis lógica.* Al realizar la compilación de la captura del diseño se produce un archivo Netlist, este archivo registra la descripción de las celdas lógicas y sus interconexiones que conforman el diseño. En este paso se adapta el modelado del diseño anterior a un hardware en concreto, ya sea una FPGA o un ASIC.

- *Implementación.* Está formada por una serie de fase que son: 1) mapeo o planeamiento de la superficie (*floorplanning*), donde se mapea el archivo Netlist generado en la síntesis lógica, buscando distribuir los recursos lógicos programables y sus interconexiones en una forma óptima sobre cierta superficie del dispositivo, minimizando el espacio físico y los retardos de propagación de las señales en una primera aproximación, 2) colocación (*Placement*), en esta fase la herramienta CAD toma decisiones sobre la colocación de las primitivas sobre los bloques lógicos; consideran la mejor ubicación para cada bloque lógico, así como las redundancias en el diseño y una segunda aproximación para eliminar los retardos críticos, 3) trazado (*Routting*), se encarga de proponer la mejor opción entre rutas cortas o largas para la interconexión física de los bloques lógicos dentro del FPGA y 4) la traducción (*translate*), que es la extracción de manera automática de características del circuito, como por ejemplo la resistencia y la capacitancia eléctica entre interconexiones.

- *Simulación temporal.* Después de las fases anteriores, la configuración física que ha adquirido el FPGA puede simularse. También se le denomina *simulación física* debido a su proximidad al comportamiento real del diseño sobre el dispositivo. A diferencia de la simulación comportamental, aquí es posible simular retardos de propagación en las señales. Es importante calcular el retardo crítico o aquel retardo de propagación de una señal que sea el mayor de todos. Esto mediante la suma de los retardos de cada bloque lógico más los retardos de la interconexiones que participan en la ruta, desde un puerto de entrada hasta otro de salida.

- *Programación del dispositivo.* Consiste de un conjunto de módulos que programan al dispositivo deseado. Entre ellos se encuentran el generador de la información necesaria para la configuración del dispositivo y el módulo que permite la interfaz con la herramienta EDA-hardware, en este caso un FPGA.

3.4. Metodologías de modelado

Existen varias metodologías que pueden ser empleadas cuando se requieren diseñar circuitos o sistemas digitales. Estas metodologías únicamente se utilizan en el diseño conceptual del sistema digital, y son independientes de la herramienta EDA y el HDL empleados. De las metodologías más empleadas se encuentran las metodologías Ascendente (*Button-Up*) y Descendente (*Top-Down*).

3.4.1. Metodología de diseño Ascendente

En la metodología ascendente el modelado del circuito o sistema, que se pretende realizar, comienza con la descripción de los componentes básicos, utilizando primitivas de la herramienta, que más tarde se agrupan en diversos módulos, y estos a su vez se agrupan para formar módulos de mayor complejidad y asi sucesivamente hasta formar un sólo bloque que represente el sistema completo. No implica una estructuración jerárquica de los elementos que componen al sistema, esta estructuración se realiza tras la descripción del circuito, y por tanto, no resulta necesaria.

Las *primitivas* del lenguaje empleado en el modelado del diseño son los componentes de más bajo nivel que permiten realizar la descripción del sistema. Por lo común, estas primitivas son componentes que se encuentran en bibliotecas, tales como chips, resistencias, condensadores y otras unidades funcionales.

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 diseño grandes, con la unión de más de mil componentes, puede generar diseños que no funcionen adecuadamente; además, se hace complejo el análisis del circuito, con dificultades en la detección errores y anomalías de funcionamiento.

3.4.2. 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. Esta metodología inicia visualizando al sistema a diseñar con un nivel de abstracción alto, para luego dividirlo en módulos jerárquicamente inferiores, cada uno de estos módulos puede ser también dividido; el nivel de detalle del módulo de menor jerarquía depende directamente de las primitivas proporcionadas por la herramienta utilizada.

La mejora de las herramientas de diseño electrónico han permitido que actualmente se cuenten con herramientas que realizan de forma automática la metodología de diseño descendente, herramientas de síntesis que permiten la implementación automática de un circuito a partir de una idea abstracta de diseñador.

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 de compuertas, 2) incrementa la reutilización del diseño, se utilizan tecnologías genéricas en el proceso de diseño; la tecnología a utilizar se fija en las fases posteriores al diseño. Esto permite reutilizar los diseños al cambiar la tecnología de implementación, y 3) una detección de errores rápida,

con más tiempo dedicado a la definición y al diseño.

4. DESARROLLO DEL SISTEMA

Una vez analizados los aspectos generales, los cuales contemplan las bases necesarias para cumplir con el objetivo de este trabajo, en este capítulo se procede a la descripción del diseño e implementación de un Sistema de Procesamiento y Análisis Digital de Imágenes sobre un FPGA, el cual será utilizado como una herramienta que brindará una base sólida en el diseño e implementación de prototipos autónomos que incluyan algoritmos de procesamiento y/o análisis digital de imágenes. La función principal de esta herramienta es agilizar el diseño, modelado y evaluación de un algoritmo en un FPGA y permitir visualizar los resultados antes de implementarlos en un sistema final. Esta herramienta tendrá como principales campos de operación los ámbitos académico y de investigación relacionados con el procesamiento digital de imágenes.

4.1. Introducción

En este capítulo se describe el diseño e implementación de una herramienta que permite modelar y evaluar el comportamiento de algoritmos de procesamiento y análisis digital de

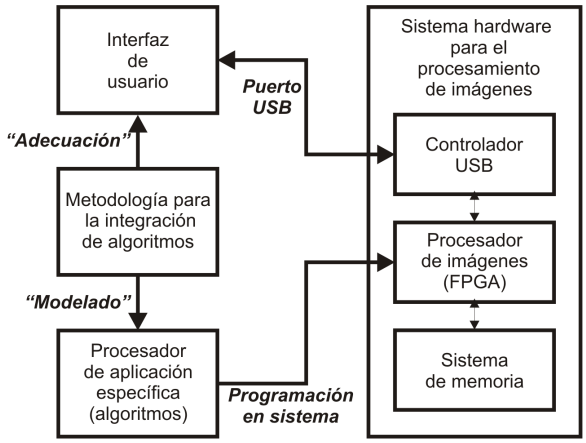


Figura 4.1: Diagrama de bloques del sistema propuesto.

imágenes cuando son aplicados sobre un CDC. El proceso de diseño e implementación de esta herramienta es dividido en cuatro fases: 1) modelado de un procesador de aplicación específica, orientado al procesamiento y análisis de imágenes, en un FPGA, 2) con base en el procesador modelado en la fase anterior, en esta fase se diseña e implementa un sistema hardware para el procesamiento de imágenes utilizando la metodología de sistemas empotrados, 3) el diseño de una interfaz de usuario, y 4) la propuesta de una metodología para la integración de algoritmos de procesamiento y análisis de imágenes a la herramienta obtenida. En la Figura 4.1 se muestra un diagrama de bloques que incluye a cada una de estas fases y la forma en que interactuarán entre si.

- *Fase 1: Modelado de un procesador de aplicación específica.* En esta fase, siguiendo una metodología descendente, se lleva acabo el diseño y modelado de la estructura básica de un procesador de aplicación específica denominado **procesador de imágenes**, este procesador estará integrado por módulos que describen o facilitan la descripción de algoritmos utilizados en el procesamiento de imágenes y módulos que le permitirán realizar una interfaz con periférios externos; además, a este procesador fácilmente y siguiendo la metodología propuesta en la Fase 4 se le podrán integrar nuevos módulos que modelarán el comportamiento de nuevos algoritmos de procesamiento y/o análisis de imágenes.

- *Fase 2: Sistema hardware para el procesamiento de imágenes.* En esta fase, siguiendo la metodología de un sistema empotrado, se implementa un sistema con base en el procesador diseñado en la fase previa. Los periféricos que integrarán al sistema serán un controlador USB, utilizado para implementar una interfaz de alta velocidad con una PC, y un subsistema de memoria formado por 512×16 localidades y que servirá como buffer de almacenamiento de la imagen a procesar y de los resultados de dicho procesamiento. Además la arquitectura del **procesador de imágenes** y la estructura del **sistema hardware para el procesamiento de imágenes** permiten fácilmente añadir nuevos periféricos al sistema.

- *Fase 3: Interfaz de usuario.* La **interfaz de usuario** permite la visualización de las imágenes a procesar o analizar, indicar cual o cuales algoritmos, contenidos en el **procesador de imágenes**, serán aplicados sobre la imagen, los parámetros específicos requeridos por cada algoritmo y muestra los resultados obtenidos de los mismos. La **interfaz de usuario** es programada en una PC utilizando la herramienta C++ Builder 6.0 y debe contemplar la implementación de una interfaz de alta velocidad

con el **sistema hardware para el procesamiento de imágenes**. Considerando que las Fases 1 y 2 deben establecer una comunicación para cumplir los puntos mencionados, estas fases son complementadas con un protocolo de comunicación, formado por 5 tramas, que rige el intercambio de información entre el **sistema hardware para el procesamiento de imágenes** y la **interfaz de usuario**.

- *Fase 4: Metodología para la integración de algoritmos.* Para obtener una herramienta que optimice las funciones para las que fue creada, es impresindible contar con una metodología que facilite el modelado e integración de nuevos algoritmos de procesamiento y/o análisis al **procesador de imágenes** y las funciones necesarias para su evaluación. Por esta razón, en esta fase se propone una **metodología para la integración de algoritmos** que establece la secuencia de procedimientos necesarios para especificar, integrar y evaluar el comportamiento de un nuevo algoritmo en la herramienta diseñada.

4.2. Modelado de un procesador de aplicación específica

Con la finalidad de obtener información objetiva de una escena captada por un sistema de adquisición, se puede hacer un procesador en la implementación de algoritmos de procesamiento y análisis digital de imágenes. El procesador empleado puede ser el utilizado por una computadora personal, un microcontrolador, un DSP o un procesador de aplicación específica modelado en un CDC.

Un gran número de algoritmos de procesamiento de imágenes es cubierto con programas de software en una PC, los cuales mejoran, restauran, comprimen o analizan imágenes en formato digital; éstas hacen uso del procesador que incluyen, el cual posee velocidades de procesamiento elevadas. Sin embargo, los tiempos de uso de estos procesadores son compartidos con otras aplicaciones de software, por lo que estos programas de procesamiento de imágenes no son de tiempo real.

Los DSPs por su parte, cuentan con operaciones de multiplicación, división, suma, multitareas, etc. y en general, operaciones en punto flotante, que se realizan en un sólo ciclo de reloj gracias a su tecnología RISC (*Reduced Instruction Set Computer*). Las características de los DSPs los hacen aptos en los algoritmos de procesamiento de imágenes, en los cuales es común encontrar multiplicaciones, sumatorias u otras operaciones que demanden muchos recursos de hardware y velocidades de procesamiento elevadas.

Una opción que ha despertado gran interés en el área de procesamiento de imágenes es el modelado de procesadores de aplicación específica en un CDC. Un procesador de aplicación específica es aquel procesador que es diseñado considerando la aplicación o el área de la ingeniería en donde será utilizado. Esta opción presenta las siguientes ventajas: 1. Optimización de recursos, permite modelar únicamente las funciones necesarias en la aplicación. 2. Modelado de arquitecturas paralelas, repercutiendo en altas velocidades de procesamiento. 3. Uso de HDLs estandarizados, lo cual genera diseños portables entre diversas tecnologías, además, la portabilidad se extiende hacia otras herramientas CAD compatibles.

Los procesadores de aplicación específica basados en la tecnología ASIC son diseñados a la medida o semi-medida. El término "a la medida" indica la cantidad de requerimientos que el usuario especifica en el diseño del procesador, como la funcionalidad que cubren, tamaño del dispositivo, basados en algún semiconductor, tiempos de propagación críticos de las señales dentro del dispositivo y la velocidad alcanzada, por mencionar algunos; el diseñador cumple con estos requisitos llevando el diseño a un nivel de capas de semiconductor, transistores, con ayuda de herramientas CAD especializadas.

Cuando el procesamiento y/o análisis de imágenes es aplicado en sistemas autónomos, tales como la visión robótica, el uso de procesadores de PCs queda descartado. De las opciones

restantes, la tendencia de usar DSPs o CDCs se inclina hacia estos últimos, debido a que se ha demostrado una mayor eficiencia de estos, específicamente en los FPGAs [30].

En esta sección se describe el modelado en un FPGA, de la estructura básica de un procesador enfocado al procesamiento y/o análisis de imágenes, al cual se denominará **procesador de imágenes** y representa al elemento central del **sistema hardware para el procesamiento de imágenes**, a desarrollarse en la Fase 2; la estructura general de este procesador es mostrada en el diagrama de bloques de la Figura 1.2, donde se puede apreciar que está conformado por varios subsistemas que se describen a continuación:

- *Módulo central de proceso.* El módulo central de proceso se encarga de administrar los recursos que integran al **procesador de imágenes**: el módulo de control USB, el módulo de control del sistema de memoria, y los algoritimos de procesamiento y análisis de imágenes; además, rige las diferentes operaciones que involucran a estos módulos y las realizadas entre el **procesador de imágenes** y la **interfaz de usuario**, desarrollada en la Fase 3, algunas de estas operaciones son la decodificación de las tramas, la recepción de la imagen a procesar, el envío de los resultados del procesamiento, etc.

- *Módulo de control USB.* El **sistema hardware para el procesamiento de imágenes** debe ser capaz de establecer una comunicación de alta velocidad con la **interfaz de usuario** utilizada en el intercambio de información entre ambos; para tal propósito se ha elegido un controlador hardware de USB, con base en las especificaciones del controlador, el módulo de control de USB se encarga de implementar el protocolo de comunicaciones entre éste y el **procesador de imágenes**.

- *Módulo de control del sistema de memoria.* Las imágenes a procesar pueden ser de un tamaño variado, pudiendo ser de hasta 512×512 píxeles (8 bits/píxel), debido a esto es necesario implementar un sistema de memoria que sirva como buffer de almacenamiento temporal de la imagen a procesar y de los resultados del procesamiento; el módulo de control del sistema de memoria implementa el protocolo para que el **procesador de imágenes** pueda accesar a este sistema de memoria.

- *Módulos de algoritmos de procesamiento y análisis de imágenes.* El **procesador de imágenes** incluye algunos algoritmos de procesamiento y/o análisis de imágenes, además, algoritmos nuevos pueden ser incluidos en la estructura del procesador; cada uno de estos algoritmos debe ser definido dentro del procesador como un nuevo subsistema o módulo.

Antes de iniciar con la descripción del **procesador de imágenes** es necesario hacer mención de algunos aspectos de relevante importancia en el modelado del mismo, estos son, las herramientas EDA elegidas y la metodología de diseño a utilizar.

Como se mencionó en la Sección 3.3, las herramientas EDA tienen por 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 **procesador de imágenes** se ha elegido como herramienta de hardware a la tarjeta de desarrollo Nexys-2 de la compañia *Digilent* y como herramienta EDA-CAD al programa ISE Fountation, en su versión 8.2i, de la compañía Xilinx.

La tarjeta Nexys-2 es una plataforma de diseño de sistemas digitales que tiene como elemento principal al FPGA 3E-500 FG320 de la familia Spartan de la compañía Xilinx. Las características más relevantes de esta plataforma de desarrollo son las siguientes:

- El FPGA 3E-500 tiene una capacidad de 500,000 compuertas (*System Gates*).

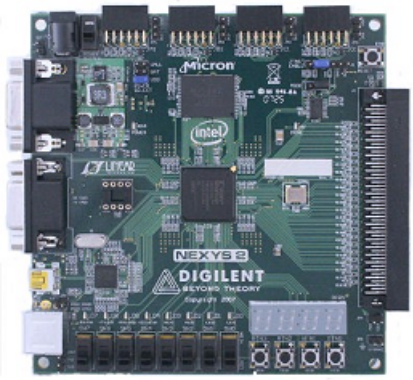


Figura 4.2: Tarjeta de desarrollo de sistemas digitales Nexys-2.

- Utiliza el puerto USB, ver. 2.0, para configurar al FPGA y establecer una transferencia de datos de alta velocidad entre el FPGA y una PC.
- Compatible con la herramienta ISE/Webpack de Xilinx.
- Incluye 16 MBytes de PSDRAM de la compañía de Micron y 16 Mbytes de StrataFlash ROM de Intel.
- Plataforma flash no volátil para configuración del FPGA.
- Fuente de alimentación conmutada de alta eficiencia.
- Oscilador integrado de 50MHz y socket para un oscilador extra.
- Conectores de expansión con 60 entradas/salidas al FPGA.
- Periféricos elementales: 8 leds, 4 buttons, 8 interruptores tipo slide y 4 displays de 7 segmentos.

En la Figura 4.2 se muestra la tarjera de desarrollo Nexys-2. El Apéndice A incluye una explicación detallada de esta plataforma de desarrollo.

Por su parte, la herramienta ISE Foundation, creada por la compañía Xilinx para configurar sus CDCs, es un entorno de desarrollo integrado (IDE), cuyo objetivo es facilitar el proceso de diseño de un sistema digital mediante la automatización del tansito por las diferentes fases que integran dicho proceso (desde el modelado, pasando por la simulación y la síntesis lógica hasta llegar a la implementación del sistema en la arquitectura configurable elegida). La Figura 4.3 muestra el entorno de desarrollo de ISE. La herramienta ISE incluye diferentes formas de modelado de sistemas digitales, captura de esquemático, editor de diagramas de estado y editor de HDL. Esta última forma de modelado permite elegir entre las normas IEEE 1076 (VHDL)[44], [46] y la IEEE 1364 (Verilog) [45], [46]; para el modelado del **procesador de imágenes** se ha elegido el estandar VHDL.

Una vez elegidas las herramientas EDA, y considerando que se usará el lenguage VHDL, es necesario definir cuál metodología se seguirá en el diseño conceptual del procesador y qué tipo de modelado se empleará en la descripción del mismo.

En la sección 3.4 se describieron los diferentes tipos de metodologías utilizadas en el diseño de un sistema digital, ascendente y descendente, siendo esta última la elegida para llevar a cabo el diseño conceptual del **procesador de imágenes** debido a que en la actualidad es ampliamente utilizada en el diseño de arquitecturas VLSI o de una complejidad elevada.

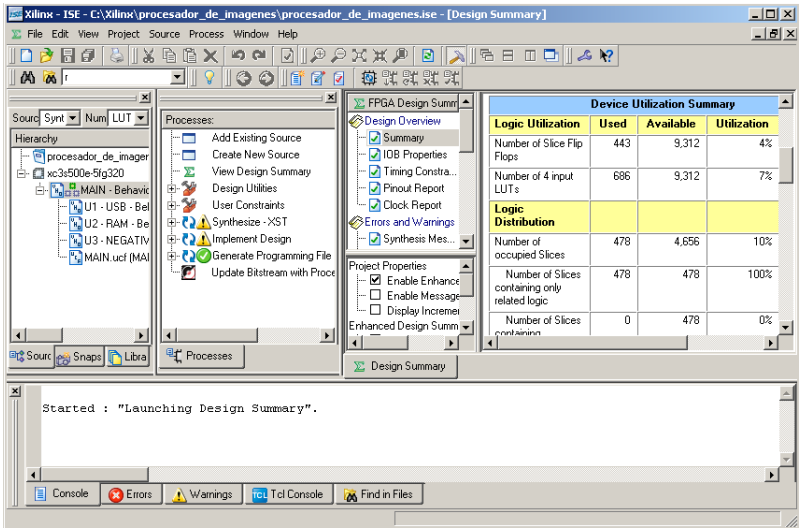


Figura 4.3: Herramienta ISE Foundations 8.2i.

Por otra parte, los distintos tipos de modelado que permite VHDL fueron descritos en la Sección 3.2.2, estos tipo son estructural, flujo de datos y comportamental; la elección del tipo de modelado depende del nivel de abstracción que se desea manejar. Para el modelado del **procesador de imágenes** se ha elegido el estilo comportamental o algorítmico para generar los diferentes módulos o subsistemas y del estilo estructual para unirlos.

Por último, es importante mencionar que con la finalidad de obtener un modelado portable a otras arquitecturas, de la misma compañía o de alguna otra, y considerando que el modelado será hecho con el lenguaje VHDL, en la descripción del **procesador de imágenes** no se considerará la arquitectura del FPGA Spartan 3E-500 FG320.

En la siguiente sección se presenta la descripción del modelado del **procesador de imágenes** usando las herramientas EDA y siguiendo la metodología de diseño y estilo de modelado elegidos.

4.2.1. Módulo Central de Proceso

El *módulo central de proceso* es el encargado de administrar la operación de los módulos que complementan al **procesador de imágenes** e indicarles las funciones a desarrollar por cada uno de ellos dependiendo de la información proveniente de la **interfaz de usuario**; en general, este módulo rige el funcionamiento del **procesador de imágenes**, llevando a cabo funciones como la recepción de la imagen a la cual se le aplicará un algoritmo de procesamiento, la decodificación de peticiones provenientes de la **interfaz de usuario** para indicar al **procesador de imágenes** la ejecución de una operación o función específica, administrar los tiempos de ejecución de los demás bloques, etc.

Este módulo representa el nivel más alto de jerarquía en el procesador y está diseñado como una máquina de estados, la cual opera a una frecuencia de 50 MHz. Contiene señales de entrada/salida mediante las cuales interactúa con los demás módulos y permiten añadir nuevos módulos al procesador, este tipo de señales comprende señales de control, buses de datos y recursos de comunicación o almacenamiento.

La máquina de estados para el diseño de este módulo se muestra en la Figura 4.4 y está compuesta por 11 estados cuya descripción se presenta a continuación:

- *inicial* y *decod_id*: *inicial* es el estado de inicio de la máquina de estados, en este estado se esperan datos provenientes del *módulo de control USB*, Sección 4.2.2, la llegada de datos es indicada cuando la señal `UsbReadOk` se encuentra en alto, este evento

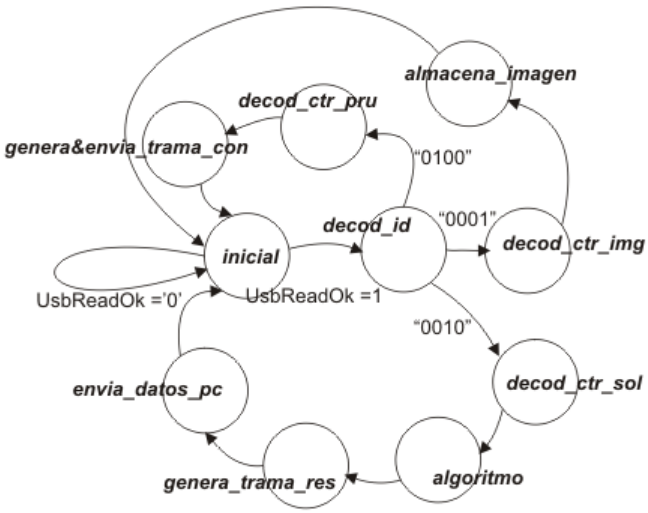


Figura 4.4: Máquina de estados para el diseño del Módulo Central de Proceso.

ocasoinará que la máquina salte al estado siguiente, *decod_id*. *decod_id* se encarga de decodificar el campo ID de las tramas recibidas por el **procesador de imágenes**; si el primer nibble es "0001", el estado siguiente será *decod_ctr_img*; si es "0010" será *decod_ctr_sol*, y para "0100" el estado siguiente es *decod_ctr_pru*.

- *decod_ctr_img* y *almacena_imagen*: *decod_ctr_img* es el estado al que la máquina saltará si el campo CTR de la trama es el correspondiente al "Envió de imagen" y en el cual se obtendrán los atributos de la imagen (alto y ancho), esta información es utilizada por el siguiente estado de la máquina, *almacena_imagen*, en el cual se almacena la información de la imagen en el sistema de memoria.

- *decod_ctr_pru* y *genera&envia_trama_con*: El primero de estos estados será ejecutado si en el campo CTR de la trama recibida se identifica la "Trama de prueba de comunicación" esto conseguirá que el estado siguiente sea el *genera&envia_trama_con*, en el cual se genera la "Trama de confirmación" y se la envía a la **interfaz de usuario**.

- *decod_ctr_sol* y *algoritmo*: Cuando en el campo CTR de la trama recibida se identifica la "Trama de solicitud de aplicación de algoritmo", el estado *decod_ctr_sol* obtiene el algoritmo o algoritmos que se van a ejecutar, y los parámetros que estos requieren. El siguiente estado es *algoritmo*, el cual realiza las operaciones necesarias para la ejecución del algoritmo requerido y almacena sus resultados en memoria.

- *genera_trama_res* y *envia_datos_pc*. El estado *genera_trama_res* es el encargado de generar la "Trama de resultados"de acuerdo al algoritmo o algoritmos ejecutados. El estado *envia_datos_pc* es el responsable de enviar los resultados a la PC.

A partir del modelado del *módulo central de proceso* en VHDL se obtiene el símbolo del mismo, Figura 4.5, y la descripción de sus entradas y salidas, Tabla 4.1.

El *módulo central de proceso* da seguimiento a las peticiones realizadas por la **interfaz de usuario**; una vez que la petición ha sido envíada, éste la recibe del *módulo de control USB*, Sección 4.2.2, y decodifica (estado *decod_id*) basándose en el protocolo de comunicación, Sección 4.4.4.

Cuando hay al menos 1 byte presente en el *módulo de control USB*, éste activa la señal `UsbreadOk` para indicar al *módulo central de proceso* que hay datos disponibles. El *módulo central de proceso* tendrá que activar la señal `ReadUsb` (2 ciclos de reloj) para obtener el byte que está en `FifoData`.

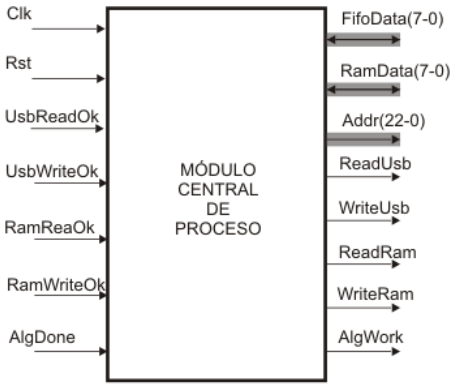


Figura 4.5: Símbolo del Módulo de Central de Proceso.

Señales	Ent.	Sal.	Descripción
Clk	✓		Señal de reloj para sincronizar la máquina de estados.
Rst	✓		Señal de reset para inicializar al módulo.
FifoData(7-0)	✓	✓	Arreglo de señales de 8 bits bi-direccional para lectura-escritura al módulo de control USB.
UsbReadOk	✓		Indica una lectura al USB finalizada.
UsbWriteOk	✓		Indica una escritura al USB finalizada.
RamReakOk	✓		Indica una lectura a la RAM finalizada.
RamWriteOk	✓		Escritura a la RAM finalizada.
Data(7-0)	✓	✓	Arreglo de señales de 8 bits bi-direccional para lectura-escritura al módulo de control RAM.
Addr(22-0)		✓	Arreglo de señales de 23 bits para dirreccionamiento a la RAM.
AlgDone	✓		Señal de finalización del algoritmo.
AlgWork		✓	Señal de inicialización del algoritmo.

Tabla 4.1: Señales de entrada y salida al Módulo Central de Proceso.

Si la petición recibida ha sido la de guardar la imagen a procesar en memoria, tendrá que hacer uso del *módulo controlador de SRAM*, Sección 4.2.3, para ello el *módulo central de proceso* cuenta con la señal `WriteRam` que le indica al *módulo de control SRAM* el inicio de la escritura de 1 byte presente en `RamData` en la localidad de la memoria del sistema direccionada por `ADDR`. En el momento en que la escritura finaliza, el *módulo de control SRAM* le indica al *módulo central de proceso* con la señal `RamWriteOk`.

Cuando el *módulo central de proceso* lee de la memoria los resultados obtenidos por un módulo de algoritmos de procesamiento y/o análisis de imágenes, hace uso de la señal `ReadRam` para iniciar la lectura, y la señal `RamReadOk` para checar que la lectura se ha realizado. Para enviar estos resultados a la **interfaz de usuario**, el *módulo central de proceso* tendrá que poner el byte obtenido de la memoria del sistema en `FifoData` y activar `WriteUsb`, con esto el *módulo de control USB* envía el byte presente en `FifoData` a un puerto USB de la PC, y cuando termina, activa la señal `UsbWriteOk` para que el *módulo central de proceso* pueda realizar otras acciones.

Para que el *módulo central de proceso* administre la ejecución de los módulos de algoritmos, Sección 4.2.4, cada uno de estos cuenta con dos señales de control: (`AlgWork`), mediante la cual el *módulo central de proceso* le indica al algoritmo que inicie su ejecución, y (`AlgDone`), utilizada por el algoritmo para indicar al *módulo central de proceso* que ha finalizado su ejecución; cuando el *módulo central de proceso* recibe una petición de ejecución de un algoritmo, activa la señal de inicialización de dicho algoritmo, y detiene su ejecución esperando por la señal de finalización del algoritmo para poder realizar otra acciones.

4.2.2. Módulo de control USB

Como se ha mencionado, la herramienta propuesta necesita de una interfaz de alta velocidad para el intercambio de información entre el **sistema hardware para el procesamiento de imágenes** y la **interfaz de usuario**. Se ha elegido al puerto USB de la PC para que cumpla con esta funciones.

La transferencia de datos de alta velocidad con la que cuenta la tarjeta Nexys-2, a través de un puerto USB 2.0, tiene por función principal permitir la configuración del FPGA; además, la compañía Digilent proporciona junto con la tarjeta Nexys-2 una biblioteca DLL que permite al usuario implementar un intercambio de información entre la PC y su aplicación en el FPGA; el uso de la biblioteca DLL presenta la desventaja de tener un número limitado de funciones y que al usar esta característica la portabilidad del sistema hacia otras tecnologías se pierde. Por los motivos mencionados, se ha decidido emplear un controlador USB externo, logrando mantener la compatibilidad y portabilidad hacia otros FPGAs.

El controlador USB elegido para implementar la interfaz mencionada es el DLP-USB245M de la compañía FDTI; debido a sus características, mencionadas a continuación, el controlador DLP-USB245M es ideal para implementar una interfaz entre una PC y un sistema autónomo.

Las características más importantes del DLP-USB245M son las siguientes:

- Velocidad de transferencia: 8 megabaudios.
- Cuenta con una FIFO de transmisión con un tamaño de 384 bytes, y una FIFO de recepción de 128 bytes.
- Drivers VCP y DLL para desarrollo de software.
- Cuenta con una interfaz simple hacia un CPU, un MCU, etc, a través de un puerto de entrada/salida.
- No se requiere de un conocimiento profundo del USB, puesto que el protocolo USB es manejado automáticamente dentro del módulo.
- Regulador de 3.3v integrado.
- 4.4v-5.25v tomados directamente del puerto USB.
- Compatible con USB 1.1 y USB 2.0.
- Ideal para prototipos en desarrollo.

El DLP-USB245M está compuesto por 24 pines, los cuales se describen detalladamente en la Tabla 4.2.

El *módulo de control USB*, con base en los diagramas de tiempos del controlador DLP USB245M, Figura 4.6 y Tabla 4.3, implementa el protocolo que permite al **procesador de imagénes** accesar a los recursos de este controlador para poder establecer el intercambio de información con la **interfaz de usuario**.

Cuando el DLP-USB245M recibe información proveniente de la PC (**interfaz de usuario**), activa su señal `RXF`, mandándola a nivel lógico bajo (`RXF='0'`), para indicarle al procesador (**sistema hardware para el procesamiento de imágenes**) que al menos un byte ha sido recibido y está en el buffer de recepción; en repuesta, el procesador debe activar la señal `RD` del DLP-USB245M (`RD='0'`) por un tiempo mínimo de 50ns (T1); una vez activa `RD` y cumpliéndose el tiempo T3 (30ns) el procesador puede leer el dato a través del bus de datos D7..D0. El procesador debe esperar 80ns antes de poder leer un nuevo dato.

Pin	Descripción
1	BOARD ID (salida) Identifica si la tarjeta es la DLP-USB245M (bajo) o la DLP-USB232M (alto).
2,5,7	GROUND Tierras.
3	RESET# (entrada) Puede ser usado por un dispositivo externo para poner en reset al FT245BM. Sin no se requiere, debe ser puesto a VCC.
4	RESET0# (salida) Salida del generador de reset interno. Se mantiene en alta impedancia duranre 2ms después de VCC > 3.5v y el oscilador interno inicia. Entonces se activa el regulador 3.3v interno. RESET# en bajo forzará a RSTOUT# a ir a alta impedancia. RSOUT# no es afectado por el reset del USB.
6	3V3OUT (salida) Salida del regulador L.D.O. integrado. Provee 3.3v para el transceiver del USB y el pin RSTOUT#.
8	SLEEP (salida) Va a nivel lógico bajo después de que el dispositivo es configurado vía USB, después va a alto durante la suspención del USB. Puede ser usado para controlar la alimentación para lógica externa usando un interruptor de nivel lógico, MOSFET canal-P.
9	SND/WUP (entrada) Si el DLP-USB245M está en USB suspendido, un flanco de subida en este pin (WAKEUP) inicia una secuancia de despierte remota. Si el dispositivo esta activo, un flaco de subida en este pin (SEND) causa que los datos en el buffer de escritura sean enviados a la PC en la proxima petición de datos sin importar cuantos bytes esten en el buffer.
10	VCC-IO (entrada) De 3.0v a 5.25v, VCC para los pines de interface de la UART. Cuando se interfaza a lógica externa de 3.3v, se conecta VCC-IO a la fuente de 3.3v de la lógica externa; en otro caso, se conecta a el VCC para manejar 5v de nivel CMOS. Este pin debe ser conectado a VCC desde EXTVCC.
11	EXTVCC (entrada) Usada para aplicar alimentación principal (4.4-5.25v) a el módulo. Se conecta a PORTVCC si el módulo es para ser alimentado por el puerto USB.
12	PORTVCC (salida) Alimentación desde el puerto USB. Se conecta a EXTVCC si el módulo es para ser alimentado por el puerto USB. 500mA es la corriente máxima disponible para el adaptador USB si el dispositivo USB es configurado para alta potencia.
13	RXF# (salida) Cuando se encuentra en nivel lógico bajo, al menos un byte esta presente en los 128 bytes de la FIFO de recepción, y se encuetra listo para ser leido con RD#. RXF# va a alto cuando el buffer de recepción está vacío.
14	TXE# (salida) Cuando se encuetra en nivel lógico bajo al menos un byte está listo para ser enviado a la PC.
15	WR (entrada) Cuando es llevado desde alto hacia nivel lógico bajo, WR lee los 8 bits y escribe el byte dentro del buffer FIFO de transmisión.
16	RD# (entrada) Cuando se encuetra en bajo, RD# lleva los 8 bits desde el estado de alta impedancia al byte actual en el buffer FIFO de tranmisión. Llevando a RTD# a alto este retorna los pines de datos al estado de alta impedancia, y prepara el próximo byte.
17-24	D7-D0 (entrada-salida) Bus de datos bi-direccional.

Tabla 4.2: Descripción de pines del DLP-USB245M.

El DLP-USB245M tiene un buffer de transmisión de 384 bytes, si este buffer está lleno o una transmisión está en curso, el DLP-USB245M lo indica activando su señal `TXE` (`TXE='1'`), informando que en ese momento no puede enviar datos a la PC. Si la señal `TXE` tiene un nivel lógico bajo, el procesador puede enviar información a la PC a través del DLP-USB245M; para llevar a cabo este proceso, el procesador únicamente debe escribir un dato al buffer de transmisión del DLP-USB245M, para conseguirlo, debe activar la señal `WR` del DLP-USB245M (`WR='1'`) por un tiempo de por lo menos 50ns y presentar el dato a transmitir al bus de datos D7..D0 (`UsbData`) 20 ns antes del flanco de caida de `WR`. Una vez contenido el dato en su buffer, el DLP-USB245M lo envia a la PC.

Con base en lo mencionado en los parrafos anteriores, se describe el *módulo de control USB*, el cual está formado por dos máquinas de estados, una para el proceso de lectura y

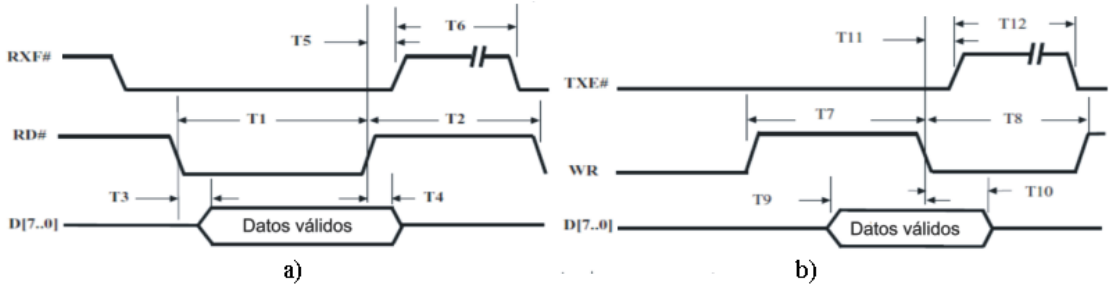


Figura 4.6: Tiempos nescesarios que deben de cumplir las señales de control del DLP-USB245M. (a) Ciclo de lectura FIFO . (b) Ciclo de escritura FIFO.

Tiempo	Descripción	Min	Max	Unidad
T1	Ancho de pulso de RD activo	50		ns
T2	Tiempo de pre-carga de RD a RD	50		ns
T3	RD activo a datos válidos		30	ns
T4	Tiempo de retención de datos válidos	10		ns
T5	RD inactivo a RXF#	5	25	ns
T6	RXF inactivo después del ciclo RD	80		ns
T7	Ancho de pulso de WR activo	50		ns
T8	Tiempo de pre-carga de WR a WR	50		ns
T9	Tiempo de establecimiento de datos antes de WR inactivo		20	ns
T10	Tiempo de retención de datos desde WR inactivo	10		ns
T11	WR inactivo a TXE#	5	25	ns
T12	TXE inactivo después del ciclo RD	80		ns

Tabla 4.3: Descripción de los tiempos de lectura y escritura FIFO del DLP-USB245M.

otra para el proceso de escritura. En la Figura 4.7 se muestra el símbolo obtenido de la descripción utilizando VHDL del *módulo de control USB* y en la Tabla 4.4 se detallan las funciones de cada una de las señales de este módulo.

La operación del *módulo de control USB*, cuando realiza un proceso de lectura y cuando se realiza uno de escritura al DLP-USB245M, puede ser apreciado en los diagramas de tiempos de la Figura 4.8

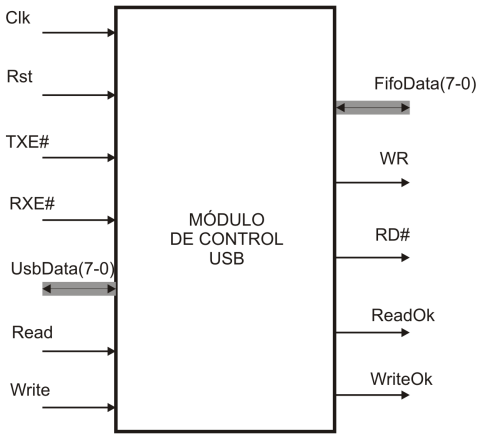


Figura 4.7: Símbolo del módulo de control USB.

Señal	Ent.	Sal.	Descripción
Clk	✓		Señal de reloj para proceso síncrono.
Rst	✓		Señal que inicializa al módulo de control USB.
TXE#	✓		Indica que el buffer de transmisión se encuentra vacío.
RXF#	✓		Indica que el buffer de recepción tiene al menos 1 byte.
FifoData(7-0)	✓	✓	Arreglo de señales de 8 bits para lectura/escritura al buffer FIFO.
Read	✓		Inicia lectura en el USB.
Write	✓		Inicia escritura en el USB.
UsbData(7-0)	✓	✓	Arreglo de señales de 8 bits para lectura/escritura del módulo de control USB.
WR		✓	Lee datos de FifoData y los escribe en el buffer FIFO de trasnmisión.
RD#		✓	Lee datos del buffer FIFO de recepción y los almacena en UsbData.
ReadOk		✓	Indica que una lectura al USB ha sido completada.
WriteOk		✓	Indica que una escitura al USB ha sido finalizada.

Tabla 4.4: Descripción de entradas y salidas del módulo de control USB.

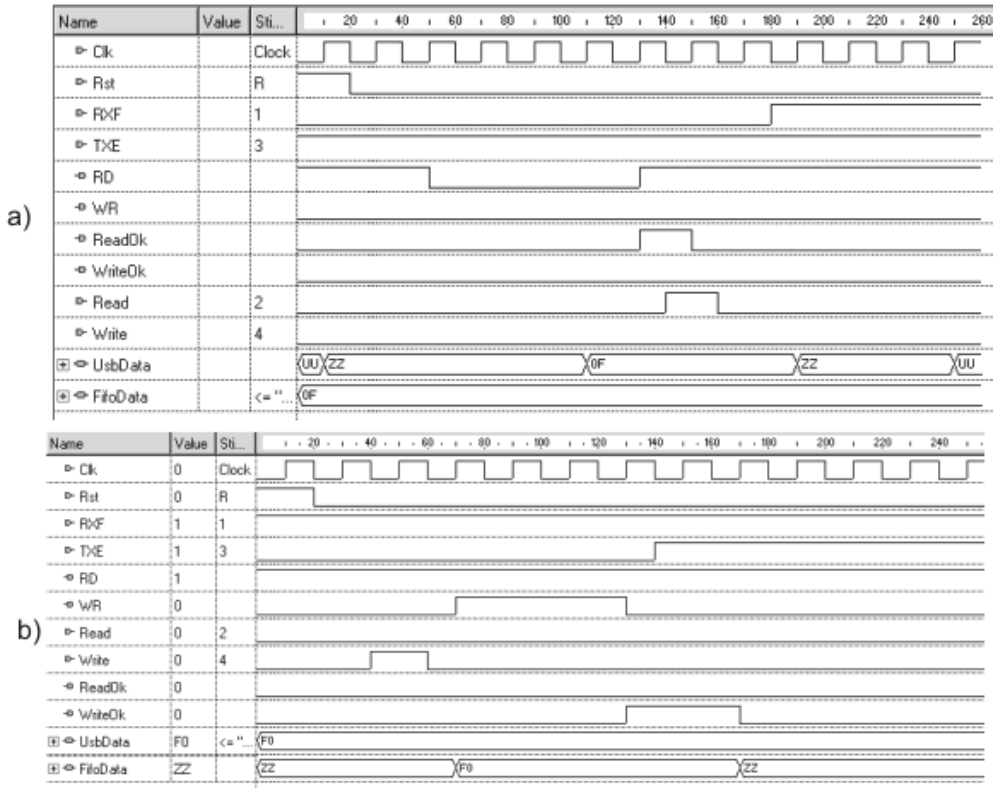


Figura 4.8: Diagrama de tiempos de módulo de control USB. (a) Lectura al USB. (b) Escritura al USB.

4.2.3. Módulo de control SRAM

El **procesador de imágenes**, como su nombre lo indica será utilizado para manipular imágenes, es decir aplicar algoritmos de procesamiento y/o análisis a una imagen; considerando que la imagen a manipular puede ser de un tamaño variable, un tamaño tipico es de 512×512 píxeles (8 bits/píxel), es necesario que el procesador tenga acceso a un espacio de almacenamiento externo (memoria de datos externa) que pueda utilizar como buffer de almacenamiento temporal de la imagen a procesar y de los resultados del procesamiento.

El sistema de memoria de datos del **sistema hardware para el procesamiento de imágenes** tiene una capacidad de 16 Mbytes y está formado por una memoria DRAM seudo-

estática M45W8MW 16 Cellular RAM de 128 Mbits organizada como 8 Mbytes×16 bits, de la compañía Micron. Este dispositivo puede operar como una memoria asíncrona SRAM con ciclos de lectura y escritura de 70 ns, o como una memoria síncrona con un bus de 80 MHz.

Las características principales de esta memoria son:

- Voltaje de alimentación: 1.7 - 3.3 Volts.
- Organización 8 Mbytes x 16.
- Modos de operación síncrona y asíncrona.
- Tiempos de acceso de 70 ns.
- Consumo de potencia menor a 35 mA.

La descripción de pines de la memoria M45W8MW 16 Cellular RAM puede ser consultada en la Tabla 4.5.

Señal	Descripción
ADDR(23:0)	Bus de direcciones de 24 bits para la SRAM.
DATA(15:0)	Bus de datos de 16 bits de la SRAM.
OE	Señal de habilitación de salida cuando se lee a la SRAM.
WE	Señal de habilitación de escritura a la SRAM.
MT-ADV	Dirección válida.
MT-CLK	Señal de reloj para SRAM en modo síncrono.
MT-UB	Señal de activación del byte superior para direccionamiento por byte.
MT-LB	Señal de activación del byte inferior para direccionamiento por byte.
MT-CE	Habilitación del chip.
MT-WAIT	Señal de espera para SRAM en modo síncron.

Tabla 4.5: Descipción de pines de entrada/salida del sistema de memoria SRAM M45W8MW 16.

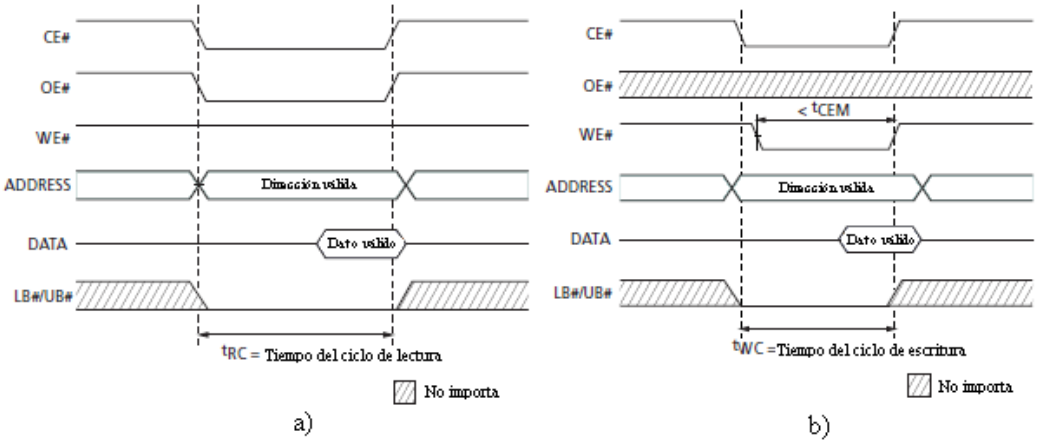


Figura 4.9: Diagrama de tiempos de acceso a la SRAM MT45W8MW16 en modo asíncrono. (a) Operación de lectura. (b) Operación de escritura.

Debido a la facilidad en la implementación de su protocolo y a que sus tiempos de acceso de alguna forma son compatibles con el controlador USB, se decidió utilizar el modo de operación asíncrono de la memoria M45W8MW, comportándose como una memoria SRAM con un tiempo de acceso de 70 ns (^tRC,^tWC). La Figura 4.9 muestra los diagramas de

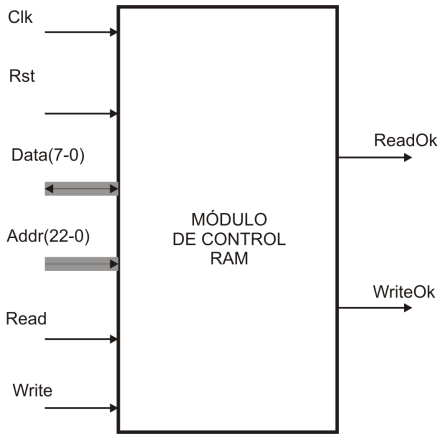


Figura 4.10: Símbolo del módulo de control SRAM.

tiempos de los ciclos de lectura y escritura de la memoria en el modo de operación elegido, mismos que serán empleados en la descripción del *módulo de control SRAM*.

El *módulo de control SRAM*, con base en los diagramas de tiempos de la memoria M45W8MW 16 Cellular RAM, implementa el protocolo que permite al **procesador de imagénes** almacenar o recuperar información en el sistema de memoria de datos. Este protocolo está formado por dos procesos síncrocnos, el que modela el ciclo de lectura a la memoria y el que modela el ciclo de escritura a la misma. En la Figura 4.10 se muestra el *módulo de control SRAM* generado con la herramienta ISE Foundation. La descripción de las señales que intervienen en este módulo puede ser consultada en la Tabla 4.6.

Señal	**Ent.**	**Sal.**	**Descipción**
Clk	✓		Señal de reloj para el proceso síncrono.
Rst	✓		Señal de inicialización del módulo de control SRAM.
Data(7-0)	✓	✓	Arreglo de 8 bits para lectura/escritura a la SRAM.
Addr(22-0)	✓		Arreglo de señales de 22 bits para direccionamiento a la SRAM.
Read	✓		Señal de inicio de lectura a la SRAM.
Write	✓		Inicio de escritura a la SRAM.
ReadOk		✓	Indica una lectura a la SRAM finalizada.
WriteOk		✓	Indica una escritura a la SRAM finalizada.

Tabla 4.6: Descripción de entradas y salidas del módulo de control SRAM.

El *módulo de control SRAM* puede ser utilizado por el *módulo central de proceso* o por cualquier *módulo de algoritmo* cuando requieren accesar a los datos almacenados en la memoria de datos o cuando requieren almacenar información en la misma.

Cuando un módulo requiere accesar información de la memoria de datos, debe direccionar la localidad deseada a través del bus `ADDR[22..0]` y activar un proceso de lectura mediante la señal `Read`; con esa información, el *módulo de control SRAM* implementa el protocolo de acceso a la memoria y al terminar este proceso lo indica a través de la señal `ReadOk`, esta señal indica al módulo que solicitó el acceso a la memoria que el dato está disponible en el bus `Data[7..0]`.

Cuando un módulo requiere almacenar información en la memoria de datos, debe indicar en que localidad se almacenará el dato utilizando el bus `ADDR[22..0]`, el dato a almacenar debe ser puesto en el bus `Data[7..0]` y debe activar el proceso de escritura mediante la señal `Write`; con esa información, el *módulo de control SRAM* implementa el protocolo de escritura a la memoria y al terminar este proceso lo indica a través de la señal `WriteOk`, esta señal indica al módulo que requirió se almacenara un dato que su solicitud ha sido efectuada.

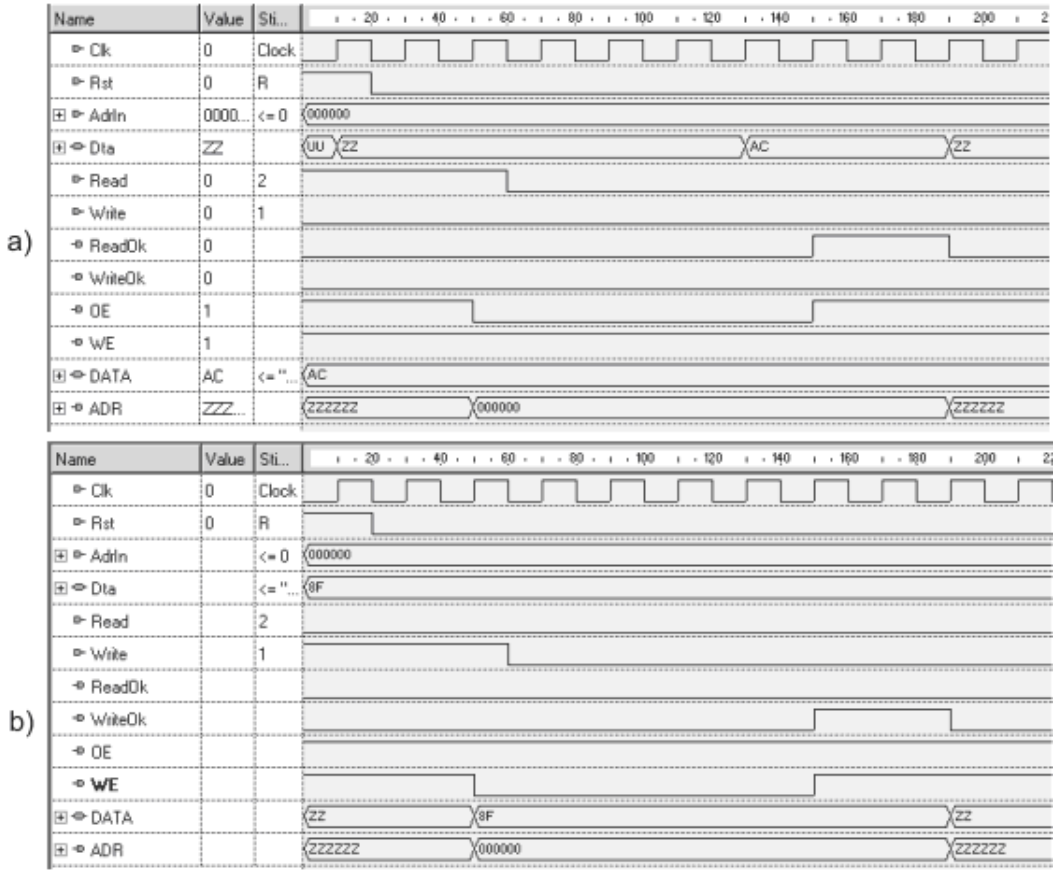


Figura 4.11: Diagrama de tiempos del módulo de control SRAM. (a) Lectura a la SRAM. (b) Escritura a la SRAM.

Los diagramas de tiempos obtenidos durante accesos al sistema de memoria por parte del *módulo de control SRAM* se muestran en la Figura 4.11.

4.2.4. Módulos de algoritmos

El objetivo principal del sistema propuesto, es permitir al usuario evaluar el desempeño de algoritmos de procesamiento y/o análisis de imágenes cuando estos son modelados en un FPGA. Para lograr dicho propósito, el **procesador de imágenes** fue modelado para que el usuario pueda fácilmente añadirle nuevos módulos; estos módulos representan los algoritmos que el usuario desea evaluar.

Para poder integrar un *módulo de algoritmo* al **procesador de imágenes**, es necesario cubrir ciertos requisitos, los cuales se listan a continuación:

- Debe incluir señales que le permitan establecer una comunicación con el *módulo central de proceso*. Estas señales de control son: la señal de entrada de inicialización del algoritmo, `AlgWork`, y la señal de salida de finalización de ejecución del algoritmo `AlgDone`.

- Debe incluir señales que le permitan establecer una comunicación con el *módulo de control SRAM*. Estas señales son de dos tipos, por un lado señales de datos, representadas por `RamData` (bus de datos) utilizado para el indicar el byte que se va a escribir o leer al sistema de memoria; y por el otro las señales de control: `ADDR` (bus de direcciones) indica la localidad del sistema de memoria que será accesada, `WriteRam` (salida) y `RamWriteOk` (entrada) utilizadas para iniciar y finalizar una escritura al sistema de

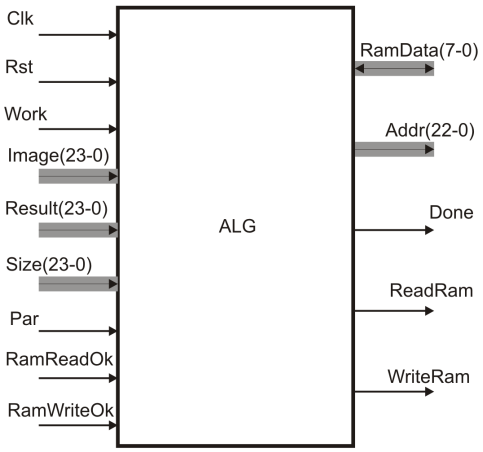


Figura 4.12: Símbolo de un módulo de algoritmo de procesamiento y/o análisis de imágenes.

memoria respectivamente, `ReadRam` (salida) y `RamReadOk` (entrada) usadas para iniciar una lectura y para saber que dicha lectura se ha concluido.

Debido a las diferencias que existen en los algoritmos de procesamiento y análisis de imágenes, se establece que para cada uno debe haber un bus para la localidad de memoria de la imagen a procesar (`Image`), otro para el tamaño de la imagen (`Size`), y uno más para la dirección de memoria de la información resultante (`Result`). Cuando se da el caso en que el algoritmo posee parámetros, como pueden ser umbrales o propiedades de la imagen a procesar y/o analizar, estos debe de entrar al algoritmo como un arreglo de bits (`Par`), cuya tamaño depende de las necesidades de cada parámetro.

En la Figura 4.12 se muestra un módulo de algoritmo de procesamiento y/o análisis de imágenes obtenido con la herramienta ISE Foundation; la descripción de las señales de entrada y salida de este módulo se presenta en la Tabla 4.7.

Señal	**Ent.**	**Sal.**	**Descripción**
Clk	✓		Señal de reloj para el proceso síncrono.
Rst	✓		Señal para la reset del algoritmo.
Work	✓		Señal de inicialización del algoritmo.
RamData(7-0)	✓	✓	Arreglo de 8 bits para lectura/escitura de datos a la SRAM.
Addr(22-0)		✓	Arreglo de 23 bits para el direccionamiento a la SRAM.
Image(23-0)	✓		Localidad SRAM de inicio de la imagen a procesar.
Size(23-0)	✓		Tamaño en bytes de la imagen a procesar.
Result(23-0)	✓		Localidad SRAM de inicio de la información resultante.
ReadRam		✓	Señal para la lectura a la RAM.
WriteRam		✓	Señal para la escritura a la SRAM.
RamReadOk	✓		Lectura a la SRAM finalizada.
RamWriteOk	✓		Escritura a la SRAM finalizada.
Par	✓		Parámetro opcional del algoritmo.
Done		✓	Señal de finalización del algoritmo.

Tabla 4.7: Descripción de pines de entrada y salida de los algoritmos de procesamiento y análisis de imágenes.

Para el modelado de estos módulos se puede ocupar una máquina de estados dentro de un proceso en VHDL. La cantidad de estados de cada módulo depende de la complejidad del algoritmo.

Como se mencionó en la Sección 4.2.1, cuando se realiza una petición para la ejecución de un algoritmo, el *módulo central de proceso* detiene su ejecución, pasa una cantidad de parámetros necesarios al algortimo e inicia la ejecución del mismo activando la señal `AlgWork`.

Dentro del módulo de algoritmo es monitoreada la señal al inicio de la máquina de estados, y cuando su valor es de '1', nivel lógico alto, el algoritmo toma los parámetros y comienza a procesar y/o analizar la imagen.

En el análisis o procesado de la imagen los algoritmos deben leer y almacenan datos en la SRAM. Para leer un dato de la SRAM, el algoritmo tendrá que indicar la localidad de memoria (`ADDR`) y llevar a `ReadRam` a un nivel lógico alto por 2 ciclos de `Clk`; cuando la señal `RamReadOk` (entrada) se activa se sabrá que el dato fue leído. Por otra parte, para escribir datos a la SRAM se indica la dirección de memoria deseada, se escribe en el bus de datos el valor que se quiera almacenar, y por último se activa `WriteRam`; el algoritmo espera hasta que `RamWriteOk` tiene un nivel lógico alto para que el algoritmo realice otra acción.

Al finalizar el pocesamiento y/o análisis, los resultados del algoritmo se encuentran en SRAM a partir de la localidad `Result`. El algoritmo activa `AlgDone` por 2`Clk`, suficientes para que el *módulo central de proceso* reciba otras peticiones y ejecute otros algoritmos.

En el Apéndice B se muestra como quedan interconectados el *módulo de control USB* y *módulo de control SRAM* con el *módulo central de proceso*; además, se muestra como se interconecta un *módulo de algoritmo* genérico al mismo.

4.3. Sistema hardware para el procesamiento de imágenes

En esta segunda fase, denominada **sistema hardware para el procesamiento de imágenes**, se presenta el diseño e implementación de un sistema orientado al procesamiento y análisis digital de imágenes utilizando la metodología de diseño de un sistema empotrado; el **sistema hardware para el procesamiento de imágenes** basa su diseño en el **procesador de imágenes**, modelado en la primera fase.

Un sistema embebido o empotrado es un conjunto de dispositivos electrónicos que incluye procesadores dentro de sus implementaciones con el propósito principal de simplificar el diseño del sistema, proporcionar flexibilidad, remover errores, hacer modificaciones, y sumar nuevas características con sólo rescribir el software que controla al dispositivo, además es diseñado para realizar una función dedicada, [26], [32], [8]; la esencia del diseño de sistemas embebidos es que implementan un conjunto específico de funciones mientras satisfacen las restricciones sobre características tales como desempeño, costo, emisión, consumo de potencia, y peso, [42]. Un sistema empotrado es aquel sistema que se encuentra formado por una combinación de software y hardware, y eventualmente de componentes mecánicos u otras partes, específicamente diseñado y optimizado para resolver un problema concreto [20], [7]. El **sistema hardware para el procesamiento de imágenes** cumple con la definición de los sistemas embebidos, tanto en software como en hardware, por lo tanto será diseñado siguiendo la metodología de desarrollo de los mismos.

Para el diseño de sistemas empotrados, se emplea un ciclo de diseño que cuenta con la interacción y optimización de siete fases, Figura 1.3, las cuales se describen a continuación:

1. *Especificación del producto.* En esta fase del desarrollo de sistemas empotrados se define de la manera más clara y precisa lo que hará el sistema a desarrollar, es decir, es una descripción de los requerimientos técnicos y funcionales del sistema que se va a implementar, con el objetivo de hacerlo robusto. Entre otros aspectos, en esta fase se determinan cuales serán sus entradas y salidas del sistema, se especifica la interfaz de interacción del sistema con el usuario, el tipo de información que procesará, etc.

2. *Selección del procesador.* Con la finalidad de elegir el procesador que nos brinde la mejor opción para alcanzar el producto especificado, en esta fase se hace un estudio de los procesadores existentes en el mercado que puedan satisfacer las necesidades de la aplicación. En la toma de esta decisión se consideran los pines de entrada/salida

que requieren las entradas/salidas del sistema a desarrollar, las interfaces requeridas, protocolos de comunicación, consideraciones en tiempo real, el ambiente de desarrollo, la velocidad de procesamiento, el costo del procesador, etc.

3. *Definición de software y hardware.* La elección del hardware que se va a emplear, considerando que sus características resuelvan el problema planteado, entran dentro de esta fase. Para el caso de la definición o especificación del software, este incluye a la declaración de requisitos basados en la definición del hardware, las especificaciones de ingeniería y la definición general de los requerimientos del sistema.

4. *Sistema de evaluación.* En esta fase, con la ayuda de herramientas especializadas, la interacción de los elementos de hardware y de software es simulada y evaluada. El uso de emuladores y simuladores (herramientas EDA) hace posible que el diseñador conozca con cierta anticipación el comportamiento del sistema que será implementado; gracias a esto se podrán detectar y corregir posibles errores de funcionalidad.

5. *Diseño paralelo de software y hardware.* Es dentro de esta fase donde se diseñan de manera detallada, todos los componentes de software y hardware que fueron especificados en los requerimientos o especificaciones del producto, poniendo especial atención en mantener la coherencia entre los componentes del producto final.

6. *Integración de componentes de software y hardware.* En esta fase del desarrollo del sistema empotrado se integran los componentes desarrollados en la fase anterior verificando que la interacción entre ellos sea la correcta. Para tal fin, se debe de contar con las herramientas y los métodos adecuados en la integración del sistema.

7. *Prueba y verificación del producto.* Por último, en esta fase del desarrollo, se realizan las pruebas necesarias al producto resultante para constatar su correcto funcionamiento, y si es preciso, se resalizan modificaciones y/o correciones para adaptar el producto a las necesidades del usuario y posteriormente entregarlo como un producto terminado.

En las siguientes subsecciones se describe el diseño del **sistema hardware para el procesamiento de imágenes** siguiendo la metodología de diseño de un sistema empotrado y utilizando como elemento central al **procesador de imágenes** modelado en un FPGA.

4.3.1. Especificación del producto

Las especificaciones iniciales del **Sistema hardware para el procesamiento de imágenes** son las siguientes:

1. El **sistema hardware para el procesamiento de imágenes** debe cumplir dos propositos: 1. Ser un sistema que permita al diseñador desarrollar aplicaciones que requieran de un sistema autónomo; 2. Formar parte de una herramienta de evaluación de algoritmos de procesamiento y análisis de imagenes cuando son implementados en un CDC.

2. El procesador utilizado en el diseño del **sistema hardware para el procesamiento de imágenes** debe ser implementado en los recursos de un FPGA.

3. Las entradas al sistema serán imágenes en tonos de gris con un tamaño máximo de 256×256 píxeles, las cuales serán analizadas y/o procesadas de acuerdo al algoritmo implementado. Las salidas podrán ser imágenes con niveles de gris u otros resultados (no precisamente imágenes sino datos), estas salidas dependerán del tipo de procesamiento y/o análisis que se le aplique a las imágenes de entrada.

4. El **sistema hardware para el procesamiento de imágenes** dispondrá de una interfaz de comunicación para recibir la información de la imagen a procesar y/o analizar, y devolver los resultados obtenidos del procesamiento de la imagen. El protocolo de comunicación que usará dicha interfaz debe tener una alta velocidad de transferencia de datos.

5. El **sistema hardware para el procesamiento de imágenes** debe disponer de un sistema de almacenamiento que podrá utilizar como buffer de almacemamiento temporal de la imagén original y de los datos resultantes del procesamiento y/o analisis.

6. El **procesador de imágenes**, elemento central del **sistema hardware para el procesamiento de imágenes**, tendrá implementados algoritmos de procesamiento y/o análisis de imágenes básicos.

7. La arquitectura del **sistema hardware para el procesamiento de imágenes** estará orientada para facilitar al usuario el diseño e implementación de nuevos algoritmos de procesamiento y/o análisis de imágenes.

4.3.2. Selección del procesador

En la Sección 4.2, *Modelado de un procesador de aplicación específica*, se ha diseñado un procesador de aplicación específica, denominado **procesador de imágenes** debido a que está orientado al procesamiento y análisis digital a imágenes. Este procesador fue modelado haciendo uso del lenguaje de descripción de hardware VHDL, e implementado en el FPGA Spartan 3E-500 de la compañía Xilinx, los FPGAs se usan cada vez más como plataformas de desarrollo de producto finales, su empleo depende de la aplicación, del funcionamiento deseado, del desarrollo y costes de producción, [38]. Estos hechos permiten con gran facilidad modificar la arquitectura del procesador o ampliarla añadiendo nuevos módulos, por ejemplo, el usuario puede integrar nuevos algoritmos de procesamiento y/o análisis de imágenes.

Algunas otras características con las que cuenta el **procesador de imágenes** son los módulos utilizados en el control de una SRAM (sistema de almacenamiento) y de un controlador de protocolo USB (recepción-transferencia de datos); además permite procesar imágenes en tonos de gris no mayores a 256×256 píxeles.

Las características mencionadas en los parrafos anteriores hacen del **procesador de imágenes** la elección ideal del elemento central a utilizar en el desarrollo del **sistema hardware para el procesamiento de imágenes**, quedan cubiertos diversos puntos de las especificaciones mencionadas en la Seccion 4.3.1.

Cabe hacer mención que no fue necesario un estudio comparativo de procesadores existentes en el mercado que pudieran ser utilizados en el desarrollo de este sistema, debido a que uno de los objetivos que persigue el desarollo del **sistema hardware para el procesamiento de imágenes** es servir como una herramienta de evaluación de algoritmos de procesamiento y/o análisis de imágenes cuando estos son modelados en un FPGA.

4.3.3. Definición de los componentes de hardware y software

El **sistema hardware para el procesamiento imágenes** incluye tanto componentes de software como de hardware, los cuales deben ser definidos. En lo que concierne a los componentes de software, se tiene al controlador para la comunicación con el USB (Sección 4.2.2) y el controlador del sistema de memoria (Sección 4.2.3), ambos implementados en el **procesador de imágenes** modelado en la primera fase (Sección 4.2). Con respecto a los componentes de hardware se tiene al *controlador USB externo* y el *sistema de memoria*.

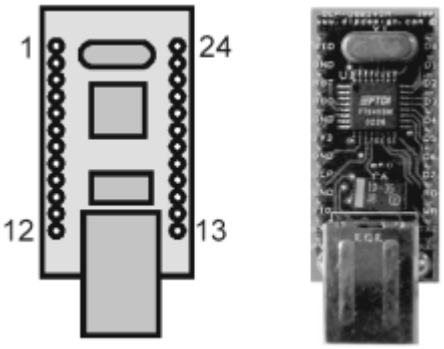


Figura 4.13: Controlador de USB externo DLP-USB245M.

4.3.3.1. Definición de los componentes de hardware

Los componentes de hardware son aquellos dispositivos digitales que auxilian al **sistema hardware para el procesamiento de imágenes** en la obtención de las imágenes que serán procesadas y/o analizadas, en el almacenamiento de las mismas, y en el envío de los resultados obtenidos a partir del algoritmo de procesamiento y/o análisis de imágenes implementado en el **procesador de imágenes**.

4.3.3.1.1. Controlador USB externo

El *controlador USB externo* es empleado para obtener una transferencia de información a una velocidad alta entre el **sistema hardware para el procesamiento de imágenes** y una PC. El *controlador USB externo* que se usará es el DLP-USB245M, manteniendo así la portabilidad del sistema hacia diversas aplicaciones, hacia otras tecnologías y hacia diversas herramientas de modelado.

El *controlador USB externo* DLP-USB245M provee un método fácil y económico de transferencia de datos desde un periférico hacia una computadora cliente, o viceversa, a una velocidad arriba de 8 millones de bits por segundo (1 Mbyte). Su diseño simple basado en FIFOs lo hace fácil de interfazar a cualquier microcontrolador o microprocesador por medio de puertos de entrada/salida.

Si se requiere mandar datos desde un periférico hacia una computadora cliente, se escribe la información en bytes sobre el módulo DLP-USB245M cuando TXE# se encuentra en un nivel lógico bajo. Cuando el buffer de transmisión se encuentra lleno (384 bytes) o el dispositivo está ocupado almacenando el byte previo escrito, el DLP-USB245M manda a un nivel lógico alto a TXE# para evitar que más datos sean escritos hasta que alguno de los datos de la FIFO haya sido transferido, sobre el USB hacia la computadora cliente.

Cuando la computadora cliente manda datos por el USB hacia el periférico, el dispositivo manda a RXF# a un nivel lógico bajo para permitirle al periférico saber que al menos un byte de datos se encuentra disponible. En ese momento el periférico puede leer datos hasta que RXF# retorne a un nivel lógico alto, indicando que ya no existen datos disponibles para ser leídos.

En la Figura 4.13 se muestra la apariencia física del DLP-USB245M, las características más importantes del DLP-USB245M, como son la descripción de los pines y los diagramas de tiempos para su acceso, pueden ser consultados en la Sección 4.2.2.

4.3.3.1.2. Sistema de memoria

Para que el **sistema de hardware para el procesamiento de imágenes** cumpla con su propósito de analizar y/o procesar imágenes es necesario contar con un *sistema de memoria* que tiene como función servir como un espacio de almacenamiento temporal de los

datos que forman parte de la imagen a procesar y/o analizar enviada desde una PC a través del *controlador USB externo* y almacenar los datos resultantes de dicho procesamiento.

La tarjeta Nexys-2 (Apéndice A) tiene integrado un dispositivo de RAM externa, se trata de una pseudo-estática DRAM M45W8MW 16 Cellular RAM de 128Mbit perteneciente a la compañía Micron; esta memoria está organizada como 8Mbytes×16bits. Puede operar como una SRAM asíncrona típica con tiempos de ciclos de lectura y escitura de 70ns; en este modo de operación, la Cellular RAM automáticamente refresca sus arreglos internos de DRAM, con lo cual requiere de un controlador de menor complejidad.

La SRAM que se encuentra en la tarjeta Nexys-2 se elige para conformar al *sistema de memoria* del **sistema de hardware para el procesamiento de imágenes**, debido a que se encuetra en la misma plataforma de desarrollo del FPGA Spartan 3E-500, a que tiene una gran capacidad y a que la implementación del controlador para esta memoria es muy simple. Cuenta con un bus de datos de 16 bits y 24 bits de bus de direcciones, y con las señales de control: OE para la habilitación de salida cuando se lee a la SRAM, y WE para la habilitación de la escritura, además de la señal de habilitación del chip (CE).

Las características más relevantes de la memoria DRAM M45W8MW 16 Cellular RAM, como son la descripción de los pines y los diagramas de tiempos para su acceso, pueden ser consultados en la Sección 4.2.3.

4.3.3.2. Definición de los componentes de software

Los componentes de software del **sistema hardware para el procesamiento de imágenes** están basados en la definición de los componentes de hardware; el controlador de la comunicación USB basa su funcionamiento en el *controlador USB externo* DLP-USB245M, y el controlador del *sistema de memoria* se basa en la SRAM M45W8MW 16. La definición de ambos componentes de software, o sistemas de control de hardware, se realizó en la primera fase del diseño de este trabajo, en las Secciones 4.2.2 y 4.2.3, y de este modo queda cubierta esta etapa de la metodología de sistemas empotrados aplicada al **sistema hardware para el procesamiento de imágenes**.

4.3.4. Sistema de evaluación

Esta fase, haciendo uso de herramientas especializadas, tiene por finalidad predecir el comportamiento del sistema antes de empezar el diseño de sus componentes, buscando posibles errores que se pudieran presentar, asi como dar la solución más adecuada a los mismos. Al carecer de una herramienta especializada que permita emular o simular el **sistema hardware para el procesamiento de imágenes** antes de su implementación, el desarrollo de esta fase se llevará a cabo en forma individual para cada uno de los componente del sistema en la fase siguiente, Diseño paralelo del hardware y software.

4.3.5. Diseño paralelo del hardware y el software

Ya que las tareas que realizarán los componentes de software y hardware fueron definidas con sus respectivas especificaciones, será necesario realizar una descripción detallada del diseño paralelo de cada uno de estos elementos del sistema, buscando asegurar la congruencia entre ellos mediante la evaluación de su funcionamiento en forma conjunta para cerciorar el correcto funcionamiento del **sistema hardware para el procesamiento de imágenes**.

4.3.5.1. Componentes de software

El controlador del sistema de memoria (*módulo de control SRAM*), el controlador del DLP-USB245M (*módulo de control USB*) y los módulos que modelan los algoritmos de

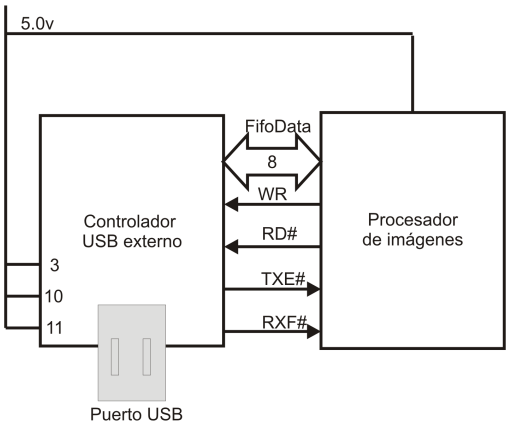


Figura 4.14: Conexiones entre el procesador de imágenes y el DLP-USB245M.

procesamiento y/o análisis de imágenes (*módulos de algoritmos*), representan a los componentes de software y cuyo modelado fue documentado detalladamente en las Secciones 4.2.3, 4.2.2 y 4.2.4 respectivamente.

4.3.5.2. Componentes de hardware

La memoria DRAM M45W8MW 16 Cellular RAM es utilizada para formar al *sistema de memoria* y el DLP-USB245M para implementar la transferencia de información entre el **sistema hardware para el procesamiento de imágenes** y la **interfaz de usuario**, ambos elementos forman parte de los componentes de hardware.

4.3.5.2.1. Componente de interfaz USB

El **sistema hardware para el procesamiento de imágenes** contará con una transferencia de información de alta velocidad con la **interfaz de usuario**, se ha mencionado que el dispositivo elegido para este propósito es el DLP-USB245M.

Considerando las características de este dispositivo, descritas en las Secciones 4.2.2 y 4.3.3.1.1, se diseña la interfaz entre el **procesador de imágenes** y el DLP-USB245M, mostrada en la Figura 4.14.

4.3.5.2.2. Componente del sistema de memoria

El **sistema hardware para el procesamiento de imágenes** debe contar con un sistema de memoria externo que funga como un buffer de almacenamiento temporal donde

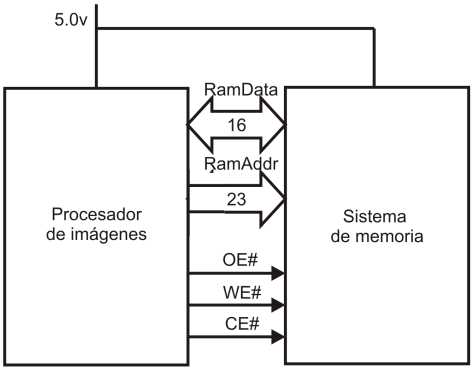


Figura 4.15: Conexiones entre el procesador de imágenes y la DRAM M45W8MW 16 Cellular RAM.

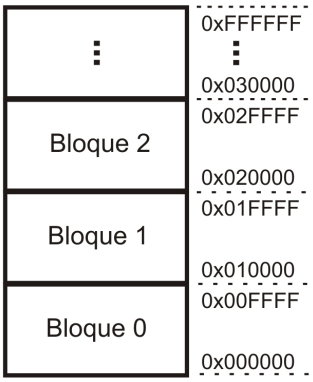


Figura 4.16: Organización del sistema de memoria.

se pueda almacenar la información a procesar y los resultados de dicho procesamiento, el dispositivo elegido para formar el sistema de memoria es la DRAM M45W8MW 16 Cellular RAM.

Considerando las características de este dispositivo, descritas en las Secciones 4.2.3 y 4.3.3.1.2, se diseña la interfaz entre el **procesador de imágenes** y la DRAM M45W8MW 16 Cellular RAM, mostrada en la Figura 4.15.

La organización de la memoria se muestra en la Figura 4.16. Debido a que las imágenes a procesar son de 256×256 píxeles máximo, se ha dividido el sistema de memoria en segmentos o bloques fijos de 0x10000 bytes. La imagen a procesar se almacena en el bloque inicial, y a cada algoritmo de procesamiento y/o análisis implementado le corresponde uno de los subsecuentes bloques; en el caso en que cierto algoritmo requiera de varios bloques de memoria para sus resultados, se asignarán y no podrán ser ocupados por otros algoritmos.

4.3.6. Integración de los componentes de hardware y software

La integración de los componentes que conforman al **sistema hardware para el procesamiento de imágenes** se llevó a cabo interconectando al **procesador de imágenes** con el sistema de memoria y el controlador USB externo, logrando así, integrar completamente al **sistema hardware para el procesamiento de imágenes**, Figura 4.17.

Una vez integrado el **sistema hardware para el procesamiento de imágenes** se lleva a cabo la programación o configuración del FPGA Spartan 3E-500, en el cual se encuentra alojado el **procesador de imágenes** y el software que controla a los elementos de hardware. Esta configuración del FPGA se realiza con ayuda del software *Digilent ExPort*, cuya interfaz

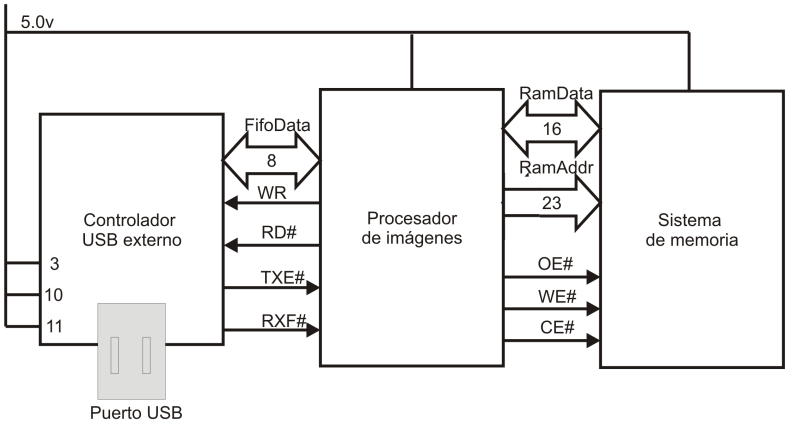


Figura 4.17: Integración del sistema hardware para el procesamiento de imágenes.

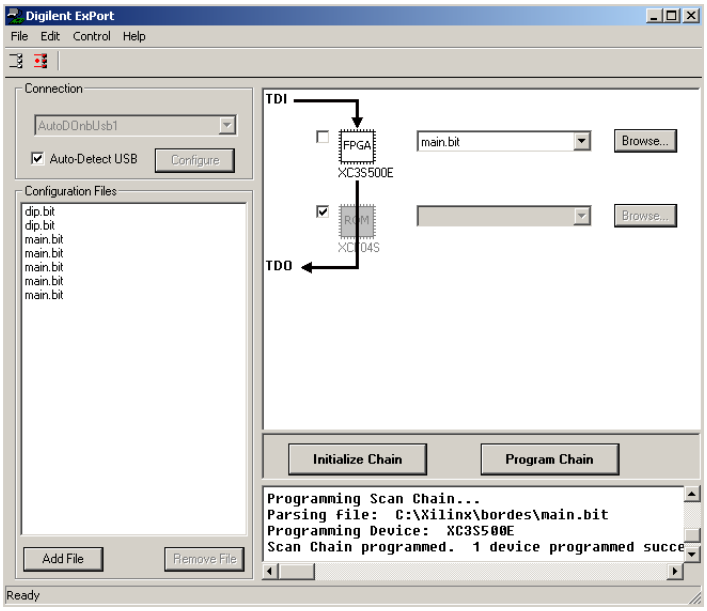


Figura 4.18: Software Digilent ExPort usado para la configuración del FGPA 3E-500.

se muestra en la Figura 4.18, al recibir un archivo de tipo *bit* generado a partir de los módulos que constituyen al procesador de aplicación específica en VHDL, y la síntesis-implementación de los mismos. El software *Digilent ExPort* asigna recursos y realiza conexiones internas en el FPGA; esta configuración o programación es ISP y con un reset al FGPA se borra. Se ocupa este tipo de configuración debido a que la herramienta está enfocada al diseño de prototipos, y es más rápido realizar las modificaciones desde la PC que programando la memoria *flash* de configuración del FPGA; aunque esta última configuración sigue siendo ISP, se llevará a cabo después de la prueba y verificación del sistema, se programará la memoria *flash* y se tendrá una versión final fija.

4.3.7. Prueba y verificación del producto

Una vez que el sistema ha sido integrado y configurado, dentro de esta fase se realizan las pruebas necesarias para verificar su correcto funcionamiento. El correcto funcionamiento del sistema se basa en el cumplimiento de los requisitos del mismo, Sección 4.3.1, y de la entrega de resultados visibles al usuario. La siguiente prueba fue realizada al **sistema hardware para el procesamiento de imágenes**:

Se envió una imagen a través de un puerto USB de la computadora al **sistema hardware para el procesamiento de imágenes** para comprobar el buen funcionamiento de los componentes de hardware y software relacionados con la transferencia de información por USB. La imagen es de 23×7 píxeles en tonos de gris (8 bits/píxel) que contiene la palabra "imagen", Figura 4.19.

La imagen es recibida por el **procesador de imágenes** y almacenada en el sistema de

Figura 4.19: Imagen de 23×7 para prueba y verificación del producto.

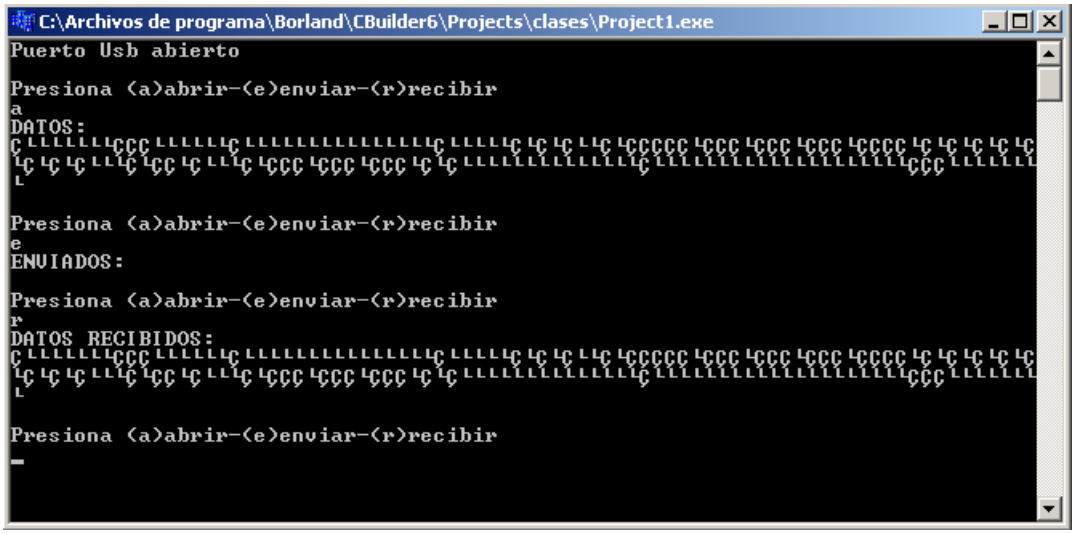


Figura 4.20: Prueba realizada al sistema de hardware para el procesamiento de imágenes.

memoria; esta prueba también verifica que el software que controla al sistema de memoria sea seguro y los accesos de escritura y lectura sean correctos.

Luego que se almacena la imagen se hace una petición al **procesador de imágenes** para que vuelva a leer la imagen del sistema de memoria y la regrese a la computadora. En la Figura 4.20 se muestra la realización de la prueba. Se abre la imagen, se envía, y por ultimo se recibe.

La imagen recibida se muestra en la Figura 4.21, y comparada con la imagen de la Figura 4.19 resultan ser iguales, por lo tanto se han obtenido los resultados esperados. Cabe aclarar que hasta ahora no se han mostrado los resultados de la implementación de algún algoritmos de procesamiento y/o análisis de imágenes, tan sólo se ha enviado y recibido una imagen en tonos de gris para verificar el funcionamiento del sistema. La visualización de los resultados de un algoritmo está documentada después del diseño e implementación de la **interfaz de usuario**.

4.4. Interfaz de usuario

Un elemento importante en el desarrollo de una herramienta de evaluación donde interviene un elemento hardware, como un procesador, es aquel que permite interactuar al usario con dicha herramienta. Esta fase, presenta el desarrollo de la **interfaz de usuario** que permite al usuario enviar a la herramienta la imagen que se desea procesar y visualizar los resultador generados por la misma.

Es decir, la **interfaz de usuario** es la parte del sistema que se encarga de proporcionar las imágenes a procesar y/o analizar y ayuda a visualizar las imágenes y los resultados obtenidos de aplicarles una técnica de procesamiento y/o análisis específica; agiliza el proceso de modelado-desarrollo-implementación permitiendo valorar los resultados obtenidos.

Figura 4.21: Imagen de 23×7 píxeles resultante de la prueba y verificación del producto.

Una interfaz de usuario *utilizable* debe proveer acceso a las funciones y características de una aplicación, de tal manera que refleje la forma de pensar del usuario acerca de las tareas que la aplicación potencial soportará. Esto requiere que la aplicación no sólo proporcione soporte para aspectos necesarios del trabajo del usuario, sino que debe también proveer los significados para ellos con el fin de interactuar con la aplicación en forma intuitiva y naturalmente [51].

La **interfaz de usuario** debe ser programada siguiendo las siguientes especificaciones:

1. Debe brindar opciones de inicialización, apertura y cierre de un puerto USB de la PC, para la comunicación con el **sistema hardware para el procesamiento de imágenes**.

2. Debe implementar funciones de envío de imágenes al **sistema hardware para el procesamiento de imágenes** y recepción de resultados provenientes del mismo, a través del puerto USB de la PC previamente abierto. Los resultados recibidos puedieran ser imágenes u otros datos resultantes.

3. Permitir implementar un tratamiento previo a las imágenes antes de ser enviadas al sistema hardware; este tratamiento puede ser la adición de ciertos tipos de ruido a la imagen a procesar.

4. Visualización de las imágenes a procesar, del tratamiento previo (si lo hubiera), y de los resultados obtenidos al procesar y/o analizar dichas imágenes.

La herramienta resultante de este trabajo de tesis, podrá ser utilizada por investigadores, profesores y alumnos, dedicados al desarrollo de sistemas autónomos que modelen algoritmos de procesamiento y/o análisis de imágenes, como un medio que permita evaluar el comportamiento de los algoritmos previamente a su implementación en una aplicación final.

Una descripción general de la operación y características que integran la herramienta es la siguiente, cuando el usuario ejecuta la aplicación de la **interfaz de usuario**, como primer acción, el puerto USB es inicializado y una prueba de conexión con el **sistema hardware para el procesamiento de imágenes** es realizada; si esta conexión es exitosa, el usuario podrá seleccionar uno de varios algoritmos implementados en el **sistema hardware para el procesamiento de imágenes**. Después de esto y conociendo el funcionamiento del algoritmo a aplicar, el usuario podrá abrir una imagen desde una ubicación específica, la cual se mostrará en pantalla. Luego podrá introducir los parámetros requeridos por el algoritmo (si los hubiera) y enseguida solicitar la ejecución del algoritmo presente en el sistema hardware. La **interfaz de usuario** desplegará los resultados obtenidos y el usuario podrá almacenar dichos resultados o aplicar algún otro algoritmo sobre ellos. En caso de que la prueba de conexión resulte fallida, el sistema se cerrará para evitar algún daño al hardware.

Por cada nuevo algoritmo que el usuario necesite evaluar, se tendrá que añadir a la **interfaz de usuario** una nueva ventana GUI para la presentación del mismo. Además, dentro de la **interfaz de usuario** se deberá asignar a dicha ventana GUI del algoritmo una trama única que será utilizada para la comunicación con el **sistema hardware para el procesamiento de imágenes**. Una metodología integral para llevar a cabo estas funciones, y algunas otras, es presentada en la sección 4.5.

El paradigma orientado a objetos es una manera nueva y diferente de pensar respecto a la programación. Una vez que se conoce que todo supone ser un objeto y se aprende a pensar más en el estilo orientado a objetos, se puede comenzar a crear buenos diseños que toman ventajas de todos los beneficios que ofrece la POO (Programación Orientada a Objetos) [14].

Un metodología, en el área de computación, es un conjunto de procesos y heurísticas usadas para romper la complejidad de un ploblema de programación [14]. Para el diseño

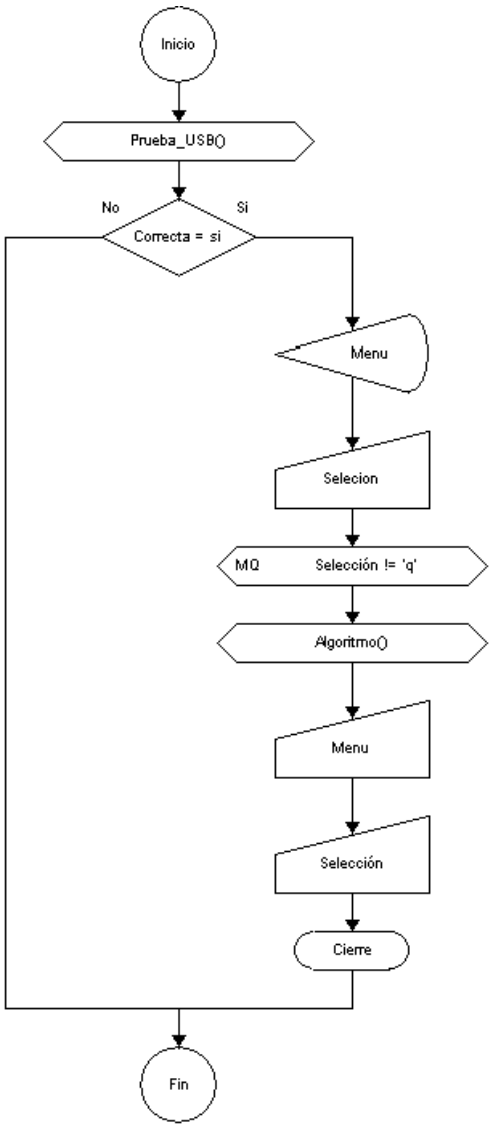


Figura 4.22: Flujo de tarea de la interfaz de usuario.

de la **intefaz de usuario** se ha elegido una metodología llamada **Bridge** [51], la cual es una metodología comprensible e integrada para diseños OO (Orientados a Objetos) y multi-plataforma de GUIs que resuelven necesidades de usuario. Las actividades más importantes de un método tipo Bridge se dividen en tres tareas: (1) expresar los requerimientos de usuario como un flujo de tarea, (2) llevar los flujos de tarea a objetos, y (3) mapear los objetos de tarea hacia objetos GUI.

Una vez que la definición de los objetos de tarea y las salidas de los flujos de tarea han sido establecidos, estos son llevados a un test de usabilidad por un miembro del equipo, y otro miembro verifica que todos los objetos, acompañados de atributos y acciones, se encuentren disponibles para desarrollar las tareas requeridas. Esto es realizado en una fase temprana y es independiente de cualquier representación GUI de los objetos. Los resultados de las actividades de diseño son primero escritas en tarjetas y puestas en una tabla, donde son fácilmente accesibles a todos los participantes y pueden ser alteradas o descartadas. Una vez que los objetos de tarea han sido mapeados hacia objetos GUI usando prototipos en papel, un test de usabilidad es desarrollado nuevamente por los miembros del equipo y se realiza el detallado final de especificaciones de diseño por un ingeniero de usabilidad de acuerdo a estilos de guía apropiados.

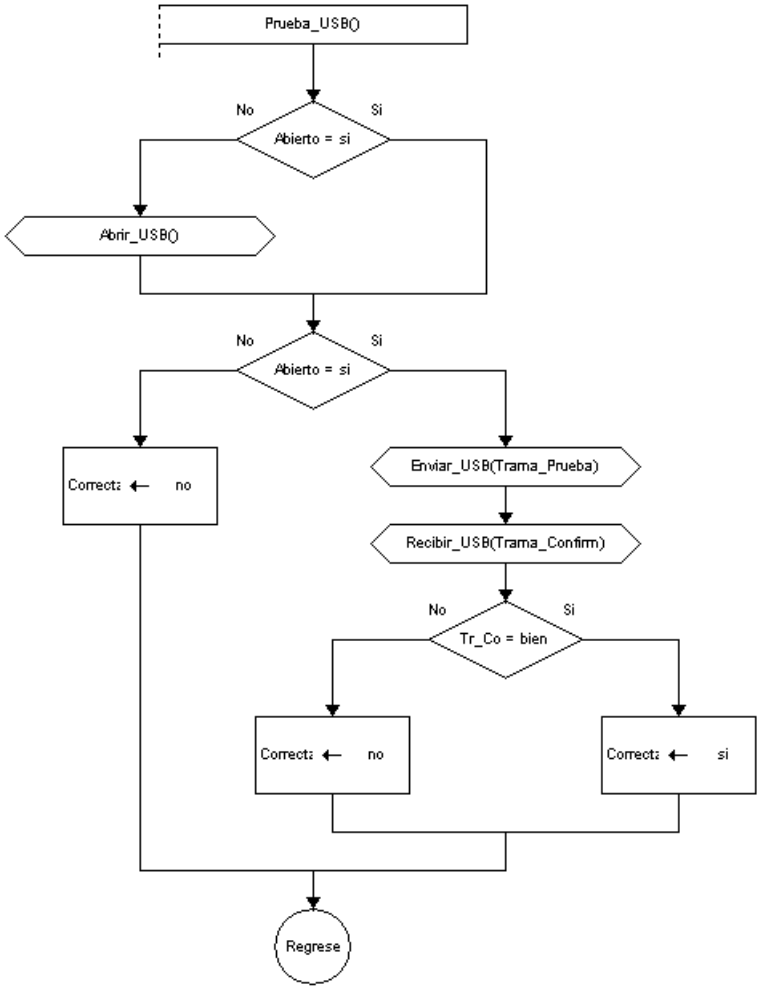


Figura 4.23: Flujo de tarea para la prueba de comunicación con el hardware.

4.4.1. Primera parte: Expresar los requerimientos de usuario como flujo de tarea

En la primera parte del diseño de la interfaz se expresan los requerimientos de usuario como flujos de tareas. Cuando la **interfaz de usuario** es ejecutada, realiza una prueba de comunicación. Si la prueba es correcta, presenta un menu de algoritmos de procesamiento y/o análisis implementados en el **sistema hardware para el procesamiento de imágenes** que el usuario podrá seleccionar; cuando el algoritmo termine su ejecución se presenta otra vez el menu para que el usuario pueda elegir otra opción. En caso en que la comunicación sea incorrecta, la **interfaz de usuario** se cerrará, esto para evitar posibles daños en el hardware, Figura 4.22.

El flujo de tarea de la prueba de comunicación inicia intentando abrir un puerto USB de la PC, Figura 4.23. Si el puerto USB se abre correctamente entonces se envía la "trama de prueba de comunicación", a través del USB, y se recibe la "trama de confirmación"; si esta última coincide con la de la Sección 4.4.4, la comunicación es correcta; en caso contrario, la comunicación no se estableció porque (1) el hardware esta dañado o (2) no es el adecuado.

Son dos flujos de tarea los que se tienen con un puerto USB de la PC, Figura 4.24. En ambos flujos de tarea, abrir y cerrar el puerto USB, se emplean las funciones propias del controlador USB DLP-USB245M.

Para abrir las imágenes a procesar, el envío de las mismas hacia el hardware y la obtención de resultados, se tienen tres flujos de tarea, Figura 4.25. En el envío de imágenes y la recepción de resultados se hace uso de las funciones propias del controlador de USB DLP-USB245M.

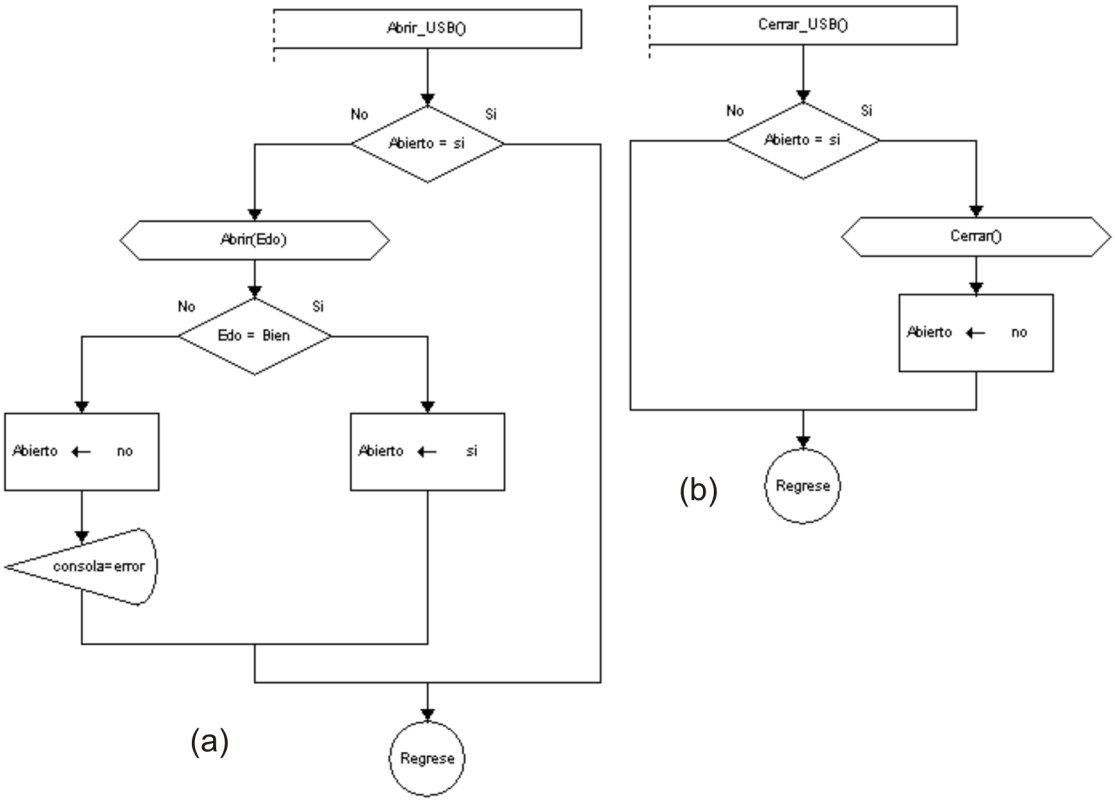


Figura 4.24: (a) Flujo de tarea para abrir un puerto USB de la PC. (b) Para cerrar el puerto USB abierto.

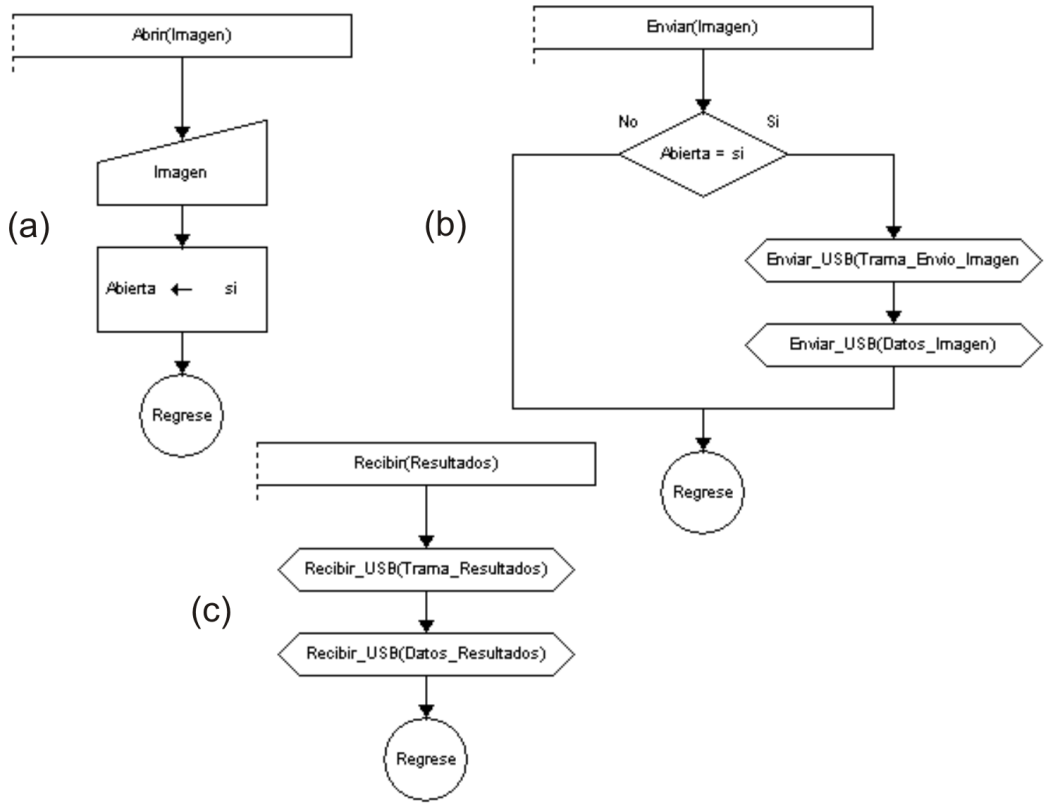


Figura 4.25: (a) Flujo de tarea para abrir imagen. (b) Flujo de tarea para enviar imágenes. (c) Flujo de tarea para recibir resultados.

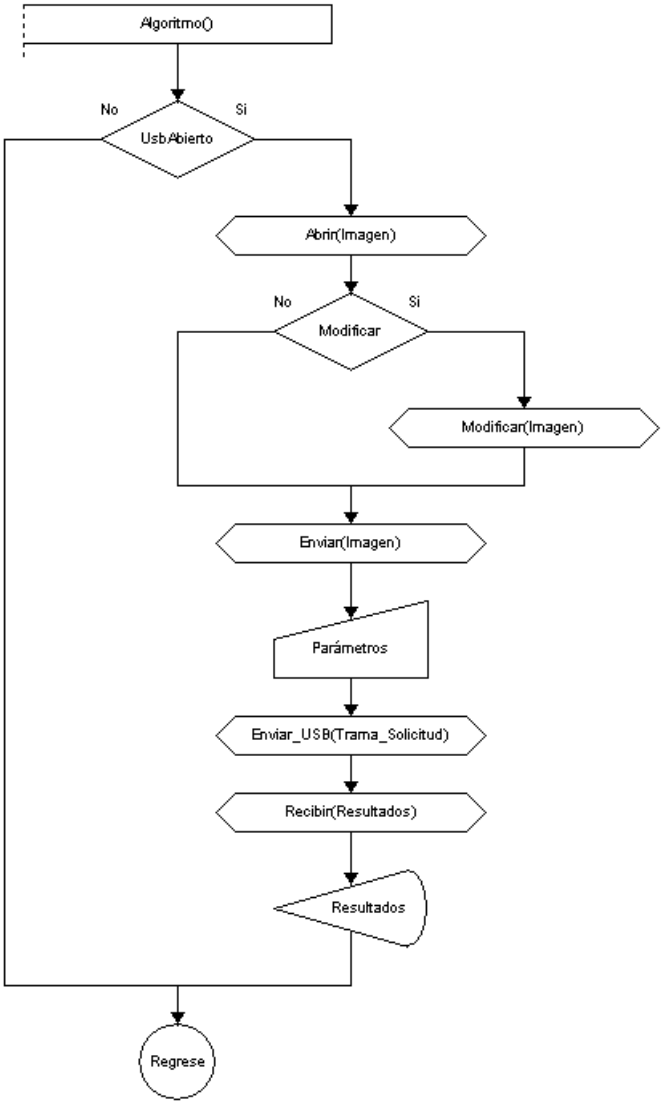


Figura 4.26: Flujo de tarea de un algoritmo de procesamiento y/o análisis de imágenes.

El flujo de tarea de un algoritmo de procesamiento y/o análisis de imágenes se muestra en la Figura 4.26. Se parte de checar si el USB está abierto, si es así, el algoritmo abre una imagen seleccionada por el usuario y se modifica si es necesario, luego se envía al **sistema hardware para el procesamiento de imágenes**. Para que el algoritmo se ejecute dentro del hardware, el usuario tiene que introducir los parámetros del algoritmo para que **la interfaz de usuario** genere la "trama de solicitud de algoritmo", Sección 4.4.4, y la envíe. Por último se reciben los resultados obtenidos por el hardware y se muestran al usuario.

4.4.2. Segunda parte: Mapear los flujos de tarea a objetos de tarea

En esta segunda parte se identificaron los objetos de tarea del diseño de la **interfaz de usuario** a partir de los flujos de tarea definidos en la primera parte. Los objetos de tarea con sus respectivos atributos y operaciones se muestran en la Figura 4.27. El objeto de tarea `Principal` está compuesto por un objeto de tarea `PuertoUsb`, y se encuentra asociado por un objeto `Imagen` y otro `Algoritmo`. Este último objeto está compuesto por objetos de tarea: `Imagen` y `Resultado`.

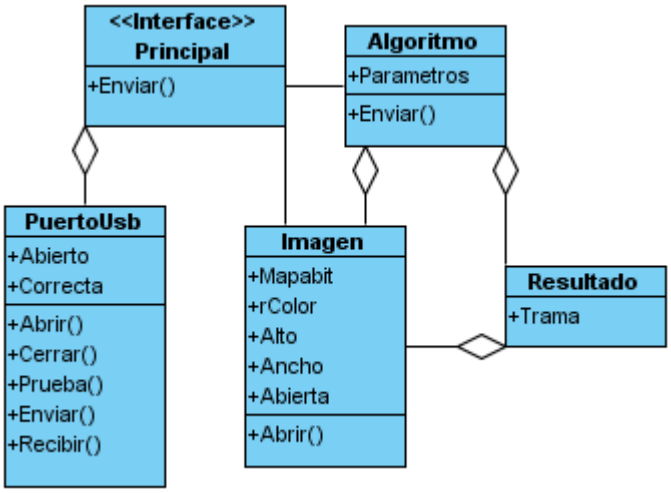


Figura 4.27: Objetos de tarea del diseño de la intefaz de usuario.

4.4.3. Tercera parte: Mapear los objetos de tarea a objetos GUI

Por último se debe llevar a cabo el mapeo de los objetos de tarea a objetos GUI, Figura 4.28. Los objetos GUI están compuestos por un conjunto de componentes visuales precompilados que son distribuidos en paquetes o bibliotecas, estos son: menús de aplicación, barra de herramientas, editor de texto, entrada de texto, imágenes, botones, paneles, etc. La **interfaz de usuario** tendrá un menu pricipal que se subdivide en (1) `Archivo`, el cual tiene acciones para cerrar la aplicación u otras modificaciones, (2) `Usb` con acciones para abrir el puerto, cerrarlo, y realizar una prueba de comunicación con el hardware, y (3) `Algoritmos`, que posee el conjunto de algoritmos de procesamiento y/o análisis implementados en el hardware y que el usuario podrá ejecutar. El diseño de cada algoritmo se realiza en otra ventana que será contenida en la **interfaz de usuario**, de esta manera la **interfaz de usuario** será una aplicación GUI de Documento de Interfaz Múltiple (MDI, *Multiple Document Interface*). Esta nueva ventana o interfaz de algoritmo contiene un menu para seleccionar las acciones que se realizarán a la imagen a procesar, como abrir, enviar, modificar, o guardar. La imagen es visualizada en el objeto GUI contenedor de imágenes `Imagen`, y los resultados en `Resultado`. Para que el usuario introduzca los parámetros del algoritmo se tienen una o varias entradas de texto para cada uno, dependiendo de las necesidades del algoritmo. Para finalizar se cuenta con un botón para la ejecución del algoritmo dentro del sistema hardware.

Teniendo el diseño de la **interfaz de usuario**, el paso siguiente es la codificación de la

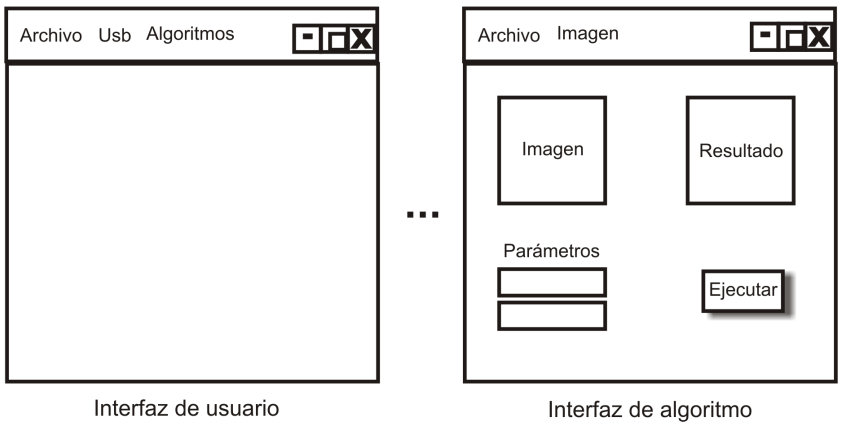


Figura 4.28: Objetos GUI.

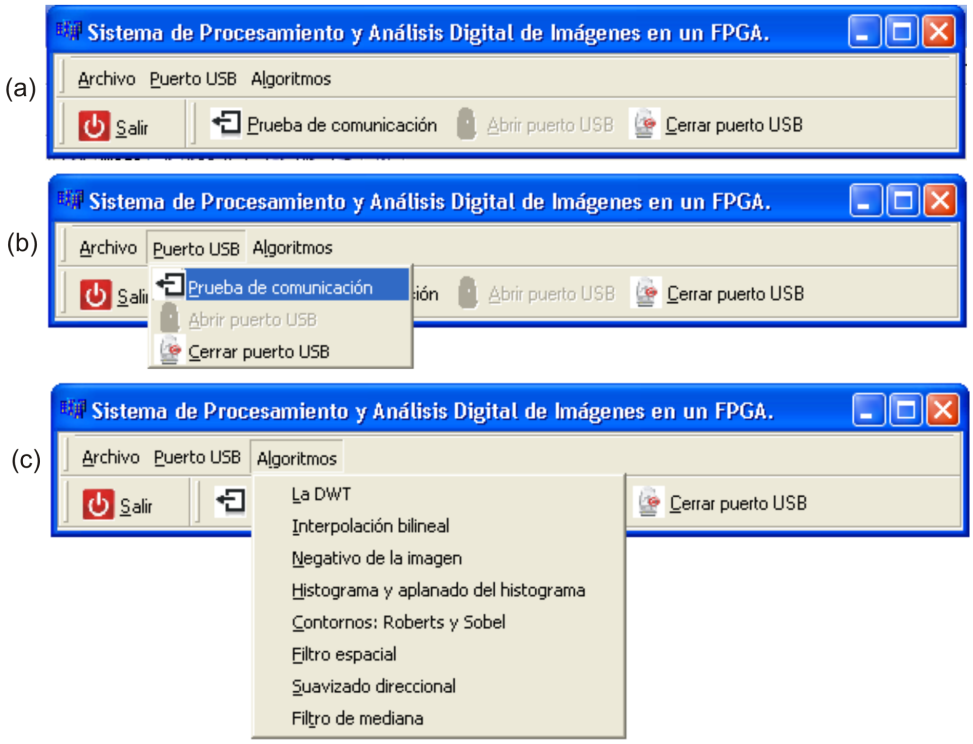


Figura 4.29: (a) Interfaz de usuario. (b) Interfaz de usuario realizando prueba de comunicación. (c) Seleccionando algoritmo.

aplicación. Con la ayuda de C++ Builder 6.0 se crearon los objetos de tarea (clases) de la segunda parte del diseño *Bridge*, también se emplearon los componentes visuales con los que cuenta esta herramienta para crear los objetos GUI de la tercera parte del diseño. La versión final de la **interfaz de usuario** se muestra en la Figura 4.29 a), en la Figura 4.29 b) se muestra la realización de una prueba de comunicación, y en la Figura 4.29 c) se muestra el menú de las interfaces de algoritmos implementadas. Se realizaron algunos pequeños cambios al diseño con la finalidad de mejorar la *usabilidad* de la aplicación, entre los cuales se puede mencionar la reducción del tamaño de la interfaz principal y la colocación, en un panel, del menú de opciones de los algoritmos.

4.4.4. Protocolo de comunicación

Cabe mencionar que la operación de este protocolo es transparente para el usuario. El protocolo consta de de 5 tramas de estructura fija y cada una de ellas está compuesta por varios campos donde cada uno de ellos cumple una función específica.

a) Trama de envío de imagen. Esta trama es utilizada por la **interfaz de usuario** para enviar la imagen que será procesada en el **sistema hardware para el procesamiento de imágenes**. La estructura de esta trama es mostrada en la Figura 4.30(a), como puede ser apreciado está constituida por cuatro campos: ID, CTR, DIR y DATOS. El campo ID contiene 8 bits, el primer nibble tiene una función común en todas las tramas y es servir como distintivo entre éstas, para la trama "envío de imagen" este nibble tiene un valor binario de "0001"; para esta trama, el nibble que complementa al campo ID no tiene función alguna. El campo CTR está formado por 32 bits, los cuales a su vez son divididos en AI y LI, cada uno formado por 10 bits y contienen el ancho y alto de la imagen a procesar y/o analizar respectivamente; los bits restantes de este campo son reservados para funciones futuras. El campo DATOS contiene la localidad

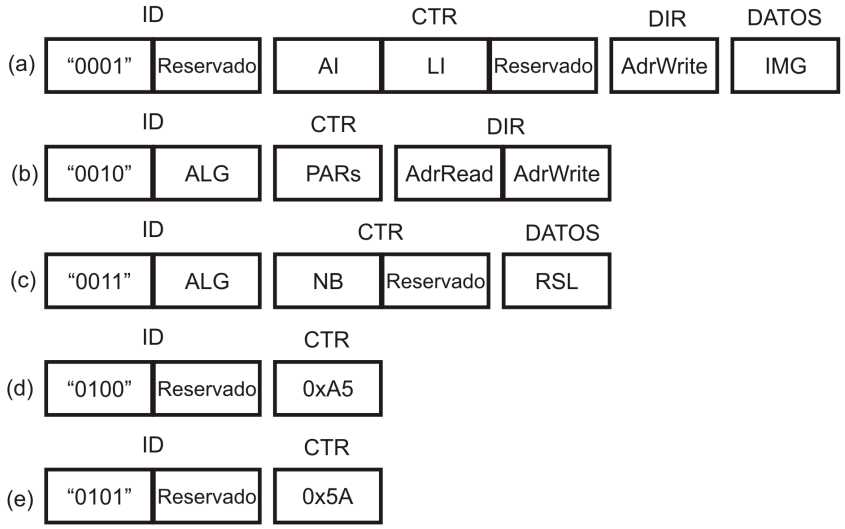


Figura 4.30: (a) Trama de envío de imagen. (b) Trama de solicitud de aplicación de algoritmo. (c) Trama de resultados. (d) Trama de prueba de comunicación. (e) Trama de confirmación.

de memoria donde se almacenará la imagen enviada y está compuesto por 24 bits. En el último campo de esta trama, DATOS, se encuentra la información de la imagen, siendo un campo de longitud variable definida por AI y LI.

b) Trama de solicitud de aplicación de algoritmo. La estructura de esta trama es diseñada para permitir a la **interfaz de usuario** indicarle al **sistema hardware para el procesamiento de imágenes** cual o cuales algoritmos deben ser aplicados sobre la imagen enviada. La estructura de esta trama, Figura 4.30(c), es compuesta por tres campos: ID, DIR y CTR. El primer nibble del campo ID tiene un valor fijo, “0010”, y su función es diferenciar esta trama de las demás; el nibble que complementa al campo ID, permite al **sistema hardware para el procesamiento de imágenes** determinar cual o cuales algoritmos deben ser aplicados a la imagen. CTR contiene 32 bits en los cuales se pueden especificar algunos parámetros requeridos por el algoritmo en evaluación y que determinaran la forma en que operará; estos parámetros dependen del algoritmo y son descritos en la especificación del algoritmo, Fase 1. Algunos bits de este campo tiene funciones reservadas para aplicaciones futuras. El campo que complementa la trama en estudio, DIR, se divide en dos grupos de 24 bits, en el primero se indica la localidad de memoria donde será leída la magen a procesar (AdrRead), y en el segundo la localidad donde se almacenarán los resultados (AdrWrite)

c) Trama de resultados. Mediante esta trama el **sistema hardware para el procesamiento de imágenes** podrá enviar a la **interfaz de usuario** los resultados obtenidos de aplicar el algoritmo a la imagen. Esta trama está diseñada con la finalidad de facilitar a la **interfaz de usuario** la interpretación de los resultados y puedan ser mostrados en una forma adecuada al usuario. Esta trama está formada por tres campos: ID, CTR y DATOS, Figura 4.30(b). El campo ID contiene 8 bits, el primer nibble tiene un valor fijo de “0011” y permite identificar a esta trama; el nibble que complementa al campo ID no tiene función alguna. La información que un algoritmo de análisis de imágenes genera no siempre es del mismo tamaño que el de la imagen en estudio; debido a esto, los 20 bits más significativos del campo CTR, etiquetados con NB, el número de bytes que son utilizados en la representación de los resultados; además, se han añadido 12 bits más al campo CTR que contemplan posibles expansiones futuras. En el último campo de esta trama, DATOS, se encuentran concentrados los resultados obtenidos de

aplicar el algoritmo sobre la imagen, éste es un campo de longitud variable definida por NB.

d) Trama de prueba de comunicación. Cuando la **interfaz de usuario** empieza su operación, inicializa al puerto USB de la PC y utiliza esta trama para solicitar al **sistema hardware para el procesamiento de imágenes** confirmar si la comunicación entre ambos ha sido establecida correctamente. Además, esta trama puede ser utilizada en cualquier momento para comprobar si la comunicación sigue establecida. Los campos ID y CTR constituyen la estructura de esta trama, Figura 4.30(d). El campo ID es formado por 8 bits, el primer nibble es el identificador de esta trama y tiene el valor constante "0100", el nibble que complementa a este campo no tiene función alguna. El campo CTR es de ocho bits y almacena el valor A5 Hex.

e) Trama de confirmación. Cuando el **sistema hardware para el procesamiento de imágenes** recibe una "trama de prueba de comunicación" que contiene un valor A5 Hex. en su campo CTR, genera como respuesta una "trama de confirmación". La estructura de esta trama, Figura 4.30(e), es compuesta por dos campos: ID y CTR. El primer nibble del campo ID tiene un valor fijo, "0101", y su función es diferenciar esta trama de las demás; el nibble que complementa a este campo no tiene función alguna. El campo que complementa la trama en estudio, CTR, esta formado por 8 bits y debe contener un valor 5A Hex.

4.5. Metodología para la integración de algoritmos

En las secciones anteriores de este capítulo, se describieron las tres primeras fases del diseño e implementación de una herramienta que permita visualizar el desempeño de un algoritmo de procesamiento y análisis de imágenes cuando opera sobre un FPGA. Estas fases iniciales forman un sistema hardware-software que incluyen, el modelado de un procesador de aplicación específica enfocado al procesamiento y análisis digital de imágenes (**procesador de imágenes**), el diseño e implemento de un sistema, utilizando la metodología de sistemas embebidos, con base en este procesador (**sistema hardware para el procesamiento de imágenes**) y la programación de una **interfaz de usuario**, que permite acceder las funciones del sistema diseñado, sobre una plataforma Windows utilizando la herramienta C++ Builder.

El **sistema hardware para el procesamiento de imágenes** es el encargado de aplicar el algoritmo de procesamiento y/o análisis sobre una imagen elegida por el usuario y la **interfaz de usuario** permite la selección de la imagen a procesar, el o los algoritmos que serán aplicados sobre la imagen y los parámetros necesarios para la operación de los mismos, así como permitir visualizar los resultados del procesamiento y/o análisis obtenidos por el algoritmo en estudio. Como complemento al diseño mencionado es necesario establecer una secuencia de procedimientos que faciliten la implementación y evaluación del desempeño de un algoritmo sobre el **sistema hardware para el procesamiento de imágenes**; en la cuarta fase se ha diseñado una **metodología para la integración de algoritmos** que cumple con el propósito mencionado; al integrar esta metodología al sistema desarrollado en las tres fases anteriores se obtiene una herramienta de diseño que permite visualizar los resultados de un algoritmo antes de ser implementado sobre una aplicación final, agilizando el ciclo de desarrollo de aplicaciones de procesamiento y/o análisis de imágenes sobre un FPGA. La Figura 4.31 muestra las diferentes fases que componen a esta metodología, mismas que son descritas el los párrafos siguientes.

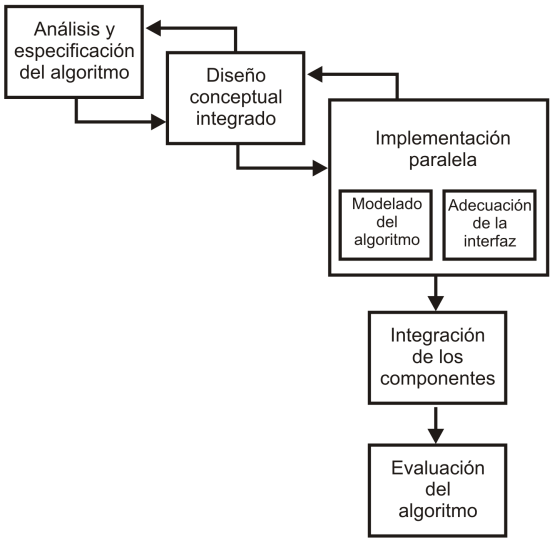


Figura 4.31: Diagrama de bloques de la metodología propuesta.

4.5.1. Fase 1: Análisis y especificación del algoritmo

Con la finalidad de optimizar la implementación de un algoritmo en el **sistema hardware para el procesamiento de imágenes**, en esta fase se deben definir en forma clara y precisa los aspectos relevantes de la función a desempeñar por el algoritmo de procesamiento y/o análisis de imágenes que va a ser añadido al sistema. Aspectos que deben ser definidos en esta fase son:

- Cuántos algoritmos serán evaluados.
- El tipo de imagen que será procesada.
- El tipo de datos resultantes del procesamiento (otra imagen) o análisis (datos).
- Segmentos de memoria adicionales a utilizar (si fuera necesario).
- Parámetros que modifican el flujo de operación del algoritmo en evaluación (por ejemplo, para una DCT se debe especificar en tamaño de bloque).
- Considerando que algunas funciones ya están establecidas por el sistema (estas son: las localidades de memoria a utilizar, los elementos que permiten establecer la comunicación entre la **interfaz de usuario** y el **procesador de imágenes**, los elementos que permiten acceder al sistema de memoria) el algoritmo debe ser adaptado a estas circunstancias.
- Se hará uso de algunas funciones definidas en la **interfaz de usuario** (Generación de ruido y evaluación objetiva de la calidad de una imagen).

4.5.2. Fase 2: Diseño conceptual integrado

Con base en la información generada por las especificaciones del algoritmo, en esta fase se debe establecer como serán diseñados y construidos los elementos que conformarán a la **interfaz de usuario** y al algoritmo que será integrado al **procesador de imágenes**; así como la forma en que interactuarán entre si.

Para diseño del algoritmo de procesamiento y/o análisis se debe establecer:

- La metodología a utilizar en la descripción del algoritmo.

- La herramienta CAD a utilizar, HDL, esquemático, editor de maquinas de estado o una combinación de éstas.
- Si se elige un HDL, es necesario determinar el tipo de descripción más adecuado para cada uno de los elementos que integran al algoritmo, flujo de datos, algorítmica o estructural.

Para el diseño de **la interfaz de usuario** es importante considerar:

- La forma en que la **interfaz de usuario** presentará los resultados.
- El tipo de datos a utilizar.

En esta fase, también deberán establecerse los requerimientos técnicos y funcionales para establecer una correcta comunicación entre ambos procesos. Estos requerimientos deben ser considerados en el modelado del nuevo algoritmo y en la adecuación de la **interfaz de usuario**.

- Identificador para cada algoritmo a implementar.
- Parámetros específicos de cada algoritmo.

4.5.3. Fase 3: Implementación paralela

En esta fase, con base en el diseño conceptual y los requerimientos para establecer una correcta comunicación entre los procesos, se desarrollarán en forma paralela la adecuación de la interfaz y el modelado del algoritmo de procesamiento y/o análisis.

a) *Modelado del algoritmo de procesamiento y/o análisis.* Esta subfase presenta los procedimientos requeridos para modelar y añadir un nuevo algoritmo al **procesador de imágenes**. En la consecución de estos procedimientos, además de tomar en cuenta los requerimientos mencionados en el párrafo anterior, es necesario considerar la estructura del procesador diseñado, Sección 4.2. Los procedimientos mencionados deben seguir las normas establecidas por la herramienta ISE Foundation y aunque han sido orientados a un diseño con un HDL pueden ser utilizados cuando se decide modelar al algoritmo con otra herramienta CAD.

   a. Agregar al proyecto un nuevo módulo HDL con el nombre del algoritmo de procesamiento y/o análisis a implementar.
   b. De acuerdo a la estructura del procesador de imágenes, la entidad de este módulo debe estar compuesta por las siguientes señales: reloj (Clk), reset (Rst), localidad de memoria de la imagen a procesar y/o analizar, localidad de memoria de la imagen o datos resultantes y las señales de control de acceso a la RAM (Data, Read, Write, ReadOk, WriteOk). Considerando el tipo de algoritmo a modelar, tanto en el **procesador de imágenes** como en el nuevo módulo creado, deben ser definidos los registros que contendrán los parámetros requeridos por este último y las señales de control activación y finalización. La función de cada una de estas señales ha sido detallada en la Sección 4.2.1.
   c. La arquitectura del módulo creado describe la estructura o el comportamiento del algoritmo a añadir y debe estar regido por los diagramas de tiempos de las señales definidas en el **procesador de imágenes** (Sección 4.2) y en su entidad.

d. Una vez concluida la descripción del algoritmo, éste debe ser agregado al **procesador de imágenes** a través de una sentencia de instanciación; en este proceso es importante establecer una correcta correspondencia entre las señales del **procesador de imágenes** y el algoritmo añadido.

b) *Adecuación de la interfaz.* Considerando que la **interfaz de usuario** ha sido ya diseñada e implementada (Sección 4.4); esta subfase únicamente contempla la forma de cómo adecuar esta interfaz al nuevo algoritmo o conjunto de algoritmos que serán agregados al sistema; es decir, sólo se encarga de agregar y adaptar nuevos elementos que permitan interpretar y visualizar los resultados generados por el nuevo algoritmo. Para cumplir con la función de esta fase, los siguientes procedimientos han sido establecidos, mismos que deben seguir las normas de la herramienta C++ Builder 6.0, ya que ésta fue utilizada en la programación de la **interfaz de usuario**.

Como se explicó en la Sección 4.4, una *interfaz de algoritmo* tiene como funciones: generar la "trama de solicitud de aplicación de algoritmo" y mostrar los datos resultantes del procesamiento. Tomando en cuenta estos aspectos, los procedimientos para adecuar la **interfaz de usuario** a un nuevo algoritmo son los siguientes:

a. Dar de alta al nuevo algoritmo en el menú principal de la **interfaz de usuario**.

b. Creación de una unidad nueva dentro del proyecto de la **interfaz de usuario** con el nombre del algoritmo de procesamiento y/o análisis a implementar.

c. Esta unidad debe ser construida de acuerdo a las especificaciones generadas en la Fase 1; debe contener los elementos que permitan al usuario elegir los parámetros que controlarán el funcionamiento del algoritmo a procesar, y en base a estos formar la "trama de solicitud de aplicación del algoritmo", y visualizar los resultados del procesamiento de la imagen.

d. De la misma forma debe implementarse el protocolo establecido en la Sección 4.4, entre la *interfaz de algoritmo* y **la interfaz de usuario**, que permitirá el intercambio de información entres ambas.

e. Si es necesario usar algunas funciones definidas en la **interfaz de usuario**, se hará siguiendo los procedimientos establecidos en la Sección 4.4.

4.5.4. Fase 4: Integración de los componentes

En esta fase la nueva versión de la **interfaz de usuario** y el **procesador de imágenes**, con el nuevo algoritmo añadido, son integrados.

Para llevar a cabo esta fase, se hace uso del protocolo de comunicaciones definido en la Sección 4.4.4, que rige el intercambio de información entre la **interfaz de usuario** y el **procesador de imágenes**.

4.5.5. Fase 5: Evaluación del algoritmo

La evaluación del comportamiento del algoritmo es la última fase de la metodología propuesta; esta evaluación consiste en determinar si el algoritmo implementado ha cumplido con los requerimientos esperados y si los resultados obtenidos concuerdan con la teoría. Dependiendo del algoritmo, la evaluación de los resultados puede hacerse mediante criterios objetivos o a simples valoraciones subjetivas (Sección 2.2). Si la evaluación indica que los resultados concuerdan con los esperados, el proceso se da por concluido.

De no coincidir los resultados con lo esperado, es posible regresar a cualquier fase de la metodología para determinar cual o cuales son las posibles causas y poder corregirlas. Algunas causas de esta diferencia pueden ser una mala interpretación en la teoría del algoritmo, errores en el protocolo de comunicaciones, un modelado erróneo del o de los algoritmos en estudio, mala interpretación de los resultados por parte de la **interfaz de usuario**, etc.

5. RESULTADOS

En este capítulo se presentan los resultados experimentales de utilizar la **herramienta para el modelado y evaluación de algoritmos de procesamiento y análisis de imágenes en un FPGA** en la valoración de algunos algoritmos específicos. También se muestra cómo estos algoritmos son integrados a dicha herramienta siguiendo la **metodología para la integración de algoritmos** propuesta. El conjunto formado por estos algoritmos, forma parte también de los resultados de este trabajo de tesis y representa una biblioteca inicial que forma parte de la herramienta propuesta.

5.1. Implementación del algoritmo de la DWT

La DWT es una transformada utilizada frecuentemente en el procesamiento digital de imágenes; dentro de esta área, la compresión de imágenes mediante la DWT es ampliamente reconocida, ejemplo de este hecho es que la DWT representa la base del estándar JPEG2000 [1]. La DWT subdivide a una imagen en cuatro regiones o sub-bandas: la región LL, donde se encuentra la mayor parte de la información de la imagen, conteniendo las frecuencias

bajas; las sub-bandas LH, HL y HH, contienen los detalles de la imagen, concentrando las frecuencias altas de la imagen. Mas detalles de esta transformada pueden ser encontrados en la Sección 2.3.3.3.

El resultado de aplicar la DWT de un nivel a una imagen, son cuatro sub-imágenes, una para cada sub-banda. Estas sub-imágenes son de una cuarta parte del tamaño de la imagen original y se puede decir que en la sub-banda LL se encuentra la imagen original en forma reducida. Dependiendo del número de niveles, la DWT va reduciendo a las sub-imágenes en un factor de cuatro.

A continuación se describe cómo este algoritmo es integrado a la **herramienta para el modelado y evaluación de algoritmos de procesamiento y análisis de imágenes en un FPGA** (capítulo 4) siguiendo la **metodología para la integración de algoritmos** (Sección 4.5), así como los resultados de la evaluación.

Fase 1. Análisis y especificación del algoritmo. Las especificaciones del algoritmo a implementar son las siguientes:

- Únicamente se evalúa un algoritmo, la DWT utilizado filtros Haar.
- Se procesarán imágenes en tonos de gris 8 bits/píxel.
- El resultado del procesamiento es una nueva imagen con el mismo número de elementos que la original.
- Considerando que la imagen a procesar se almacena en el segmento de memoria 0x000000-0x00FFFF, los coeficientes resultantes del procesamiento serán almacenados a partir de la localidad 0x010000.
- Parámetros a considerar en la DWT:
  - Número de niveles
- Señales que intervienen en la interfaz con el *módulo central de proceso* del **procesador de imágenes** (Sección 4.2.1):
  - DwtWork, señal de inicialización de la DWT.
  - DwtDone, señal de finalización del algoritmo de la DWT.
- Señales que intervienen en la interfaz con *módulo de control SRAM* del **procesador de imágenes** (Sección 4.2.3):
  - Data, bus de datos.
  - Addr, bus de direcciones.
  - Read, señal para la lectura.
  - Write, señal de escritura.
  - ReadOk, señal de lectura finalizada.
  - WriteOk, señal de escritura terminada.

Fase 2. Diseño conceptual integrado. En esta fase se desarrollan: los diseños conceptuales del algoritmo (dentro del **procesador de imágenes**) y de su correspondiente interfaz (dentro de la **interfaz de usuario**).

a) Diseño conceptual del algoritmo. Para la descripción de este algoritmo se utilizará la metodología descendente.

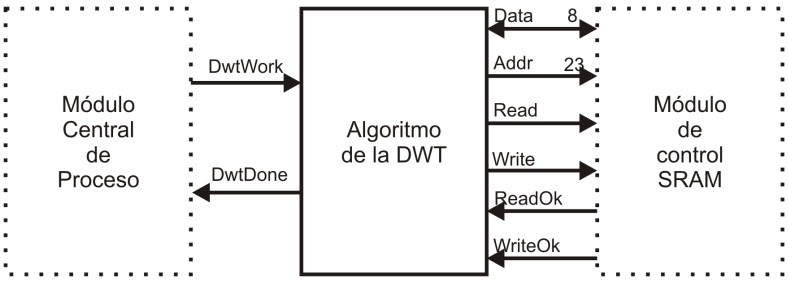


Figura 5.1: Diseño conceptual del algoritmo de la DWT.

Considerando que el modelado del **procesador de imágenes** fue hecho con la herramienta ISE Foundation ver 8.2i de la compañía Xilinx, en el modelado del algoritmo de la DWT también se utilizará esta herramienta. Además, se decide modelar al algoritmo mediante el lenguaje descriptor de hardware VHDL.

El modelado de un sistema digital empleando la metodología descendente empieza con la representación abstracta del sistema, en esta representación únicamente se definen sus señales de entrada y salida, Figura 5.1.

La Figura 5.1 también muestra la interacción que tiene el modulo en diseño (*Algoritmo de la DWT*) con el *módulo central de proceso* y con el *módulo de control SRAM* dentro del **procesador de imágenes**.

A partir de la representación abstracta del *Algoritmo de la DWT*, se definen los diferentes bloques que lo integrarán; el modelado conceptual del *algoritmo de la DWT* de un nivel se muestra en la Figura 5.2. Las entradas al módulo de la DWT son píxeles de la imagen a procesar y las salidas los coeficientes resultantes de la transformada. Este módulo cuenta con cuatro bloques internos a los cuales se conectan los píxeles de entrada, cada bloque forma un filtro Haar y genera píxeles de salida para formar una sub-banda: LL, LH, HL o HH. Cuando se aplica la DWT de más de un nivel, la nueva sub-imagen formada LL es puesta a la entrada del algoritmo de la DWT de un nivel, generándose así una reducción de la imagen original en factores de un cuarto.

Una vez delimitadas las primitivas a utilizar en la descripción del algoritmo y decidido la herramienta a emplear en el modelado del mismo, se determina que el tipo de descripción algorítmico o comportamental es el ideal para ser usado en el modelado del algoritmo de la DWT.

b) Diseño conceptual de la interfaz del algoritmo de la DWT. Para el diseño conceptual de la interfaz del algoritmo de la DWT, se emplea la metodología de diseño de aplicaciones GUI *Bridge*, que es la misma que se usó en el diseño de la **interfaz de usuario**. El paradigma de programación empleado para este diseño es la programación orientado a objetos.

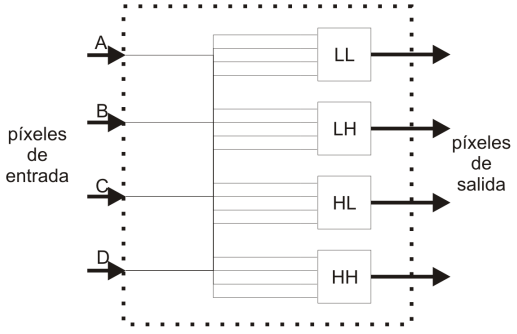


Figura 5.2: Modelado del algoritmo de la DWT de un nivel.

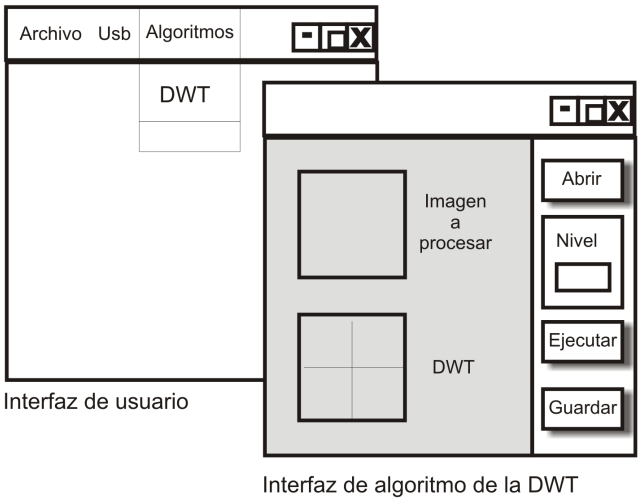


Figura 5.3: Diseño conceptual de la interfaz de algoritmo de la DWT.

Al aplicar la primera fase de la metodología Bridge, se determina que los flujos de tarea, requerimientos de usuario, de la interfaz de *algoritmo de la DWT* son los siguientes:

- Abrir puerto USB, Figura 4.24 (a), y cerrar puerto USB, Figura 4.24 (b).
- Prueba de comunicación con el hardware, Figura 4.23.
- Abrir, enviar y recibir archivo de imagen, Figura 4.25 (a), (b) y (c).

Cabe mencionar que estos flujos de tarea ya han sido diseñados e implementados y se encuentran integrados en la **interfaz de usuario**, al igual que el mapeo de los flujos de tarea a flujos de objeto, segunda fase de la metodología Bridge, Figura 4.27, y que únicamente son retomados.

En la Figura 5.3 se muestra la tercera fase de la metodología Bridge para el diseño de la interfaz de algoritmo de la DWT. Esta interfaz de algoritmo está compuesta por los siguientes componentes GUI:

- Botón GUI para la apertura de imagen.
- Botón GUI para el envío de imagen al hardware.
- Componente GUI de imagen para la imagen a procesar.
- Componente GUI de imagen para los resultados.
- Componente para guardar los resultados.

La interfaz de algoritmo de la DWT es una aplicación SDI (*Single Document Interface*) contenida en la **interfaz de usuario**, y para su ejecución tendrá que incluirse en el menú de esta última.

c) Definición de las tramas "envío de imagen", "solicitud de aplicación del algoritmo" y "resultados". Para establecer la correcta comunicación entre la **interfaz de usuario** y el **sistema hardware para el procesamiento de imágenes**, es necesario establecer las tramas de "envío de imagen", "solicitud de aplicación del algoritmo", la cual tiene por objetivo dar inicio a la ejecución del *algoritmo de la DWT* dentro del **procesador de imágenes**, y la trama de "resultados", la cual establece el tipo de resultados (datos), el número de bytes recibidos vía USB, etc.

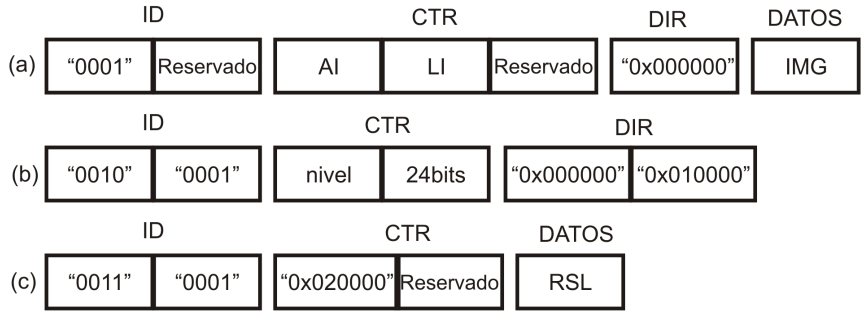


Figura 5.4: (a) Trama de envío de imagen de la DWT. (b) Trama de solicitud de aplicación de algoritmo de la DWT. (c) Trama de resultados de la DWT.

En la Figura 5.4 (a) se muestra la trama de “envío de imagen”para el *algoritmo de la DWT*, el alto y ancho de la imagen a enviar es variable (AI y LI) y determinan la cantidad de datos (IMG), además, la imagen a procesar se almacenará en memoria a partir de la localidad 0x000000. La trama de solicitud de aplicación utilizada por este algoritmo se muestra en la Figura 5.4 (b). De acuerdo al protocolo de comunicación, Sección 4.4.4, el primer nibble de esta trama indica que se trata de una trama de “solicitud de aplicación de algoritmo”, y el segundo nibble es el número de algoritmo; como parámetros se tiene al nivel de la DWT, y las localidades de escritura y lectura a la memoria concuerdan con las especificaciones, Fase 1.

La Figura 5.4 (c) muestra la estructura que la trama de resultados presentará para el *algoritmo de la DWT*, el número de datos obtenidos es 0x020000.

Fase 3. Implementación paralela. En esta fase se implementan en forma paralela el modelado del algoritmo de la DWT y la adecuación de la **interfaz de usuario**.

a) Modelado del algoritmo de la DWT. Para el modelado de este algoritmo se toma en cuenta su diseño conceptual, Fase 1 (a), y se siguen los siguientes procedimientos.

a) Se crea un nuevo módulo HDL dentro del proyecto del **procesador de imágenes** en la herramienta CAD ISE Foundation 8.2i, este módulo recibe el nombre de “DWT”.

b) La entidad del módulo DWT se forma a partir de las señales que intervienen tanto en el procesador como en la interfaz con la memoria, estas fueron mencionada en la Fase 1, *análisis y especificación del algoritmo*. Además, el módulo de la DWT requiere de otras dos señales: una señal de reloj para procesos síncronos (Clk), y la señal de reset (Rst) para reinicializar el módulo.

c) Basándose en el estilo de descripción de hardware para el diseño de circuitos digitales llamado comportamental, se realiza la arquitectura del módulo DWT. Esta arquitectura es dividida en cuatro bloques de menor jerarquía (LL, LH, HL, HH) que procesan los píxeles de entrada de una imagen por separado y generan los coeficientes de la DWT, Figura 5.2.

d) El bloque del **procesador de imágenes** que se encarga de la generación y decodificación de tramas, debe integrar un procedimiento que le permita reconocer la trama de solicitud de aplicación del *algoritmo de la DWT*. De la misma manera dicho bloque debe poder generar la trama de resultados y enviarla a la **interfaz de usuario**.

e) La instanciación del módulo DWT al **procesador de imágenes** es el último procedimiento de la implementación, con ello, el *algoritmo de la DWT* queda integrado al procesador, permitiendo, a partir de este momento, la evaluación del mismo.

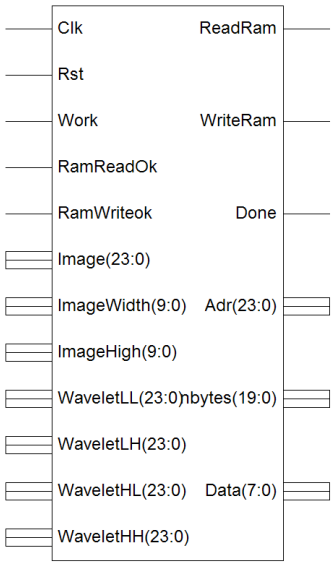


Figura 5.5: Símbolo del algoritmo de la DWT, generado a partir de su implementación.

En la Figura 5.5 se presenta el símbolo del *algoritmo de la DWT*, obtenido con la herramienta ISE Foundation ver. 8.2i.

b) Adecuación de la interfaz de usuario. La **interfaz de usuario** permite la visualización de los resultados producidos por el **procesador de imágenes**, los cuales son obtenidos a partir de la aplicación de la DWT a una imagen en tonos de gris. Para ello debe integrarse a la **interfaz de usuario** la unidad correspondiente al *algoritmo de la DWT*; los pasos para llevar a cabo esta tarea son los siguientes:

a) El primer paso es dar de alta al *algoritmo de la DWT* en el menú de algoritmos de la **interfaz de usuario**, para que al dar clic en el *item* de la DWT se abra la ventana de aplicación de este algoritmo.

b) Se crea una nueva unidad con el nombre "DWT"; dentro de la herramienta C++ Builder a esta unidad se le denomina "forma" y originalmente es creada sin objeto alguno.

c) Una vez creada la unidad que será utilizada para el algoritmo DWT y tomando en cuenta que la herramienta C++ Builder 6.0 permite el diseño de aplicaciones GUI de Documento de Interfaz Único (SDI, *Single Document Interface*), el usuario puede proceder a colocar los objetos GUI y objetos de tarea definidos para esta interfaz en la Fase 2 (b); el diseño o colocación de dichos objetos depende directamente de la aplicación.

d) Dentro de esta unidad, también debe ser considerada la generación de la trama de "solicitud de aplicación del algoritmo", formada en la Fase 2 (c) considerando como parámetro el nivel deseado para la DWT.

e) Por último, cuando la interfaz de la DWT reciba los resultados provenientes del **sistema hardware para el procesamiento de imágenes** debe interpretarlos y mostrarlos al usuario.

La Figura 5.6 muestra la ventana diseñada para el *algoritmo de la DWT*.

Fase 4. Integración de los componentes. Una vez que la **interfaz de usuario** ha sido modificada y adecuada para la visualización de los resultados de la DWT en una

Figura 5.6: Interfaz de algoritmo para la DWT.

nueva ventana y se tiene una nueva versión del **procesador de imágenes**, debido a la implementación del *algoritmo de la DWT* y la reprogramación del FPGA, ambos componentes son integrados al **sistema hardware para el procesamiento de imágenes** para su evaluación.

Fase 5. Evaluación del algoritmo de la DWT. La evaluación de los resultados de la aplicación de la DWT a la imagen de Lena en tonos de gris concuerda con la teoría. Cuando el algoritmo implementado en el **procesador de imágenes** es aplicado sobre la imagen, se generan cuatro sub-imágenes que son las bandas LL, LH, HL y HH, como se muestra en la Figura 5.6.

5.2. Implementación de los algoritmos de la DWT y la interpolación bilineal

La herramienta de procesamiento y análisis de imágenes permite la implementación de varios algoritmos a la vez y la interacción entre ellos. En este ejemplo se implementa el algoritmo de la DWT, realizado en el ejemplo anterior, y la interpolación bilineal.

Una descripción de la técnica de la DWT fue descrita en el ejemplo anterior. Por otro lado, la interpolación bilineal es una técnica de procesamiento de imágenes que amplía o reduce el tamaño de las mimas, Sección 2.3.1.1.7. A diferencia de otros métodos de modificación del tamaño de la imagen, la interpolación bilineal establece la modificación en cualquier porcentaje. A partir de los porcentajes de cambio de la imagen, de altura y de anchura, se recorre píxel por píxel y se calculan las nuevas posiciones de cada uno de ellos en la imagen resultante mediante un escalamiento; a partir de la posición del píxel original se calculan incrementos y contribuciones para formar el valor y posición del nuevo píxel, de esta forma se genera la imagen modificada en tamaño, mediante un escalado isométrico, donde se guardan las proporciones verticales y horizontales de la imagen original.

Fase 1. Análisis y especificación del algoritmo. Las especificaciones de ambos algoritmos se describen por separado aunque existan algunas dependencias entre ellos. No será necesario expresar los requerimientos del *algoritmo de la DWT* debido a que ya fueron descritos en el ejemplo anterior. A continuación se presentan las especificaciones del *algoritmo*

de interpolación bilineal.

- Se evalúan dos algoritmos: la DWT utilizando filtros Harr y la interpolación bilineal.
- Se procesarán imágenes en tonos de gris 8bits/píxel.
- La imagen de entrada para la interpolación bilineal puede ser proporcionada por el usuario u obtenida de los resultados del algoritmo de la DWT.
- La interpolación bilineal se utilizara para a partir de los coeficientes generados por el filtro LL de la DWT, reconstruir la imagen original.
- Parámetros a considerar en la interpolación bilineal:
  - Origen de la imagen de entrada.
  - Porcentaje de ampliación/reducción.
- Señales que intervienen en la interfaz con el procesador de imágenes:
  - BilinealWork, para la inicialización de la interpolación bilineal.
  - BilinealDone, indica la finalización de la interpolación bilineal.
- Señales que intervienen en la interfaz con la memoria:
  - Data, bus de datos.
  - Addr, bus de direcciones.
  - Read, señal para la lectura.
  - Write, señal de escritura.
  - ReadOk, señal de lectura finalizada.
  - WriteOk, señal de escritura terminada.
- Se hará uso del PSNR, criterio de evaluación de la calidad de la imagen procesada.

Fase 2. Diseño conceptual integrado. En esta fase se desarrolla el diseño conceptual del *algoritmo de interpolación bilineal* y el diseño conceptual para su interfaz con el usuario. Para el caso de la DWT se retoma el diseño conceptual del ejemplo anterior.

a) Diseño conceptual del algoritmo de interpolación bilineal. Para el desarrollo del *algoritmo de interpolación bilineal* se emplea la metodología descendente, partiendo de un alto nivel de abstracción y dividiendo al algoritmo en bloques de menor jerarquía, los cuales realizan el procesamiento de la imagen apoyándose en las primitivas de la herramienta.

Para la descripción de este algoritmo se emplea la herramienta ISE Foundation 8.2i y el lenguaje descriptor de hardware estándar VHDL. En el modelado del mismo se determina que el tipo de descripción para este sistema digital es el algorítmico o comportamental.

Básicamente las entradas y salidas del módulo del *algoritmo de interpolación bilineal* están conformadas por aquellas señales que intervienen en la interfaz con el *módulo de control SRAM* y las que intervienen en la interfaz con el *módulo central de proceso*. En la Figura 5.7 se muestra el diseño conceptual para el algoritmo de interpolación bilineal, considerando a este como una entidad con señales de entrada/salida que se conecta con otros módulos.

Siguiendo la metodología descendente, el *algoritmo de interpolación bilineal* es dividido en dos sub-modulos: “contribuciones por cercanía” y “obtención del nuevo píxel” (Figura

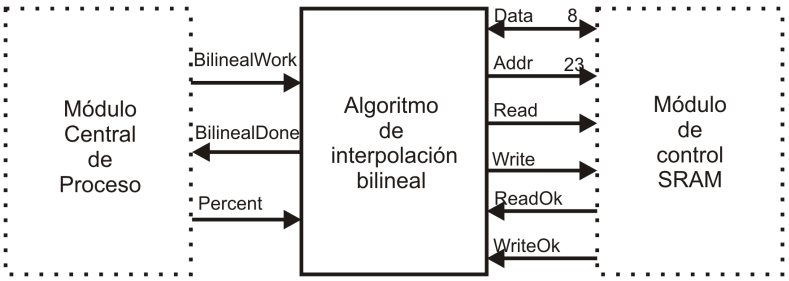


Figura 5.7: Diseño conceptual del algoritmo de interpolación bilineal.

5.8); ambos modelados utilizando descripción comportamental. Este diseño conceptual muestra de manera general la manera en que será implementada la arquitectura del *algoritmo de interpolación bilineal*; la cual está compuesta por bloques funcionales trabajando en forma conjunta para ampliar o reducir la imagen de entrada.

b) Diseño conceptual de la interfaz de algoritmo de la interpolación bilineal. La interfaz de algoritmo para la interpolación bilineal tiene que proveer al usuario la manera de interactuar y facilitar el entendimiento del mismo, para ello se crea un diseño conceptual inicial basándose en la metodología Bridge para el diseño de aplicaciones GUI. Además, siguiendo esta metodología se emplea el paradigma de programación orientado a objetos.

En la primera fase de la metodología Bridge se expresan los requerimientos de usuario como flujos de tarea. Los flujos de tarea para la interfaz del *algoritmo de la interpolación bilineal* son los siguientes:

- Abrir puerto USB, Figura 4.24 (a), y cerrar puerto USB, Figura 4.24 (b).
- Prueba de comunicación con el hardware, Figura 4.23.
- Abrir, enviar y recibir archivo de imagen, Figura 4.25 (a), (b) y (c).

Los flujos de tarea son retomados de la **interfaz de usuario**, no es necesaria su implementación ya que son funciones propias de la **interfaz de usuario** y pueden ser empleadas por las interfaces de algoritmos implementadas.

En la segunda fase de la metodología se expresan (mapean) los flujos de tarea a objetos de tarea. Estos objetos de tarea, también retomados de la **interfaz de usuario**, están compuestos por atributos y métodos como se muestran en la Figura 4.27.

El mapeo de los objetos de tarea a objetos GUI para este diseño forma parte de la tercera fase de la metodología. Los objetos GUI generados para esta interfaz son los siguientes:

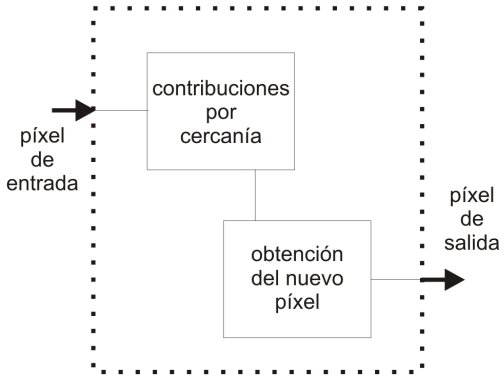


Figura 5.8: Diseño conceptual para el modelado del algoritmo de interpolación bilineal.

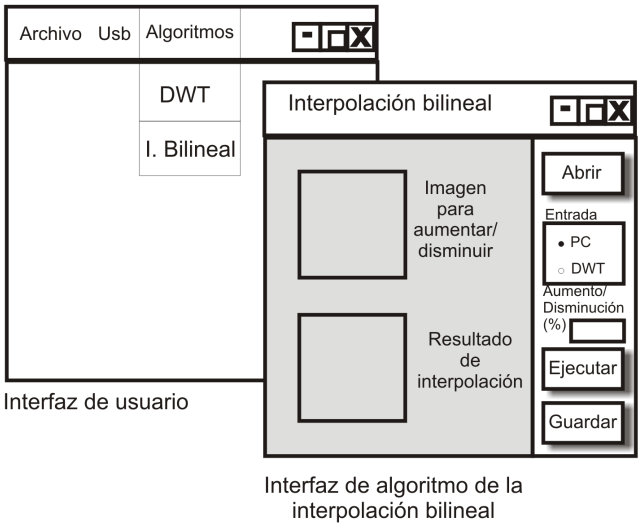


Figura 5.9: Diseño conceptual de la interfaz de algoritmo de al interpolación bilineal.

- Botón GUI para la apertura de imagen.
- Botón GUI para el envío de imagen al hardware.
- Componente GUI de imagen para la imagen a procesar.
- Componente GUI de imagen para los resultados.
- Objeto GUI para selección de la entrada de la imagen (usuario o *algoritmo de la DWT*).
- Objeto GUI de entrada para porcentaje de ampliación/reducción.
- Componente para guardar los resultados.

En el menú de la **interfaz de usuario** se agrega un ítem para el *algoritmo de la interpolación bilineal*. Se toma en cuenta que este algoritmo es de tipo SDI y cuando se ejecuta por el usuario su ventana aparece sobre la ventana de la **interfaz de usuario**. En la Figura 5.9 se muestra el diseño conceptual de la interfaz del *algoritmo de interpolación bilineal*, el cual cumple con las especificaciones del mismo, Fase 1.

c) Definición de las tramas "envío de imagen", "solicitud de aplicación del algoritmo" y "resultados". La comunicación entre la interfaz del *algoritmo de interpolación bilineal* y el **sistema hardware para el procesamiento de imágenes** se lleva a cabo mediante el uso de un protocolo que hace uso de tramas de comunicación definidas (Sección 4.4.4); las tramas que son necesario definir para la implementación de este algoritmo son: la trama de "envio de imagen", la trama de "solicitud de aplicación del algoritmo" y la trama de "resultados".

La trama de "envió de imagen" para este algoritmo es ocupada únicamente cuando el usuario elige que la imagen de entrada sea leída de la PC, Figura 5.10 (a). La trama de "solicitud de aplicación de algoritmo" para la interpolación bilineal, de acuerdo a la Sección 4.4.4, es la que se muestra en la Figura 5.10 (b). Existen dos parámetros que varían de acuerdo al usuario. Por un lado la opción para la imagen de entrada: proporcionada por el usuario u obtenida de los resultados de la DWT. Y por otra parte el parámetro para el porcentaje de ampliación/reducción de la imagen, introducido por el usuario.

La trama de "resultados" para la interpolación bilineal se muestra en la Figura 5.10 (c). En ella, están presentes los datos necesarios para que el **procesador de imágenes** recupere los resultados del *algoritmo de interpolación bilineal* sobre la imagen indicada por el usuario.

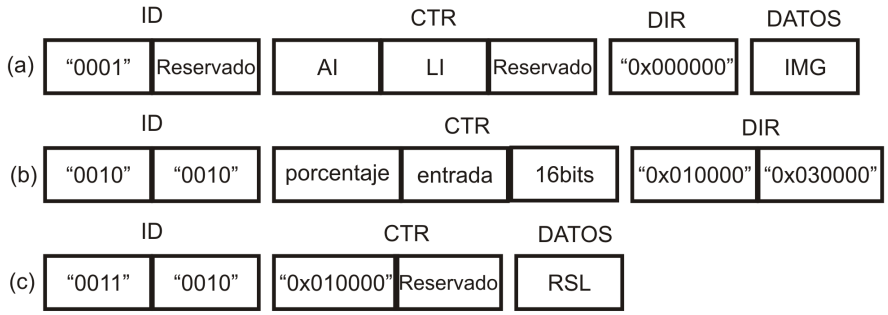


Figura 5.10: (a) Trama de envio de imagen del algoritmo de interpolación bilineal. (b) Trama de solicitud de aplicación de algoritmo. (c) Trama de resultados.

Fase 3. Implementación paralela. En esta fase, se implementan en forma paralela: el modelado del *algoritmo de interpolación bilineal* con base en el diseño conceptual del mismo, Fase 2 (a), y la adecuación de la **interfaz de usuario** basándose en el diseño conceptual de la interfaz de *algoritmo de interpolación bilineal*, Fase 2 (b).

a) Modelado del algoritmo de interpolación bilineal. De acuerdo con el diseño conceptual de este algoritmo, Fase 2 (a), y con la herramienta ISE Foundation 8.2i se tienen los siguientes procedimientos para la implementación:

a) Creación de un nuevo módulo HDL dentro del proyecto del **procesador de imágenes** en la herramienta CAD ISE Foundation 8.2i, con el nombre de “Bilineal”.

b) La entidad de este módulo se compone de las señales que intervienen con el **procesador de imágenes** y las señales requeridas para la interfaz con la memoria, mencionadas en el análisis y especificación del algoritmo, además, posee otras dos señales indispensables: señal de reloj (Clk) y señal de reset (Reset).

c) La arquitectura de este módulo, basada en los diagramas de tiempos de accesos a memoria, es implementada con ayuda de procesos síncronos en VHDL (estilo comportamental).

d) El bloque del **procesador de imágenes** que se encarga de la generación y decodificación de tramas, debe integrar un procedimiento que le permita reconocer la trama de solicitud de aplicación del *algoritmo de interpolación bilineal*. De la misma manera dicho bloque debe poder generar la trama de resultados y enviarla a la **interfaz de usuario**.

e) Considerando el diseño del *algoritmo de interpolación bilineal* descrito en este módulo, se prosigue a su instancición por parte del **procesador de imágenes**. Con esto el procesador podrá inicializar al *algoritmo de interpolación bilineal* de acuerdo a las peticiones en la **interfaz de usuario**.

En la Figura 5.11 se presenta el símbolo del *algoritmo de la interpolación bilineal*, obtenido con la herramienta ISE Foundation ver. 8.2i.

b) Adecuación de la interfaz de usuario. Para la adecuación de la **interfaz de usuario**, y que ésta pueda ejecutar el *algoritmo de interpolación bilineal* implementado en el hardware, y visualizar sus resultados, se siguen los siguientes procedimientos:

a) Se da de alta al *algoritmo de interpolación bilineal* dentro del menú de algoritmos implementados en la **interfaz de usuario**, así la próxima vez que el usuario de clic en ese ítem se abrirá la interfaz de *algoritmo de la interpolación bilineal* para poder trabajar en ella.

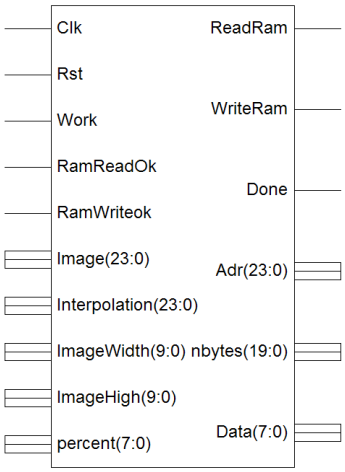


Figura 5.11: Símbolo del algoritmo de interpolación bilineal generado a partir de su implementación.

Figura 5.12: Interfaz de algoritmo de la interpolación bilineal.

b) Una nueva unidad con el nombre de "Bilineal" es creada dentro del proyecto de la **interfaz de usuario**. Esta unidad contendrá en conjunto de objetos de tarea y objetos GUI para dar la funcionalidad a la interfaz de *algoritmo de la interpolación bilineal*.

c) Con base en el diseño conceptual de esta interfaz de algoritmo, Fase 2 (b), se conforma el formulario que contiene los componentes visuales de acuerdo a la Figura 5.9. Las acciones necesarias sobre estos objetos GUI se realizan sobre la unidad "Bilineal". La trama de "solicitud de aplicación de algoritmo" se genera y envía cada vez que se ejecuta la interfaz de *algoritmo de interpolación bilineal*; algunos nibbles de la trama se mantienen fijos y otros cambian de acuerdo a los requerimientos del usuario, como son el porcentaje de ampliación/reducción y la elección de la imagen de entrada.

d) Por último, cuando la interfaz de la interpolación bilineal reciba los resultados provenientes del **sistema hardware para el procesamiento de imágenes** debe interpretarlos y mostrarlos al usuario.

e) El PNSR se calcula y se muestra para que el usuario pueda valorar los resultados.

Siguiendo el diseño conceptual del diseño de la interfaz de *algoritmo de la interpolación bilineal*, Fase 2 (b), se codifica e implementa esta interfaz dentro del proyecto de la **interfaz de usuario**. El resultado es la aplicación GUI de tipo SDI que se muestra en la Figura 5.12.

Fase 4. Integración de los componentes. Para la integración de ambos componentes primero se llevó a cabo la reprogramación del FPGA que contiene al **procesador de imágenes**; esta nueva versión del procesador ahora contiene al *algoritmo de interpolación bilineal* pudiendo trabajar en forma conjunta con el *algoritmo de la DWT*, previamente implementado. Con respecto a la **interfaz de usuario**, después de modificada y adecuada, fue compilada con la herramienta C++ Builder 6.0 generando un archivo ejecutable nuevo que contiene otra interfaz visual, la del *algoritmo de la interpolación bilineal*. Después de integrados ambos componentes ahora pueden ser evaluados.

Fase 5. Evaluación del algoritmo de interpolación bilineal. En la evaluación del *algoritmo de interpolación bilineal* se presentan los siguientes resultados. La imagen Elaine, 256×256 píxeles, 8 bits/píxel, fue reducida al 50 % empleando el *algoritmo de interpolación bilineal*. En la Figura 5.12 se muestran estos resultados, los cuales concuerdan con la teoría, Sección 2.3.1.1.7.

Además de la implementación de los algoritmos descritos en las secciones anteriores, los siguientes algoritmos de procesamiento y análisis de imágenes fueron integrados al **procesador de imágenes**, el diseño de los mismos fue realizado siguiendo la metodología para la integración de algoritmos, Sección 4.5.

5.3. Algoritmo del negativo de la imagen

El negativo de la imagen es una técnica de procesamiento que pertenece a la mejora de la imagen, Sección 2.3.1.1.1. Se trata de modificar una imagen de tonos de gris que contiene regiones claras de interés sobre regiones oscuras dominantes. Es una de las técnicas de mejora más sencilla. De acuerdo con la Ecuación 2.9, el procesamiento llevado a cabo para obtener el negativo de una imagen se realiza píxel por píxel; el valor del píxel nuevo s se obtiene al restar el píxel de la imagen original r del total de niveles de gris de la misma $L-1$, y no existe dependencia entre píxeles cuando se realiza el cálculo, sino únicamente aquellos que conforman regiones o formas en la imagen original y que después del procesamiento, estas regiones se invierten resaltando las partes de interés.

La Figura 5.13 muestra el símbolo resultante del *algoritmo del negativo de la imagen*. En la Figura 5.14 se tiene a la interfaz de este algoritmo con los resultados obtenidos de aplicar el negativo a la imagen de Baboon, 256×256 píxeles, 8 bits/píxel.

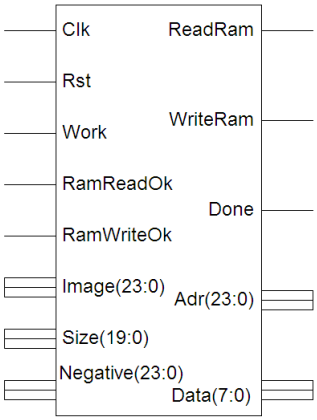


Figura 5.13: Símbolo de la implementación del negativo de la imagen.

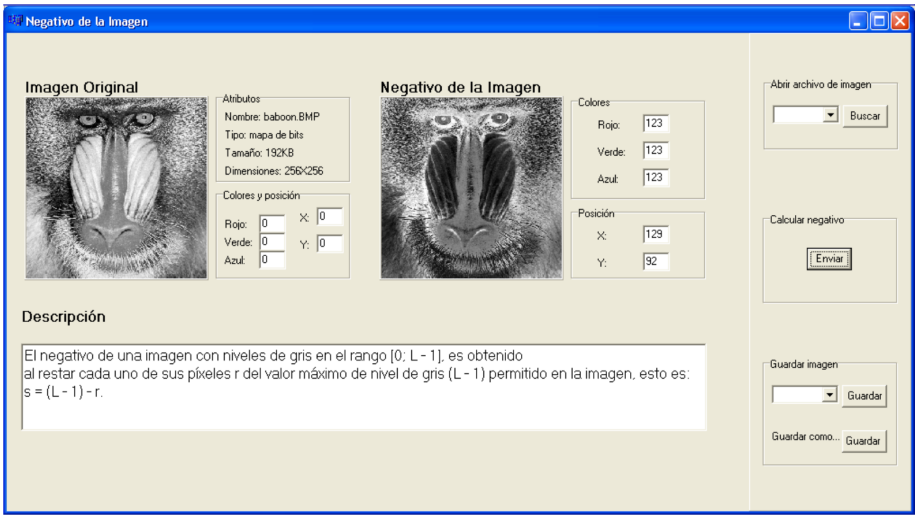


Figura 5.14: Interfaz del algoritmo del negativo de la imagen y resultados del negativo de Baboon.

5.4. Algoritmos de histograma de la imagen y ecualización del histograma

El histograma de la imagen se representa con una gráfica de dos variables y refleja el número de ocurrencias de píxeles que contienen cierto nivel de gris en una imagen, desde el nivel más oscuro hasta el más claro; en el eje horizontal se encuentran los tonos de gris, y en el vertical el número de repeticiones de dicho nivel; para imágenes muy oscuras, el cálculo de su histograma encontrará que el rango dinámico de la imagen será estrecho y recorrido a la izquierda, hacia tonos de gris oscuros; en el caso de imágenes claras, su histograma será estrecho y desplazado hacia la derecha, Sección 2.3.1.1.2. La razón de ser del histograma es debido a que suele ser empleado por otras técnica de procesamiento y/o análisis de imágenes más complejas que realizan modificaciones a la imagen, a diferencia del histograma, que se trata de una técnica puramente estadística y que como resultado arroja características que predominan en la imagen, como son brillo, contraste, etc. El procesamiento del histograma se realiza contando las coincidencias de píxeles de cada nivel de gris en un recorrido por toda la imagen. Se trata de un proceso irreversible; a partir de la imagen se calcula su histograma,

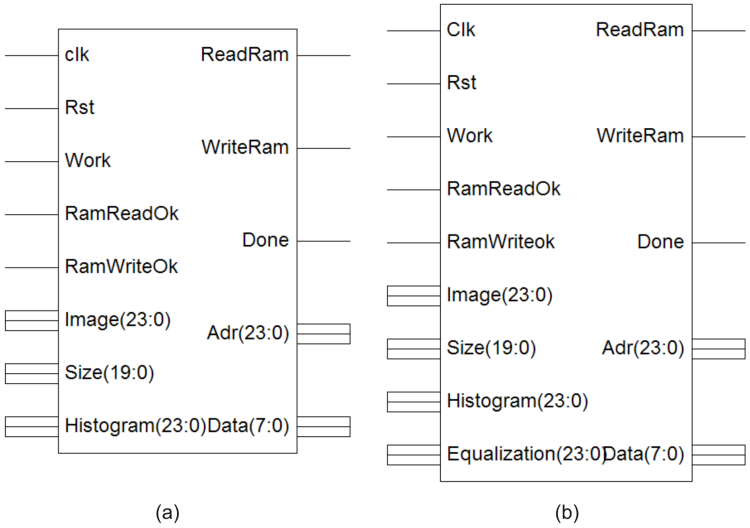


Figura 5.15: Símbolos generados de la implementación. (a) Algoritmo del histograma de la imagen. (b) Algoritmo de la ecualización del histograma.

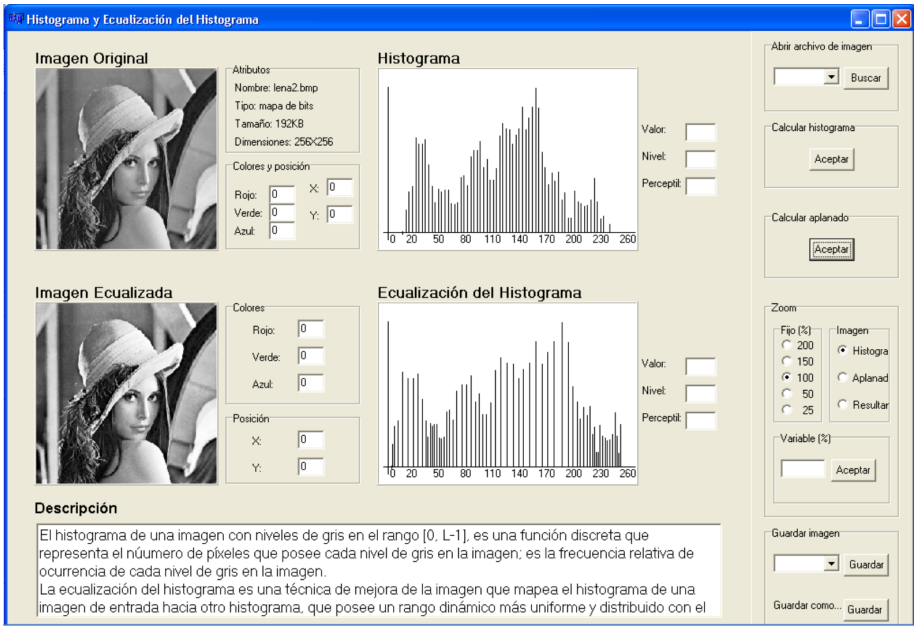


Figura 5.16: Interfaz de los algoritmos del histograma y ecualización del histograma, y resultados obtenidos para Lena.

pero teniendo el histograma de una imagen no se puede obtener dicha imagen aplicando un proceso inverso, puesto que las posiciones que ocupa cada píxel de cierto tono de gris se pierden en el cálculo del histograma; dos imágenes pueden tener el mismo histograma, pero dos histogramas iguales pueden generar imágenes diferentes.

El aplanado del histograma, también conocido como ecualización del histograma, es una técnica de mejora de contraste para imágenes con tonos de gris, Sección 2.3.1.1.3. Como su nombre lo indica, hace uso del histograma de una imagen para obtener una imagen resultante con mejor contraste o mejor discernimiento entre las distintas zonas o formas que conforman la misma. Cuando el histograma de una imagen es estrecho, el aplanado busca expandirlo en todo el rango dinámico y normalizarlo para que los píxeles se distribuyan en la imagen de forma equitativa, con esto, la imagen resultante posee mayor contraste; además puede ser empleada en otros algoritmos, afectando de manera positiva en la mejora de la visión de la misma, consiguiendo una mejor interpretación.

El aplanado del histograma es otro histograma generado a partir del histograma de la imagen que se quiere mejorar en contraste. Este histograma resultante se obtiene a partir de la Ecuación 2.11, donde la entrada al proceso son los valores del histograma original y los índices de nivel de gris donde se encuentran; a la salida se obtienen nuevos niveles de gris o índices v_k donde deberán aparecer las ocurrencias de píxeles del nivel de gris anterior a procesamiento r_k, es decir, los valores del histograma original se desplazan hacia nuevos niveles de gris para expandir el histograma en todo el rango dinámico, pudiéndose dar el caso que varias cantidades del histograma original se desplacen hacia el mismo nivel de gris r_k; el resultado de este proceso es una uniformidad del histograma original en cuanto a amplitud. Esta técnica hace uso de la probabilidad acumulada S_k de los niveles de gris r_k, que no es más que la suma acumulativa de los valores del histograma original en el k-ésismo nivel de gris. Después de obtener el aplanado del histograma, se procede a obtener la imagen resultante mejorada en contraste basándose en las posiciones de los píxeles de la imagen original.

La Figura 5.15 muestra los símbolos resultantes de los algoritmos de histograma de la imagen y ecualización de la misma; la Figura 5.16 muestra la interfaz de estos algoritmos y los resultados obtenidos para la imagen Lena, 256×256 píxeles, 8 bits/píxel : el histograma de Lena, la ecualización de ese histograma y la imagen de Lena con mayor contraste.

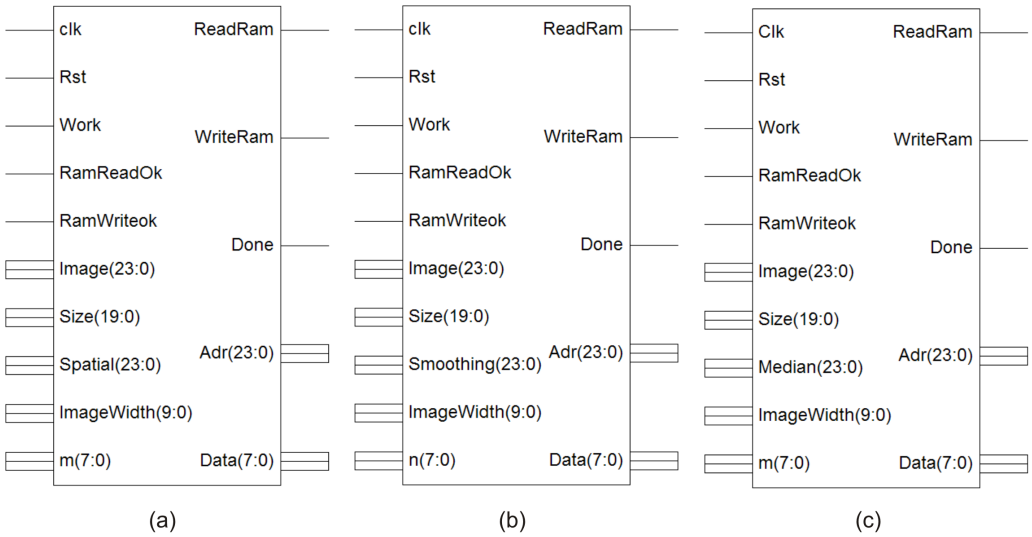


Figura 5.17: (a) Símbo del algoritmo de filtrado espacial. (b) Símbolo del algoritmo de suavizado direccional. (c) Símbolo del algoritmo de filtrado de mediana.

5.5. Algoritmos de filtrado espacial, suavizado direccional y filtrado de mediana

El filtrado espacial, el suavizado direccional y el filtrado de mediana son tres técnicas de mejora de la imagen que buscan eliminar efectos no deseados en las imágenes, como son toda clase de ruido: sal y pimienta, gaussiano, impulsional, etc. La similitud que existe entre estos filtros es que pertenecen al dominio espacial, y eliminan el ruido en las imágenes de acuerdo a plantillas de píxeles vecinos en la imagen; se realizan operaciones en el conjunto de vecinos al píxel actual de acuerdo a dicha plantilla, Secciones 2.3.1.1.4, 2.3.1.1.5 y 2.3.1.1.6.

En el caso del filtrado espacial, la plantilla indica un promediado entre los vecinos y el píxel que se está procesando; el resultado de este promediado se almacena en la imagen resultante en la misma posición. Para el suavizado direccional es distinto, la plantilla indica un promediado direccional, es decir, se establecen cuatro direcciones posibles que atraviesan por el centro, considerando al píxel actual y la diferencia menor del promedio en las direcciones, el nuevo valor del píxel es el resultante de esta operación. Similar como ocurre en el filtrado espacial se realiza el filtrado de mediana, sólo que en lugar de promediar el conjunto de píxeles vecinos se calcula la mediana entre ellos. Las diferencias entre ellos se hacen notorias cuando son aplicados ante imágenes con distintas clases de ruido.

Los símbolos obtenidos a partir de la implementación del *algoritmo de filtrado espacial*, el *algoritmo de suavizado direccional* y el *algoritmo de filtrado de mediana* se muestran en la Figura 5.17.

En las Figuras 5.18, 5.19 y 5.20 se muestran las interfaces diseñadas para el *algoritmos de el filtrado espacial*, el *algoritmo de suavizado direccional* y el *algoritmo de filtrado de mediana* respectivamente, además se muestran los resultados de estos filtros aplicados a Lena de 128×128 píxeles y 8 bits/píxel, con plantillas de 3×3. Para el algoritmo de filtrado espacial se introduce ruido *gaussiano* (5 %) a la imagen de Lena y el PSNR de la imagen resultante es de 20.70. Al igual que en el filtrado espacial, en el *algoritmo de suavizado direccional* se aplica ruido *gaussiano* (5 %) a Lena, y el PSRN fue de 23.54, mejor que el obtenido en el filtrado espacial. Para el *algoritmo de filtrado de mediana* se introduce ruido *sal y pimienta* (5 %) a Lena, y el PSNR obtenido de la imagen resultante es de 22.69, Figura 5.20.

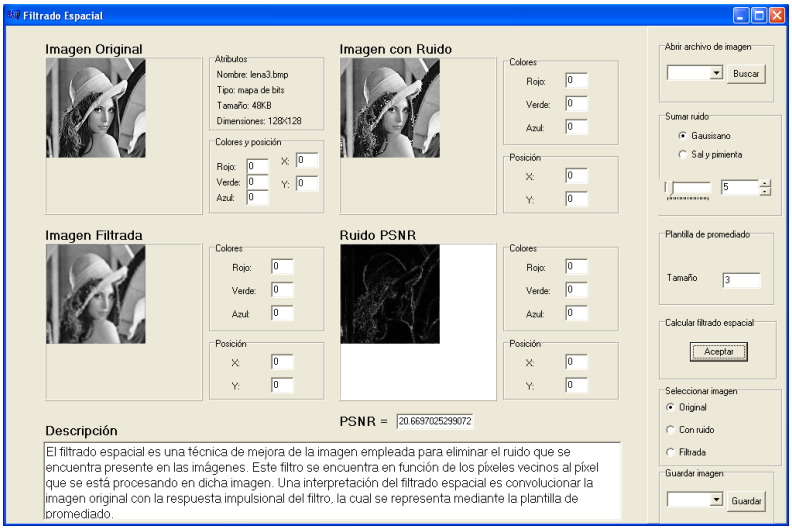


Figura 5.18: Interfaz del algoritmo de filtrado espacial y resultados sobre la imagen Lena.

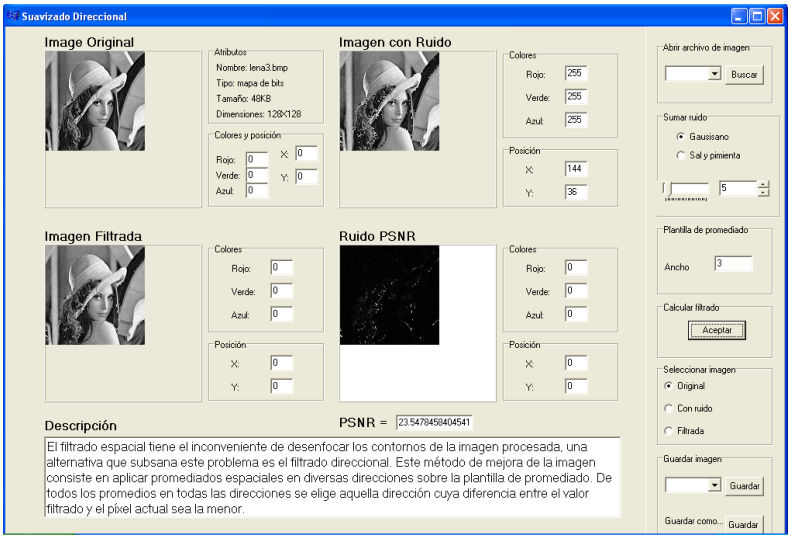


Figura 5.19: Interfaz del algoritmo de suavizado direccional y resultados sobre la imagen Lena.

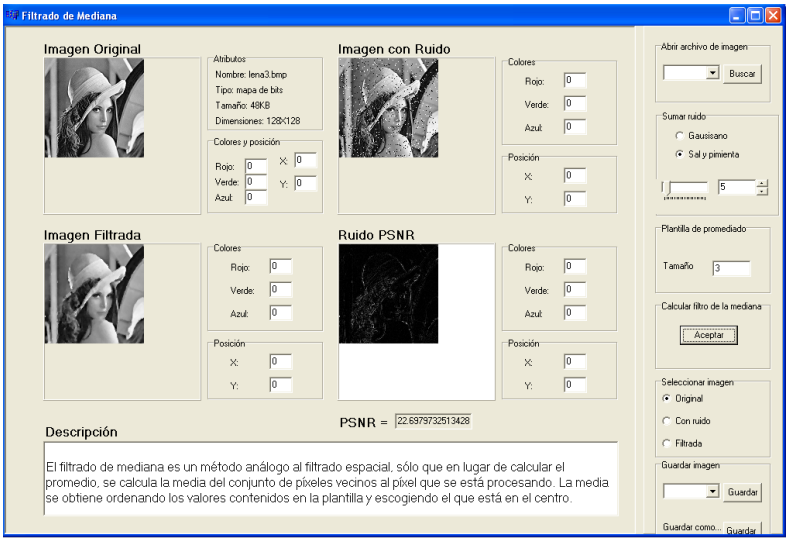


Figura 5.20: Interfaz del algoritmo de filtrado de mediana y resultados sobre la imagen Lena.

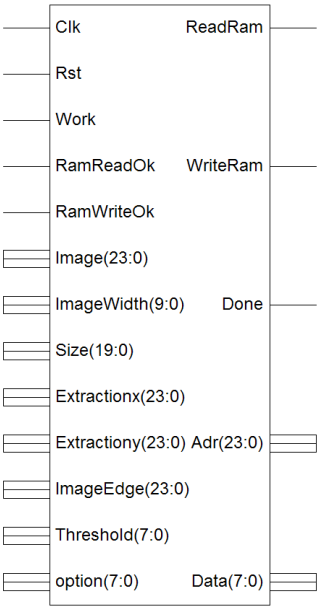


Figura 5.21: Símbolo del algoritmo de detección de contornos.

5.6. Algoritmo de detección de contornos por Roberts y Sobel

La detección de contornos es una técnica de análisis de imágenes que entra en la categoria de la segmentación de la imagen. Su objetivo principal es encontrar regiones en la imagen que posean transiciones de claro a oscuro o viceversa, a estas transiciones se les conoce como contornos de la imagen. De los principales métodos para calcular los contornos en las imágenes se encuentran las técnicas de Roberts y Sobel, Sección 2.3.4.1. En la Tabla 2.1 se muestran las plantillas Roberts y Sobel, las cuales basan su funcionamiento en el cálculo del gradiente de un conjunto de píxeles de la imagen. En ambos casos, primero se calcula el gradiente horizontal de la imagen, luego se genera el gradiente vertical de la imagen, y para finalizar se obtiene la imagen de contornos; a partir de un umbral proporcionado por el usuario, se suman las magnitudes de los gradientes vertical y horizontal, y se compara esta suma con el umbral elegido de acuerdo a la Ecuación 2.53, declarándose contornos a aquellos píxeles mayores al umbral.

Se ha mencionado que los contornos de la imagen son transiciones de claro a oscuro o viceversa, pero también se sabe que son aquellas frecuencias altas que se encuentran en la imagen, y cuando se aplican las técnicas de Roberts y Sobel basadas en el gradiente, lo que se aplica a la imagen son filtros pasa-altas; la mayor parte de la energía está contenida en las frecuencias bajas de la imagen.

Como resultado de la implementación del *algoritmo de detección de contornos* (mediante Roberts y Sobel) en el **procesador de imágenes**, se obtiene el símbolo del mismo, Figura 5.21.

La interfaz para el *algoritmo de detección de contornos* se muestra en la Figura 5.22. En ella también se aprecian los resultados obtenidos al aplicar la detección de contornos mediante Roberts en la imagen Lena, 256×256 píxeles, 8 bits/píxel; con el umbral de detección puesto en 40. Por otro lado, en la Figura 5.23 se muestran los resultados obtenidos de aplicar la detección de contornos a la imagen de Lena, pero ahora haciendo uso de la técnica de Sobel con un umbral de detección puesto en 115.

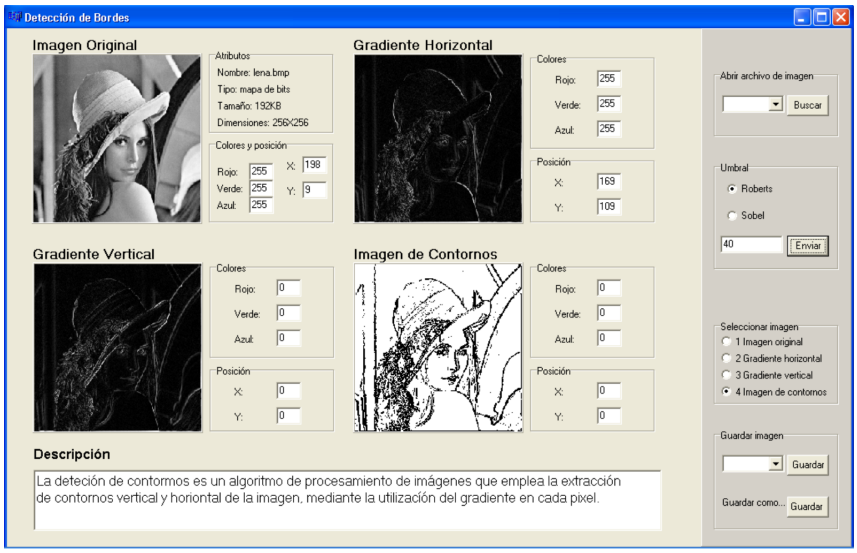


Figura 5.22: Interfaz del algoritmo de detección de contornos mediante Roberts y los resultados obtenidos para la imagen Lena.

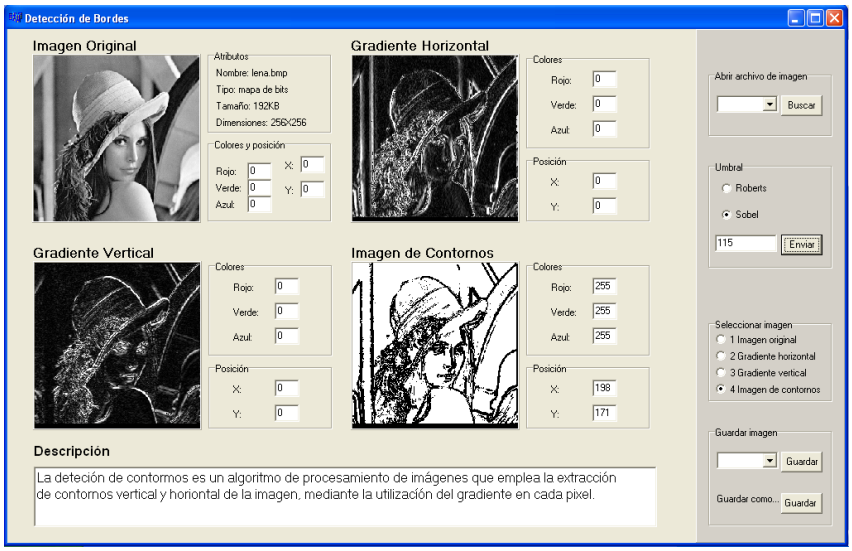


Figura 5.23: Resultados obtenidos de la detección de contornos mediante Sobel de la imagen Lena.

6. CONCLUSIONES Y PERSPECTIVAS

Ente capítulo contiene las conclusiones a las que se ha llegado después que el sistema de procesamiento y análisis digital de imágenes sobre los recursos de un FPGA ha sido terminado y probado en la evaluación de algunos algoritmos. Finalmente se proponen las perspectivas o trabajos futuros que le darán seguimiento a este trabajo.

6.1. Conclusiones

1. Un estudio bibliográfico del estado del arte, referente a herramientas enfocadas a la evaluación de algoritmos de procesamiento y/o análisis de imágenes sobre un CDC, permitió conocer las características que debe reunir una herramienta de este tipo para que pueda ser considerada competitiva.

2. El diseño, modelado e implementación de un procesador de aplicación especifica, enfocado al procesamiento y análisis de imágenes, sobre un FPGA maneja una arquitectura abierta, es decir, el procesador fue diseñado para facilitar el modelado e integración de nuevos algoritmos de una manera simple; además demostró un alto desempeño durante

la ejecución de éstos, al manejar estructuras concurrentes dentro de los recursos del FPGA permitiendo alcanzar velocidades de procesamiento elevadas en la ejecución del algoritmo elegido, por ejemplo, para una imagen de 256×256 píxeles (8 bits/píxel) a una frecuencia de operación del FPGA de 50MHz, el algoritmo de la DWT de un nivel consume 11.79 ms (este tiempo también incluye los tiempos de acceso al sistema de memoria).

3. El uso de la metodología descendente en el diseño y modelado del procesador de imágenes, permitió implementar una arquitectura modular, este hecho generó código reutilizable, fácilmente modificable y una arquitectura a la que fácilmente se le pueden integrar nuevos módulos.

4. El uso de un lenguaje estandarizado como lo es VHDL en la descripción del Procesador de imágenes concede la ventaja de volverlo compatible con otras tecnologías de FPGAs y portable hacia herramientas CAD de distintas compañías.

5. En el diseño y codificación de la interfaz de usuario se usó el paradigma de programación orientado a objetos y se empleó la metodología de diseño de aplicaciones GUI *Bridge*, la cual divide el diseño en cuatro fases hasta llegar a un documento de interfaz que contiene elementos visuales para la interacción con el usuario. El seguir esta metodología en el diseño de la interfaz de usuario optimizó el tiempo requerido en el desarrollo de ésta; además, la aplicación resultante presentó una mayor eficiencia en el uso de recursos y el tiempo de procesamiento en comparación con una primera versión que no utilizó dicha metodología.

6. La metodología propuesta, la cual es utilizada en la interacción de la "interfaz de usuario" y el "sistema hardware para el procesamiento de imágenes", facilita al usuario la integración y evaluación de nuevos algoritmos sobre la "herramienta para el procesamiento y análisis de imágenes sobre un FPGA".

7. La principal aportación de este trabajo de tesis, una herramienta enfocada a la evaluación de algoritmos de procesamiento de imágenes en un CDC, optimiza el tiempo de desarrollo de aplicaciones de procesamiento y/o análisis de imágenes sobre un FPGA. La herramienta está orientada para ser usada por personas que laboran en los ámbitos académicos y de investigación.

8. Finalmente, como una aportación adicional de este trabajo de tesis se han diseñado, modelado e integrado, al procesador de imágenes, algunos algoritmos de uso frecuente en el procesamiento y/o análisis de imágenes, los cuales pueden ser consultados en la sección de resultados de este trabajo de tesis

6.2. Perpectivas

- La herramienta obtenida en este trabajo de tesis implementa algoritmos únicamente sobre imágenes fijas en tonos de gris, de este hecho se desprenden dos trabajos futuros:

  - Ampliar el campo de trabajo de la herramienta a imágenes de colores.
  - Ampliar el campo de trabajo de la herramienta a secuencias de imágenes (video).

- Con la finalidad de comparar características de rendimiento, se propone el modelado e implementación de diversas arquitecturas básicas de procesadores y su integración a la diseñada.

A. LA TARJETA NEXYS-2

La tarjeta Nexys-2 es una plataforma para el diseño de circuitos y sistemas digitales, completa y lista para usarse. Se encuentra basada en el FPGA Spartan3E-500 de la compañía Xilinx. Por sus componentes que la conforman es ideal para todo tipo de sistemas digitales, incluyendo sistemas de procesadores embebidos basados en el MicroBlaze de Xilinx. La Nexys-2 puede almacenar incontables sistemas digitales basados en FPGAs, y los diseños pueden ir creciendo mas allá de las capacidades de la tarjeta, usando conectores de expansión. Se le pueden agregar otras características como control de motores, conversiones Analógica/Digital y Digital/Analógica, circuitos de audio, e interfaces de sensores-actuadores, todo esto a través de sus conectores para módulos periféricos de 8 bits. La tarjeta Nexys-2 es compatible con todas las versiones de las herramientas Xilinx ISE. En la Figura A.1 se muestra el diagrama de bloques de la tarjeta Nexys-2.

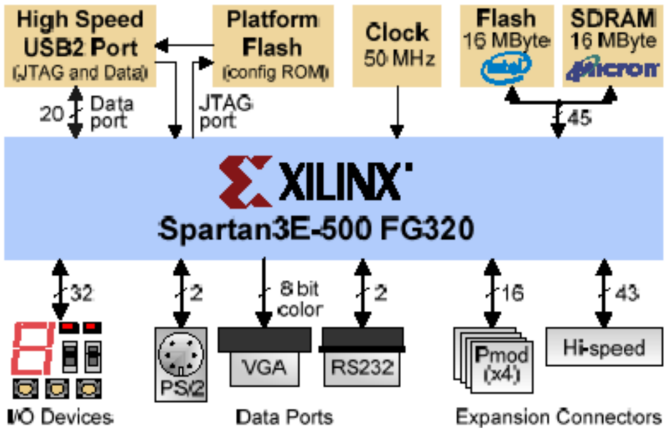


Figura A.1: Diagrama de bloques de la plataforma Nexys-2.

Alimentación	Dispositivo	Corriente(máx/típ)
3.3V principal	IC6: LTC1765	3A/100mA
2.5V FPGA	IC7: LTC3417	1.4A/50mA
1.2V FPGA	IC7: LTC3417	1.4A/200mA
1.8V SRAM	IC5: LTC1844	150Ma/90mA
3.3V USB	IC5: LTC1844	150Ma/60mA

Tabla A.1: Fuentes de alimentación para la Nexys-2.

A.1. Fuente de alimentación

La fuente de alimentación para la tarjeta Nexys-2 puede provenir del cable del USB, de 5-15VCD de un regulador de voltaje, o de un conjunto de baterías. Un *jumper* selecciona la fuente de la cual proviene la alimentación. La alimentación principal maneja un regulador de voltaje de 3.3V, aunque hay dispositivos que requieren 2.5, 1.8 y 1.2V, estos voltajes son alcanzados con reguladores. La alimentación pricipal es generada a partir de un regulador de interrupción de alta eficiencia.

La corriente total que es consumida por la tarjeta depende de la configuración del FPGA, de la frecuencia de reloj y las conexiones externas; la corriente se incrementará mientras más extensos sean los circuitos configurados en el FPGA. En la Tabla A.1 se tiene una descripción de los voltajes de algunos dispositivos de la Nexys-2, y corrientes que consumen.

También puede recibir o entregar voltaje a través de los conectores Pmod. Un USB cliente puede proporcionar sólo 500mA. Cuando la tarjera Nexys-2 se alimenta por medio del USB, se debe tener cuidado de no sobrepasar esta corriente para no dañar al cliente USB, por lo regular la Nexys-2 usa 300mA del USB y sobran 200mA para periféricos.

A.2. Configuración del FPGA y de la plataforma Flash

El FPGA de la tarjeta Nexys-2 debe de ser configurado o programado por el usuario antes que este desempeñe alguna función. Durante la configuración, un archivo de *bit* es transferido en las células de memoria dentro del FPGA para definir las funciones lógicas y la interconexión de circuitos. El FPGA puede ser programado de dos formas: directamente de la PC usando el puerto USB de la Nexys-2, y desde la ROM Flash, también programable desde el puerto USB. A tráves de un jumper de la tarjeta se determina cual fuente se usará para cargar su configuración. Cuando se elige la configuración del FPGA a través de la plataforma Flash, la configuración es cargada automáticamente en el ciclo de encendido. El software *Adept Suite* gratuito es usado para configurar el FPGA y su plataforma Flash a

partir de un archivo de bit ubicado en la PC. La configuración de PC durará hasta un reset por ciclo de encendido o por un reset en el FPGA.

La tarjeta Nexys-2 incluye un reloj de 50MHz y un zócalo para un segundo oscilador. Ambas señales de reloj se conectan a los pines de entrada de reloj global en el FPGA. Los sintetizadores de reloj, o también llamados DLLs, duplican o cuatriplican la frecuencia de entrada, dividiendo la frecuencia de entrada por un entero, y definiendo fase precisa y relaciones de retardo entre varias señales de reloj.

A.3. Entradas/salidas de usuario

La Nexys-2 include dispositivos de entrada, de salida, y puertos de datos, para implementar diseños sin la necesidad de otros componentes. Contiene 4 pushbuttons y 8 interruptores con resistores de protección contra corto circuitos. Para las salidas de circuitos se cuenta con 8 LEDs tipo ánodo, los cuales son controlados desde el FPGA, además se tiene un LED de encencido, y otro LED que indica el estado del FPGA. Y contiene displays de 7 segmentos para cuatro dígitos, el control de cada diplay se logra a partir de su respectivo cátodo.

A.4. Puerto USB

La Nexys-2 incluye un puerto USB2 de alta velocidad basado en el controlador USB Cypress CY7C68013A. Puede ser usado para programar los dispositivos del dispositivo Xilinx, para transferencia de datos de usuario arriba de los 38Mbytes/segundo y para alimentar a la Nexis-2. Para la programación del dispoitivo se usa el programa de Digilent, el *Adept Suite*, también para transferencia de datos de usuario, aunque se pueden implementar programas para que el usuario transfiera datos con ayuda de la API proporcionada por Digilent para la conexión con el USB de la Nexys-2.

A.5. Puerto PS/2

Contiene un conector mini-DIN de 6 pines que se pueden usar como ratón PS/2 y como teclado. Ambos usan un arreglo de dos bits, señal de reloj y de datos, para comunicarse con un cliente. Incluyen na palabra de 11 bits que incluye un inicio, paro y paridad impar, pero los paquetes están organizados de manera distinta, y la interfaz del teclado pemite transferencia de datos bi-direccional.

A.6. Puerto VGA

La tarjeta Nexys-2 usa 10 señales del FPGA para crear un puerto VGA con colores de 8 bits y dos señales de sincronización estándar, horizontal y vertical. Las señales de color usan circuitos divisores de resistencias que trabajan en conjunto con las resistencias terminales del display VGA, creando 8 niveles de señales en las señales VGA roja y verde, y cuatro en azul. 256 distintos colores pueden ser mostrados. Un circuito controlador de video debe ser creado en el FPGA para controlar las señales de sincronización y color con el correcto tiempo para producir un sistema de display.

A.7. Puerto serial

Contiene dos señales (RXD y TXD) destinadas al puerto serial basado en el convertidor de voltaje ST Microelectronics ST3232, que convierte los dos niveles de las señales usadas por la comunicación RS-232 (de 12 a -3 para un '1' y de 12 a 3 para un '0') a señales de 3.3v usados por el FPGA. El puerto serial de la Nexys-2 es útil para varias aplicaciones y en particular para trabajar con el procesador embebido Xilinx MicroBlaze.

A.8. Memoria

La tarjeta Nexys-2 posee dos dispositivos de memoria externa: RAM y ROM. Por un lado se encuetra la RAM externa de 128Mbit Micron M45W8MW16 Cellular RAM pseudo-estática DRAM, la cual se encuentra organizada como 8Mbytes×16bits. Puede ser tratada como SRAM asíncrona con tiempos de los ciclos de lectura y escritura de 70ns o como memoria síncrona con un bus de 80MHz. La ROM externa es la 128Mbit Intel TE28F128J3D75-110 StrataFlash, organizada como 8Mbytes×16bits, internamente contiene 128 bloques que pueden ser borrados individualmente, y esta memoria soporta tiempos para el ciclo de lectura de 110ns. Cuenta con un buffer interno de 32bits que se escibe en 70ns y este escribe a la Flash en 218μs. Ambos dispositivos comparten el bus de datos de 16bits y el bus de direcciones de 24bits. La RAM es direccionable por byte y cada memoria cuenta con señales de control para accesarlas individualmente.

A.9. Conectores perféricos y conectores de expansión

La tarjeta Nexys-2 posee cuatro conectores periféricos Pmod de 12 pines, de los cuales 8 son señales de datos, dos pines de GND (tierra) y dos de VDD (alimentación), con protección anti-corto circuito. También incluye un conector Hirose FX-2 de alta densidad de 100 pines, disponible para manejar tarjetas periféricas con velocidaddes de señales arriba de los 100MHz.

B. ESQUEMA DEL PROCESADOR DE IMÁGENES MODELADO

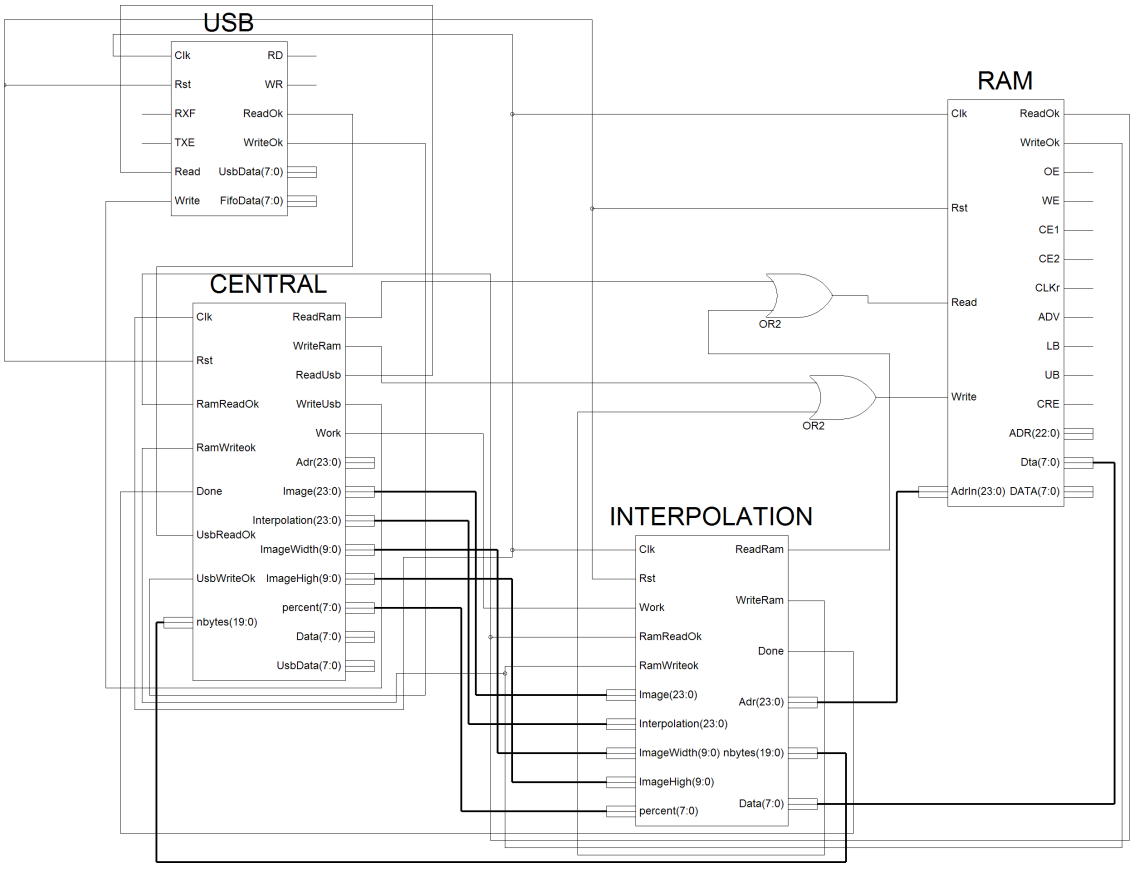


Figura B.1: Conexiones entre los módulos que conforman al procesador de imágenes modelado en la Fase 1 del desarrollo de este trabajo de tesis.

BIBLIOGRAFÍA

[1] Acharya, T., A. K. Ray. *Image Processing. Principles and Applications*. ISBN-13 978-0-471-71998-4, USA: John Wiley & Sons, 2005.

[2] Ahmed, N., et al. "Discrete Cosine Transform", *IEEE Transaction on Computers*, *Vol. 23*(1):90–93 (1974).

[3] Alvarado, S. *Diseño y Modelado de una Arquitectura VLSI para el Reconocimiento de Imágenes en un Sistema de Visión Artificial con base en la Transformada Discreta Wavelet y Memorias Asociativa Morfológicas*. Tesis de licenciatura, Universidad Tecnológica de la Mixteca, 2007.

[4] Arai, Y., et al. "A Fast DCT-SQ Scheme for Image", *Transactions of the IEICE*, *Vol. E 71*(11):1095–1097 (1988).

[5] Asensi, S. C. *IPXS v2.2 User guide*. Depto. Tecnología Informática y Computación. Universidad de Alicante, España, 2000.

[6] Bankman, I.Ñ. *Handbook of Medical Imaging. Processing and Analysis*. ISBN 0-12-077790-8, USA: Academic Press, 2000.

[7] Barr, M. *Programming Embedded Systems in C and C++*. ISBN: 1-56592-354-5, OReilly, 1999.

[8] Barr, M., A. Massa. *Programming Embedded Systems*. ISBN: 0-596-00983-6, O Reilly, 2006.

[9] Bövik, A. *Handbook of Image and Video Processing. Principles and Applications*. ISBN 0-12-119790-5, USA: Academic Press, 2000.

[10] Chan, S. C., et al. "A programmable image processing system using FPGA", *Dept. of Electron. Eng., City Polytech. of Hong Kong, Kowloon, Hong Kong*, *Vol. 2*:125–128 (1994).

[11] Chen, W. H., et al. "A fast Computational Algoritnm for the Discrete Cosine Transform", *IEEE Transactions on Communications*, *Vol. COM-23*:1004–10009 (1977).

[12] Cloutier, J., et al. "VIP: An FPGA-based Processor for Image Processing and Neural Networks", *microneuro*, *Vol. 0*:330 (1996).

[13] Cosman, P., et al. *Handbook of Medical Imaging Processing and Analysis*. ISBN 0-12-077790-8, USA: Academic Press, 2000.

[14] Eckel, B. *Thinking in C ++, 1*. ISBN 0-13-979809-9. Prentice Hall, 2000.

[15] Edward, A. *Implementation of image processing algorithms on FPGA hardware*. Master of science, Faculty of the Graduate School of Vanderbilt University, 2000.

[16] Fourier, J. B. J. *Théorie Analytique de la Chaleur*. 1822.

[17] Ghanbari, M. *Standard Codecs: Image Compression to Advanced Video Coding*. ISBN 0-85296-710-1, The Institution of Electrical Engineers, 2003.

[18] Gonzalez, R. C., R. E. Woods. *Digital Image Processing Using MATLAB*. ISBN 81-7758-898-2, USA: Pearson Educational, 2006.

[19] Gonzalez, R. C., R. E. Woods. *Digital Image Processing*. ISBN 81-203-2758-6, USA: PRENTICE HALL, 2002.

[20] Grimblatt, V. *Sistemas embebidos Semiconductor Products Sector*. Motorola Electro Industria, Abril 2002.

[21] Guzmán, E. *Compresión de imágenes mediante memorias asociativas*. Tesis de Doctorado, Instituto Politécnico Nacional, 2008.

[22] Hsieh, C.-F., et al. "Implementation of an Efficient DWT Using a FPGA on a Real-time Platform", *ICICIC '07: Proceedings of the Second International Conference on Innovative Computing, Informatio and Control*, 235 (2007).

[23] ISO/IEC-IS10918-1—CCITT-T.81. "Digital Compression and Coding of Continuos-Tone Still Image", *ISO/IEC 1992* (1992).

[24] Jahne, B. *Digital Image Processing. 5th and extended edition*. ISBN 3-540-67754-2, Germany: Springer, 2002.

[25] Khambete, M., M. Jhosi. "Blur and Ringing Artifact Measurement in Image Compression using Wavelet Transform", *Proceeding of World Academy of Science, Engineering and Technology*, *Vol. 20*:183–186 (2007).

[26] Lewis, D. W. *Fundamental of Embedded Software Where C and Assembly Meet*. ISBN: 0-13-061589-7, Prentice Hall, 2002.

[27] Mallat, S. G. "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *11*(7):674–693 (1989).

[28] Mandado, E., et al. *Dispositivos Lógicos Programables y sus aplicaciones*. ISBN 84-9732-054-9, Madrid, España: Thomson, 2002.

[29] Medany, E., W. M. *Computational Technologies in Electrical and Electronics Engineering, 2008*, 106–109 (2008).

[30] Meyer-Baese, U. *Digital Signal Processing with Field Programmable Gate Arrays*. ISBN 3-540-21119-5, USA: Springer, 2004.

[31] Navabi, Z. *Video Codec Design, Developing Image and Video Compression Systems*. ISBN: 0-471-48553-5, Great Britian: John Wiley and Sons, 2003.

[32] Noergaard, T. *Embedded Systems Arquitecture. A Comprehensive Guide for Engineers and Programmers*. ISBN: 0-7506-7792-9, ELSEVIER, 2005.

[33] Palnitkar, S. *Verilog HDL: A Guide to Digital Design and Synthesis*. ISBN: 0-13-044911-3, Madrid: Prentice Hall PTR, 2003.

[34] Pardo, F., J. A. Boluda. *VHDL Lenguaje para síntesis y modelado de circuitos*. ISBN 84-7897-595-0, España: Editorial RAMA, 2004.

[35] Parnell, K., N. Mehta. *Programmable Logic Design Quick Start Handbook*. USA: Xilinx, 2003.

[36] Pellerin, D., D. Taylor. *VHDL Made Easy!*. ISBN 0-13-650763-8, USA: Prentice Hall PTR, 1997.

[37] Petrou, M., P. Bosdogianni. *Image Processing: the fundamentals*. ISBN 0-471-99883-4, USA: John Wiley and Sons, 1999.

[38] Pierre-Deschamps, J. *Synthesis of Arithmetic Circuits: FPGA, ASIC, and Embedded Systems*. ISBN-13 978-0471-68783-2, John Wiley & Sons, 2006.

[39] Pérez, M., et al. "STREAM, Un procesador basado en FPGA para el tratamiento de secuencias de imágenes: Aplicación a la estéreo visión densa", *Cuarto Encuentro Internacional Mexicano de Ciencias de la Computación (ENC03)*, *Vol. 1*:77–81 (2003).

[40] Pérez, S. A., et al. *Diseño de Sistemas Digitales con VHDL*. Madrid España: Ed. Thomson, 2002.

[41] Rosas, R. L., et al. "SIMD architecture for image segmentation using Sobel operators implemented in FPGA technology", *Electrical and Electronics Engineering, 2005 2nd International Conference on*, 77– 80 (2005).

[42] Sangiovann, A., G. Martin. "Platform-Based Design and Software Design Platform-Based Design and Software Design Methodology for Platform-Based Design and Software Design Methodology for Embedded Systems", *IEEE Design & Test of Computers* (2001).

[43] std 1076-1993, I. "IEEE Standard VHDL Language Referente Manual", *ANSI/IEEE Std 1076-1993*, *Vol. SH16840* (June 1994).

[44] std 1076-1999, I. "IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis", *Design Automation Standards Committee of the IEEE Computer Society* (1999).

[45] std 1364-1995, I. "IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language", *Design Automation Standards Committee of the IEEE Computer Society* (1995).

[46] std 1364.1-2002, I. "IEEE Standard for Verilog Register Transfer Level Synthesis", *Design Automation Standards Committee of the IEEE Computer Society* (2002).

[47] Symes, P. *Compression Demystified*. ISBN 0-07-136324-6, USA: Ed. Mc Graw Hill, 2001.

[48] Uzun, I. S., et al. "FPGA implementations of fast Fourier transforms for real-time signal and image processing", *Sch. of Comput. Sci., Queen's Univ. of Belfast, UK*, *Vol. 152*(3):283– 296 (2005).

[49] Wallace, G. K. "The JPEG still picture compression standard", *Proceedings of Communications of the ACM*, *Vol. 34*(4):30–44 (1991).

[50] Winograd, S. "On Computing the Discrete Fourier Transform", *Proceeding of the National Academy of Sciences of the United States of America*, *Vol. 73*(4):1005–1006 (1976).

[51] Wood, L. E. *User Interface Design*. ISBN: 0849331250, CRC Press, 1997.

[52] Zanuy, M. F. *Tratamiento Digital de Voz e Imagen*. ISBN 970-15-0651-0, USA: Alfaomega grupo editor, 2001.